AVANCES EN ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

20 aniversario sarteco



SARTECO 2026 MADRID

Editado por:

Javier García Blas
Jesús Carretero Pérez
Arturo González-Escribano

ARCOS COMPUTER ARCHITECTURE, COMMUNICATIONS AND SYSTEMS RESEARCH GROUP

uc3m | Universidad Carlos III de Madrid

sarteco

Avances en arquitectura y tecnología de computadores

Actas de las Jornadas SARTECO 2026

Madrid, 17 al 19 de Junio de 2026

Avances en arquitectura y tecnología de computadores
Actas de las Jornadas SARTECO 2026

Editores: Javier Garcia Blas Jesús Carretero Pérez Arturo González-Escribano

(c) 2026, Jornadas SARTECO, - Universidad Carlos III de Madrid

ISBN: 978-84-10132-31-3

Madrid, 2026



Entidades patrocinadoras

PLATINO

NEMIX®
TECHNOLOGY & EXPERIENCE

SUPERMICRO®

ORO

COBER

PLATA

Universidad Carlos III de Madrid
DEPARTAMENTO DE INFORMÁTICA

Prólogo

La Sociedad de Arquitectura y Tecnología de Computadores (SARTECO) presenta, una vez más, las Jornadas SARTECO, las cuales integran las XXXVI Jornadas de Paralelismo (JP2026) y las X Jornadas de Computación Empotrada y Reconfigurable (JCER2026).

Además, en el contexto de las jornadas SARTECO se realizarán las siguientes actividades:

- Décima edición del Concurso "Tu Tesis en Tres Minutos" ("Your PhD Thesis in Three Minutes" competition) que pretende premiar los mejores trabajos de tesis recientes en el área
- Sexta edición del Concurso "Tu Trabajo Fin de Máster en Tres Minutos" que pretende hacer lo propio con los mejores Trabajos Fin de Máster.
- *Tutorial 1:* "CREATOR". Impartido por Diego Camarmas-Alonso, Félix García-Carballeira y Alejandro Calderón-Mateos.
- *Tutorial 2:* "Cómo ganar experiencia en RISC-V con el proyecto BZL". Impartido por Miquel Moretó, Xavier Martorell e Iván Fernández.
- *Tutorial 3:* "FIDESlib 2.0: Computación Privada Acelerada". Impartido por Carlos Agulló-Domingo, Óscar Vera-López, Adrián Nicolás-Aguilera y José L. Abellán.
- Tres *keynotes* impartidas por D. Alberto Ros (Univeridad de Murcia), D. Felix Wolf (TU Darmstadt) y D. Daniel Piso (ARM).

En esta edición de las Jornadas contamos con un excelente programa de sesiones técnicas con un total de 112 artículos repartidos entre 89 en las JP y 23 en las JCER. A fecha de hoy, el número de asistentes sin incluir los voluntarios es de mas de 190 personas.

Aprovecho también para mostrar nuestro sincero agradecimiento a nuestros sponsors por su generosa contribución y apoyo.

La celebración conjunta de estas actividades y eventos constituye un referente nacional imprescindible para la comunidad científica agrupada en SARTECO. Estas jornadas facilitan colaboraciones y sinergias entre grupos y dan a nuestra comunidad una mayor visibilidad social para potenciar nuestras oportunidades de transferencia tecnológica a la industria.

Desde la organización de estas nuevas Jornadas SARTECO, os deseamos a todos y a todas, un fructífero trabajo y una placentera estancia en Madrid.

Javier García Blas y Jesús Carretero
30 de mayo de 2026
Madrid

Comités

Comité de Dirección de las Jornadas SARTECO

Víctor Viñals Yúfera (UNIZAR, Presidente)
Katzalin Olcoz Herrero (UCM, Vicepresidente)
Gracia Ester Martín Garzón (UAL, Secretaria)
Francisco Tirado Fernández (UCM, Presidente de Honor)

Comité de Organización de las Jornadas SARTECO

Jesús Carretero. Universidad Carlos III de Madrid.
Javier García Blas. Universidad Carlos III de Madrid.

Comité de Programa JP'2026

Félix García Carballeira. Universidad Carlos III de Madrid.
David E. Singh. Universidad Carlos III de Madrid.
Alejandro Calderón Mateos. Universidad Carlos III de Madrid.
Javier Fernández Muñoz. Universidad Carlos III de Madrid.
Alberto Cascajo García. Universidad Carlos III de Madrid.
Dante Domizzi Sánchez Gallegos. Universidad Carlos III de Madrid.
Diego Camarmas Alonso. Universidad Carlos III de Madrid.
Elías del Pozo. Universidad Carlos III de Madrid.
Diana Elisabeth Carrizales. Universidad Carlos III de Madrid.
Genaro Juan Sánchez Gallegos. Universidad Carlos III de Madrid.
Catherine Alessandra Torres Charles. Universidad Carlos III de Madrid.
Cosmin Octavian Petre Carastoian. Universidad Carlos III de Madrid.
Juan Carlos Cano Resa. Universidad Carlos III de Madrid.
Paula Sánchez Checa. Universidad Carlos III de Madrid.
Dario Muñoz Muñoz. Universidad Carlos III de Madrid.

Comité de Coordinación JCER'2026

Jesús Carretero. Universidad Carlos III de Madrid.
Javier García Blas. Universidad Carlos III de Madrid.

Comité de Programa JCER'2026

Félix García Carballeira. Universidad Carlos III de Madrid.
David E. Singh. Universidad Carlos III de Madrid.
Alejandro Calderón Mateos. Universidad Carlos III de Madrid.
Javier Fernández Muñoz. Universidad Carlos III de Madrid.
Alberto Cascajo García. Universidad Carlos III de Madrid.
Dante Domizzi Sánchez Gallegos. Universidad Carlos III de Madrid.
Diego Camarmas Alonso. Universidad Carlos III de Madrid.
Elías del Pozo. Universidad Carlos III de Madrid.
Diana Elisabeth Carrizales. Universidad Carlos III de Madrid.
Genaro Juan Sánchez Gallegos. Universidad Carlos III de Madrid.
Catherine Alessandra Torres Charles. Universidad Carlos III de Madrid.
Cosmin Octavian Petre Carastoian. Universidad Carlos III de Madrid.
Juan Carlos Cano Resa. Universidad Carlos III de Madrid.
Paula Sánchez Checa. Universidad Carlos III de Madrid.
Dario Muñoz Muñoz. Universidad Carlos III de Madrid.

Comité T3M26 y TFM3M26

Víctor Viñals Yúfera (UNIZAR, Presidente)
Katzalin Olcoz Herrero (UCM, Vicepresidente)
Gracia Ester Martín Garzón (UAL, Secretaria)
Dora Blanco Heras (Coordinadora, USC)
Vicente J. Blanco Pérez (Coordinador, ULL)

Indice

Ponencias invitadas

A Journey from Cache Coherence to Microarchitecture 3
Alberto Ros

The Price Performance of Performance Models 5
Felix Wolf

La arquitectura de Arm: como trasladar ideas y conceptos a millones de dispositivos 7
Daniel Piso Fernández

Parte 1.- XXXVI Jornadas de Paralelismo

Arquitecturas del procesador, multiprocesadores y chips multinúcleo

Memoria transaccional hardware in-core de bajo coste 13
Álvaro Rubira-García, Eduardo José Gómez-Hernández, Rubén Titos-Gil, Alberto Ros

Organización del Backend y Rendimiento en Procesadores RISC-V Fuera de Orden 21
Esther Alonso García, Pablo Abad Fidalgo, Pablo Prieto Torralbo, Valentín Puente Varona

Coste computacional de una red de aprendizaje profundo basada en EfficientNet sobre dispositivos basados en GPU para computación en el borde 31
Nicolás Vilela-Pérez, Dora B. Heras, Francisco Argüello

Infraestructura para el modelado de políticas de planificación multihilo SMT en el simulador gem5 39
Manuel Antonio Pecino Fernández, Daniel Escribano Romero, Salvador Petit, Julio Sahuquillo

Redes de comunicaciones (1/2)

Análisis del impacto de la comunicación intra-nodo en el rendimiento de las redes de interconexión de sistemas para HPC e IA 51
Joaquin Tarraga-Moreno, Jesús Escudero-Sahuquillo, Pedro Javier García, Francisco J. Quiles

Metodología para la visualización en tiempo real de simulaciones de SimGrid 61
Manuel Gómez-Plana Rodríguez, Félix García Carballeira

Infraestructura P2P Agnóstica para Streaming Multimedia Seguro en Entornos Clínicos: Soberanía del Dato mediante Virtualización IA On-Premise 69
Miguel Ángel Sánchez Olmedo, Savíns Puertas-Martín, Alejandra De Torres Sánchez, Pedro Mezquita Raya, Juana López Redondo

Modelado y simulación de la red de adquisición de datos de ATLAS para el HL-LHC 77
Carlos Medrano Navalón, Mikel Eukeni Pozo Astigarraga, Gabriel Gómez López, Jesús Escudero-Sahuquillo, Pedro Javier García, Francisco J. Quiles

Tecnologías clúster, plataformas distribuidas, BigData y Deep Learning (1/4)

Modelando la gestión de datos en el cómputo continuo mediante grafos de conocimiento 89
Dante D. Sánchez-Gallegos, Gabriele Morabito, Maria Fazio, Jesús Carretero

METRIC-AI: Entorno de monitorización semi-autónomo para entornos maleables 99
Alberto Cascajo García, Javier Fernández Muñoz, David E. Singh, Jesús Carretero

Simulación del manejo seguro y confiable de datos en entornos IoT-Edge-HPC 107
Diana Carrizales-Espinoza, Dante D. Sánchez-Gallegos, Jesús Carretero

AudioSAFFE: Fusión Multi-Modal Eficiente con Audio mediante Encoders Congelados 117
Maithri R. Kulasekara, Juan Francisco Inglés Romero, Baldomero Imbernón Tudela, José Luis Abellán Miguel

Arquitecturas del subsistema de memoria y almacenamiento secundario

Mecanismo avanzado de gestión de metadatos para sistemas de almacenamiento de gran escala 127
Genaro Sánchez Gallegos, Javier García-Blas, Jesús Carretero

Prolongando el Tiempo de Vida Útil de los Aceleradores ORB 137
Nicolás Landeros Muñoz, Alejandro Valero, Rubén Gran Tejero

Checkpointing transparente en aplicaciones que utilizan sistemas de ficheros ad-hoc 147
Dario Muñoz-Muñoz, Félix García-Carballeira, Alejandro Calderón-Mateos, Diego Camarmas-Alonso, Jesús Carretero

Simulación de procesamiento en memoria de GeMV y GeMM con LPDDR5 157
Jaime Martínez Fernández, Juan Gómez Luna, José Luis Abellán Miguel, Manuel Eugenio Acacio Sánchez

Redes de comunicaciones (2/2)

Análisis mediante TCPN de los efectos del error de tiempo en TSN 169
Alitzel Galilea Torres Macías, Antonio Ramírez Treviño, José Luis Briz Velasco, Juan Segarra Flor

Generación adaptativa de rutas de telemetría INT-MD en redes congestionadas 179
Pablo Gomariz Martínez, Francisco Manuel Delicado Martínez, Gabriel Cebrián-Márquez

Performance Evaluation of Dragonfly Connection Rules 189
Javier Navaridas, Jose A. Pascual

Evaluación de arquitecturas de comunicación distribuidas para la codificación de imágenes acelerada por GPU 197
Haojie Zheng, Carlos Reaño, Juan F. Ariño-Sales

Tecnologías clúster, plataformas distribuidas, BigData y Deep Learning (2/4)

Efficient evaluation of video encoding acceleration techniques: methodology and case study 547
Xiangyu Wan, Clara Baltasar, Guillermo Vigueras, Antonio Jesús Díaz-Honrubia

Modelo para la construcción de sistemas continuos de observación de la Tierra 211
Catherine A. Torres-Charles, Dante D. Sánchez-Gallegos, Diana Carrizales-Espinoza, Jesús Carretero, J. L. Gonzalez-Compean

Clasificación y segmentación de tumores de hueso largo en tomografía computarizada 219
Máximo Rodríguez Herrero, Dante D. Sánchez-Gallegos, Jesús Carretero

K8S-sims: Framework de Pruebas para Simuladores de Kubernetes 229
Matias Nicolas Medina Jara, Vitor Da Silva Verbel, Rosana Tomàs Cuñat, Fernando Guirado Fernández, Josep Lluís Lérida Monsó

HDCLite: A Compact Encoding Scheme for Memory-Efficient HyperDimensional Computing 237
Víctor Ortega, Soledad Escolar, Fernando Rincón, Jesús Barba, Julián Caba, Juan Carlos López

Docencia

NuCachis and NuTracer: Teaching Computer Organization through an Interactive RISC-V Memory Simulation Ecosystem 251
Aitor Echevarría Floranes, Borja Pérez Pavon, Esteban Stafford Fernandez, José Luis Bosque Orero

Un intercomunicador de audio de baja latencia con compresión perceptual basada en paquetes wavelet para la docencia de tecnologías multimedia 259
Savíns Puertas-Martín, Juan José Moreno, Vicente González Ruiz

Entorno para el modelado y simulación de sistemas electrónicos digitales 265
José Antonio Verde Jiménez, Álvaro Guerrero Espinosa, Diego Camarmas-Alonso, Félix García-Carballeira, Alejandro Calderón-Mateos

Implementación en FPGA del Procesador Elemental de WepSIM 275
Álvaro Guerrero Espinosa, José Antonio Verde Jiménez, Félix García-Carballeira, Alejandro Calderón-Mateos

Evaluación de prestaciones (1/3)

Evaluación comparativa de la eficiencia energética entre CPU y GPU en tareas de machine learning 285
Emil Hristov, Belen Bermejo, Carlos Juiz

Estimación en Tiempo Real de la Planitud en Bandas de Acero Laminadas mediante un Modelo Generativo Condicionado 291
Darío G. Lema, F. Javier De la Calle Herrero, Rubén Usamentiaga Fernández, Joaquín Entrialgo Castaño

Evaluando las limitaciones de la multiplicación-acumulación en formato posit y las herramientas generadoras de código 299
Mario Alonso, Miguel Angel Sacristán, Guillermo Botella, Alberto A. Del Barrio

Estudio comparativo de arquitecturas Intel y AMD con aplicaciones de grafo 305
Jaime De Lacy Orellana-Pizarro, Lucia Pons, Julio Sahuquillo

Computación cuántica (1/2)

Optimización de Circuitos para la Preparación de Estados Lógicos en Códigos Polares Cuánticos 313
Handy Kurniawan, Carmen G. Almudéver, Francisco García Herrero

Scaling Quantum Circuit Simulation: Optimizations for the FTDD Framework 321
Vicente López Oliva, José M. Badía, Maria Isabel Castillo

Comparación de métodos clásicos y cuánticos para la clasificación del peso en corderos 331
Elena Ondicol, Virginia Riego del Castillo, Lidia Sánchez González

Clasificador Cuántico Variacional para Datos Desbalanceados y No Lineales en Simulaciones Climáticas 337
Laura María Donaire, Elena R. Rull, Marouan Khollala, Gloria Ortega, Francisco Orts, Ester M. Garzón

Evaluación de prestaciones (2/3)

Scalable Spike Transmission in Brain Network Simulations 347
Mario Ibáñez, Marvin Kaster, Borja Pérez, Han Lu, Fabian Czappa, Sanda Díaz Pier

Interferencia en redes Dragonfly: análisis de patrones de tráfico y políticas de mapeo 359
Alejandro Baviera Guaita, María Engracia Gómez Requena, Julio Sahuquillo

Análisis preliminar del rendimiento computacional de PhysiCell en función de los parámetros biológicos y computacionales 367
Antonio de Jesús Menchaca-Martínez, Johann Bourcier, María Villarroya-Gaudó

Profiling de programas XDP en ARM mediante contadores hardware: diseño e implementación de ARMpect 377
Diego Valenzuela Saz, Alex Gracia Rodríguez, José Luis Briz Velasco, Juan Segarra Flor

Computación cuántica (2/2)

Simulación de computación cuántica en RISC-V: vectorización y paralelismo multihilo 385
Rebeca Rasco Flores, Sonia González, Eladio Gutiérrez, Francisco Javier Hormigo, Óscar Plata

Agregación de Quantum Walks de Szegedy con Quantum Markov Chain Monte Carlo 391
Aingeru Ramos, Jose A. Pascual, Javier Navaridas, Ivan Coluzza

Scheduling Hybrid Classical¿Quantum Workflows with Session Constraints 401
Jose A. Pascual, Alex Amenabar, Unai Garciarena, Javier Navaridas

Evaluación de prestaciones (3/3)

U-ATOMS: Estrategia Universal de Asignación de Hilos Basada en Transporte Óptimo para Procesadores Multinúcleo SMT 413
Marta Navarro, Vicent Pallardó-Julià, Lucia Pons, Salvador Petit, María E. Gómez, Julio Sahuquillo

Espera activa de bajo consumo en sistemas de tiempo real: caracterización de TPAUSE sobre hardware COTS 421
Alex Gracia Rodríguez, Hector Blanco Alcaine, Zoltan Fodor, David Zage, José Luis Briz Velasco, Juan Segarra Flor

Forge2: Plataforma de Experimentación para una Refrigeración Líquida Directa con Cambio de Fase 427
Pedro Chaves, Enrique Torres, Alejandro Valero, Víctor Viñals

Framework Paralelo para Carga y Aumento de Datos durante el Entrenamiento de Modelos de Deep Learning 437
Antonio De Toro Castro, Marcos Lupión Lorente, Vicente González Ruiz, Juan Francisco Sanjuan Estrada, Pilar Martínez Ortigosa

Series temporales de temperaturas de GPU durante operaciones GEMM 447
Francisco Yáñez Rodríguez, Carlos Fernández Sánchez

Aumento adaptativo paralelo de imágenes sintéticas para la estimación de pose humana en el espectro térmico. 453
Antonio De Toro Castro, Marcos Lupión Lorente, Juan Francisco Sanjuan Estrada, Vicente González Ruiz, Pilar Martínez Ortigosa

Planificación de tareas moldeables en GPUs dinámicamente reconfigurables mediante Aprendizaje por Refuerzo Profundo 463
Jorge Villarrubia, Luis Costero, Francisco D. Igual, Katzalin Olcoz

Generación de Código RISC-V Optimizado con MLIR y xDSL 473
Jie Lei, Adrián Castelló, Héctor Martínez

Tecnologías clúster, plataformas distribuidas, BigData y Deep Learning (3/4)

Enabling intra-node multi-GPU malleability for general-purpose applications 485
Alex Amenabar, Jose A. Pascual, Javier Navaridas, Unai Garciarena

Poda estructurada en BERT: Un estudio de rendimiento y consumo en el Borde 493
Nicolás Hernández González, Pedro Toledo Delgado, Vicente Blanco Pérez, Francisco Almeida Rodríguez

Generación sintética multilingue para HTR: diseño de plataforma, escalabilidad y evaluación de prestaciones 501
Joaquim Oliva Costas, Daniel Grao González, Francesc Xavier Carrera Farran, Francesc Giné de Sola, Vitor Da Silva Verbel

Securizando el sistema de ficheros Expand utilizando blockchain 509
Diego Camarmas-Alonso, Félix García-Carballeira, Alejandro Calderón-Mateos, Jesús Carretero

Aplicaciones de la computación de altas prestaciones (1/3)

Combinando Escalado en Frecuencia, Procesamiento Multihilo y Vectorización para una Filogenética más Sostenible 521
Sergio Santander-Jiménez, Miguel A. Vega-Rodríguez

Hacia la identificación textil robusta mediante detección de inconsistencias del diccionario en separación espectral 529
Adrián Sarrías, Miguel Martínez Rach, Otoniel López Granado, Héctor Migallón

Arquitectura software para el análisis hiperespectral en tiempo real en sistemas industriales de clasificación de materiales 537
Adrián Sarrías, Miguel Martínez Rach, Otoniel López Granado, Héctor Migallón

Efficient evaluation of video encoding acceleration techniques: methodology and case study 547
Xiangyu Wan, Clara Baltasar, Guillermo Vigueras, Antonio Jesús Díaz-Honrubia

Arquitecturas heterogéneas y modelos de programación (2/4)

Evaluación del compromiso área-memoria en FPGAs con HBM para aplicaciones de stencils utilizando OpenCL 555
Daniel López-Martínez, Manuel De Castro, Diego R. Llanos

Planificador de Tareas Basado en Green Contexts 565
Juan José Martín Osuna, José Flich Cardo, Carles Hernández Luz

Toolchain Automatizado para la Integración de Kernels CUTLASS en Motores TensorRT 575
Alejandro Ripoll Acosta, José Flich Cardo, Carles Hernández Luz

Optimización de SpMM iterativo en RISC-V mediante ELLPack y reordenamiento de datos 583
Andrea Herrerías León, Luis F. Romero, Gerardo Bandera, Óscar Plata

Tecnologías clúster, plataformas distribuidas, BigData y Deep Learning (4/4)

Diseño, despliegue y gestión de un sistema de alto rendimiento para computación científica 591
Cosmin Octavian Petre Carastoian, Javier García-Blas, David E. Singh, Jesús Carretero

Evaluación de Mecanismos Semiasíncronos Deterministas y Adaptativos en Aprendizaje Federado 601
Víctor Hidalgo-Izquierdo, Carmen Carrión, María Blanca Caminero Herráez

Framework de Edge AI para la Monitorización de Crisis Epilépticas Mediante Dispositivos Wearables en Microcontroladores 611
Javier Navarro Lázaro, Marcos Lupión Lorente, Vicente González Ruiz, Juan Francisco Sanjuan Estrada, Pilar Martínez Ortigosa

Asignación de carga orientada a la eficiencia energética en entrenamiento distribuido heterogéneo 619
Daniel Suárez Labena, Vicente Blanco Pérez, Pedro Toledo Delgado, Francisco Almeida Rodríguez

Lenguajes, compiladores y herramientas de programación y ejecución paralela (1/2)

Análisis Comparativo del Rendimiento de Algoritmos Paralelos de Ordenamiento usando Intel TBB 631
Mario Rossainz López, Bárbara Sánchez Rinza, Gabriela Martínez Rosas, Manuel I. Capel Tuñón

¿Highway para álgebra lineal dispersa? Estudio de Rendimiento y Portabilidad 639
Patricia Siwinska, Adrián Castelló, Héctor Martínez, Andrés E. Tomás, Enrique S. Quintana-Ortí

Evaluación de la Vectorización Automática del Compilador frente a SIMD Manual para el Cálculo de Slope Entropy 649
Juan Seguí Moreno, Antonio Molina Picó

Breaking Barriers in Neural Networks and Transformers Compilation 655
Shreya Alladi, Alberto Ros, Alexandra Jimborean

Arquitecturas heterogéneas y modelos de programación (3/4)

ChipletGPU-Sim: Desarrollando un Simulador para Arquitecturas GPU Multi-Chiplet 665
Germán Vicente Tomás, Antonio García Guirado, Alberto Ros, José Luis Abellán Miguel, Manuel Eugenio Acacio Sánchez

Estrategias para implementar Propagación de Momentos en Redes Neuronales Bayesianas 675
Samuel Pérez Pedrajas, Javier Resano, Darío Suárez Gracia

Análisis de Paralelismo y Robustez de Arquitecturas de Deep Learning para HSI en Hardware Analógico 683
María B. García Flores, Carlos Cañada Rostro, Mercedes E. Paoletti Ávila, Juan M. Haut Hurtado

Memoria Reconfigurable para Resultados Parciales en Aceleradores de DNN Eficientes 689
Adrián Navarro, José Cano, José Luis Abellán Miguel, Manuel Eugenio Acacio Sánchez

Aplicaciones de la computación de altas prestaciones (2/3)

Paralelizando el método TSLLS para la modelización de paneles fotovoltaicos 701
X. Moreno-Vassart, Vicente Galiano, F. Javier Toledo, Victoria Herranz

Extensión del simulador WorkflowSim para el procesamiento de flujos de trabajo con datos continuos 709
Anna Marín Nuño, Rosana Tomàs Cuñat, Concepció Roig Mateu, Vitor Da Silva Verbel, Fernando Guirado Fernández

Plataforma de Validación de Precisión Adaptativa para Simulaciones Lattice Boltzmann con Política Automática de Asignación de Precisión 717
Marcos Ramos Montes, José Flich Cardo

Aceleración de la Transcodificación Espacial en VVC para Aplicaciones LCEVC en Ultra HD 723
Pedro Jesús Martínez-Herrero, Gabriel Cebrián-Márquez, Antonio Jesús Díaz-Honrubia, Pedro Cuenca

Lenguajes, compiladores y herramientas de programación y ejecución paralela (2/2)

Detección de crisis epilépticas en un SoC RISC-V de ultra bajo consumo 733
David Díaz Reyes, Francisco Javier Hormigo, María Ángeles González Navarro

Compile-Time Reordering for Instruction Fusion 743
Ravikiran Ravindranath Reddy, Sawan Singh, Arthur Perias, Alberto Ros, Alexandra Jimborean

A comprehensive characterization of performance and energy consumption of ORB-SLAM in multiple hardware platforms 753
Marcos Ilarraza Sarto, Jorge Ferrer Beired, Darío Suárez Gracia, Alejandro Valero, Rubén Gran Tejero

Porting the Striped Smith-Waterman Library to RISC-V via LLM-Driven Translation 761
Alejandro Fernández Camello, Manuel Prieto Matías, Carlos García Sánchez

Arquitecturas heterogéneas y modelos de programación (4/4)

Transformers Eficientes para Imágenes Hiperespectrales: Optimización de Memoria mediante SpectralStem y Kernels Triton 773
Benjamín Sánchez Calza, Mercedes E. Paoletti Ávila, Juan M. Haut Hurtado

Optimizing AI Inference on Android Devices: A Comparative Study of GPU and NPU Acceleration 779
Iulius Gherasim, Carlos García Sánchez

Creando contigüidad para la escalabilidad extrema en aplicaciones de tipo stencil 789
David Díez-Poza, Sergio Alonso Pascual, Yuri Torres, Arturo Gonzalez-Escribano

Atención de Alto Rendimiento en Arquitectura Hopper para Clasificación Hiperespectral 799
Carlos Cañada Rostro, María B. García Flores, Mercedes E. Paoletti Ávila, Juan M. Haut Hurtado

Aplicaciones de la computación de altas prestaciones (3/3)

Análisis y optimización microarquitectónica del algoritmo de Ruido Perlin 2D en AMD Zen 3 807
Luis Miguel Trinidad Salvador, Maria José Morón Fernández

Energy Early Stopping: Un mecanismo para la optimización del consumo energético en el entrenamiento de redes neuronales profundas 815
Benjamín Sánchez Calza, Mercedes E. Paoletti Ávila, Juan M. Haut Hurtado

Aceleración multi-GPU del cifrado homomórfico para aprendizaje automático 821
Carlos Agulló-Domingo, Óscar Vera-López, Seyda Guzelhan, Lohit Daksha, Aymane El Jerari, Kaustubh Shivdikar

Aprovechamiento de la gestión dinámica de recursos para la eficiencia energética: un caso de estudio 831
Paula Sánchez-Checa, Jesús Carretero, David E. Singh

Parte 2.- X Jornadas de Computación Empotrada y Reconfigurable

Aplicaciones de los sistemas empotrados y reconfigurables

Comparativa entre transformada de Fourier de tiempo corto y sonogramas neuromórficos en audio para mantenimiento predictivo con redes de convolución 845
Juan M. Montes-Sánchez, Plácido Fernández Cuevas, Carlos Rangel-Cascajosa, Juan P. Domínguez Morales, Saturnino Vicente-Díaz, Ángel Jiménez-Fernández

Predicción solar de bajo coste computacional para plataformas IoT empotradas 851
Melisa Kuzman, Soledad Escolar, Paula Cervellini, Carlos Hernández, José Luis Mira, Javier Romero

Controlador neuromórfico de robot articulado en FPGA con interfaz para aceleradores de redes neuronales pulsantes 859
Alejandro Linares Barranco, Daniel Casanueva Morato, Diógenes Enrique Trejos Hernández, Daniel Cascado Caballero, Juan P. Domínguez Morales, Ángel Jiménez-Fernández

Pruebas de Exactitud y Precisión del brazo robótico neuromórfico ED-Scorbot 865
Diógenes Enrique Trejos Hernández, Elena Cerezuela Escudero, Alejandro Linares Barranco

Arquitecturas heterogéneas empotradas

FPGA-based Energy-efficient Decision Tree Ensemble Accelerator 875
Unai Fernandez-Urroz, Libe Mori, Unai Garciarena, Javier Navaridas, Jose A. Pascual

SNNorlax: Accelerating the training and inference of Spiking Neural Networks on FPGA 883
Asier Esteban Sauce, Libe Mori, Jose A. Pascual, Javier Navaridas

Deep Learning 3D Object Detection and Classification at the Edge of IoT 893
Inés Pérez Delgado, Rogelio Hernández Lorite, Jorge Portilla Berrueco

Estudio del impacto de políticas de planificación en un robot de análisis clínico basado en un planificador dinámico 901
Plácido Fernández-Cuevas, Juan M. Montes-Sánchez, Carlos Rangel-Cascajosa, Ángel Jiménez-Fernández, Saturnino Vicente-Díaz, Antón Civit-Balcels

Inteligencia artificial en sistemas empotrados y reconfigurables

Semantic RAID: Lightweight Fault Tolerance via Semantic Redundancy for Distributed Edge Inference 911
Radu Spaimoc, Lluis Mas, Jordi Vilaplana, Josep Rius, Didac Florensa, Jordi Mateo

Generación eficiente y confiable de SVA basada en LLM locales mediante el uso de TGTS 921
Javier Romero, Soledad Escolar, Jesús Barba, Fernando Rincón, Julián Caba, Melisa Kuzman

FRAGATA: Recuperación semántica de tickets de soporte HPC mediante RAG híbrido sobre 20 años de histórico de Request Tracker 931
Santiago Paramés Estévez, Nicolás Filloy Montesino, Jorge Fernández Fabeiro, José Carlos Mouriño Gallego

Multimodal LiDAR-RGB Fusion for Object Classification on IoT Edge Devices 937
Blanca Poves Arjona, Rogelio Hernandez, Gabriel Mujica

Sistemas ciberfísicos e IoT

Triple Redundancia Modular en Rust para Sistemas Multicore bajo Radiación 949
Darío González-Montesoro, Alejandro Serrano-Cases, Sergio Cuenca-Asensi, Rodrigo Possamai-Bastos, Emanuel Atukpor, Antonio Martínez Álvarez

Análisis Experimental de la Sincronización de LoRaWAN Clase B: Hacia Redes IoT Integradas con NTN 957
María Ángeles Amador, Celia Garrido-Hidalgo, Luis Roda-Sanchez, Teresa Olivares

Calibración Resiliente de Sensores Ambientales de Bajo Coste 963
Abdellatif Bendjeddou, Ahmed Boubrima

Arquitectura basada en aprendizaje federado y comunicación LoRaWAN con componentes DtS-IoT para optimización del tráfico 971
Víctor Hidalgo-Izquierdo, Alejandro Romero Lorenzo, Teresa Olivares, Carmen Carrión, María Blanca Caminero Herráez

Arquitecturas, lenguajes y sistemas tolerantes a fallos (1/2)

Implementación en FPGA de una Neurona de Impulsos en Tiempo Real con Salida de Pulsos Pseudoaleatorios 983
Dionicio Yukitaro Kono García, María de Lourdes Rivas Becerra, María Regina Rivas Becerra, Juan José Raygoza Panduro, Edwin Christian Becerra Alvarez, Susana Ortega Cisneros

Medición de rendimiento energético en entornos HPC mediante el uso de xCAT 991
Miguel Baños González, Jorge Valle Neila, Javier Corral García, Juan Antonio Rico Gallego, Felix O. García

Diseño de un Emulador para RISC-V Implementado en Metalenguaje 1001
Elías Coronado Salazar, Juan Pablo De la Cerda González, Edwin Christian Becerra Alvarez, Juan José Raygoza Panduro, Dionicio Yukitaro Kono García, Jorge Rivera Dominguez

Hacia la eficiencia en la agricultura de precisión: diseñoo de una plataforma de sensores IoT y computación en el borde 1009
Santiago Díaz Romero, Simón Pérez Martín, Antonio Ríos Navarro, Ángel Jiménez-Fernández, Alejandro Linares Barranco

Arquitecturas, lenguajes y sistemas tolerantes a fallos (2/2)

Accelerating Zero-Knowledge Proof Kernels on CGRA Architectures 1017
Cristian Campos Ferrer, Ángeles G. Navarro, Sonia González-Navarro

Evaluación de la fiabilidad y protección frente a errores en Vision Transformers 1025
Lester Frias, José M. Badía, Germán León, Adrián Amor-Martín, José A. Belloch

Agentes LLM periféricos para la gestión autónoma del mantenimiento en infraestructura IoT crítica 1035
Ángela Gijón Flores, Félix Jesús Villanueva Molina, Fernando Rincón, Cristina Bolaños Peño, Xavier Del Toro García, Juan Carlos López

Indice de autores 1045

Ponencias invitadas

A Journey from Cache Coherence to Microarchitecture

Alberto Ros

Resumen

La arquitectura de computadores ha evolucionado a lo largo de décadas de ingenio, equilibrando rendimiento, eficiencia y programabilidad. Aunque la inteligencia artificial acapara cada vez más la atención como posible solución universal, el diseño hardware y la investigación en microarquitectura siguen siendo la base sobre la que se construye el progreso de los procesadores actuales.

En esta charla reviso mi viaje en investigación explorando la interacción entre los sistemas de memoria y la microarquitectura. El recorrido comienza con cuestiones fundamentales sobre los protocolos de coherencia de caché para seguir con su estrecha relación con los modelos de consistencia de memoria. A partir de ahí, examino cómo la implementación hardware de estos modelos de consistencia influye en el diseño de las colas de carga y almacenamiento, para enlazar con mejoras en la microarquitectura de los procesadores y técnicas de ejecución especulativas. A través de esta retrospectiva, destaco cómo la investigación en arquitectura continúa revelando los principios que guían las siguientes generaciones de procesadores.

Biografía

Alberto Ros es catedrático en el Departamento de Ingeniería y Tecnología de Computadores de la Universidad de Murcia, España. Tras recibir financiación del gobierno español para realizar sus estudios de doctorado, obtuvo el título de doctor en Informática por la Universidad de Murcia en 2009. Posteriormente, ocupó puestos posdoctorales en la Universitat Politècnica de València y en la Universidad de Uppsala. En 2018, recibió una subvención Consolidator Grant del Consejo Europeo de Investigación (ERC) para mejorar el rendimiento de las arquitecturas multinúcleo, y en 2024 obtuvo una subvención Proof of Concept del mismo organismo. Su trabajo se centra en la coherencia de caché, el diseño de jerarquías de memoria, la consistencia de memoria y la microarquitectura de procesadores, áreas en las que ha coautorizado más de 100 artículos revisados por pares. Ha sido incluido en el ISCA Hall of Fame y en el MICRO Hall of Fame, y es miembro sénior del IEEE.

The Price Performance of Performance Models

Felix Wolf

Resumen

Para comprender el comportamiento de escalado de las aplicaciones de HPC (Computación de Alto Rendimiento), los desarrolladores suelen utilizar modelos de rendimiento. Un modelo de rendimiento es una fórmula que expresa una métrica de rendimiento crítica, como el tiempo de ejecución, en función de uno o más parámetros de ejecución, como el número de núcleos y el tamaño de la entrada.

Los modelos de rendimiento ofrecen información rápida con un nivel muy alto de abstracción, incluyendo predicciones sobre el comportamiento futuro. Dada la complejidad de las aplicaciones actuales, que a menudo combinan varios algoritmos sofisticados, crear modelos de rendimiento manualmente resulta extremadamente laborioso.

El modelado de rendimiento empírico, que es el proceso de aprender tales modelos a partir de datos de rendimiento, ofrece una alternativa conveniente, pero conlleva su propio conjunto de desafíos. Los dos más destacados son el ruido y el costo de los experimentos necesarios para generar los datos subyacentes. En esta charla, revisaremos el estado del arte en el modelado de rendimiento empírico e investigaremos cómo podemos emplear el aprendizaje automático (machine learning) y otras estrategias para mejorar la calidad y reducir el costo de los modelos resultantes.

Biografía

Felix Wolf es catedrático en el Departamento de Informática de la Universidad Técnica de Darmstadt en Alemania, donde dirige el Laboratorio de Programación Paralela. Trabaja en métodos, herramientas y algoritmos que dan soporte al desarrollo y despliegue de sistemas de software paralelo en diversas etapas de su ciclo de vida. Felix recibió su doctorado por la Universidad RWTH Aachen en 2003. Tras trabajar más de dos años como posdoctorando en el Laboratorio de Computación Innovadora de la Universidad de Tennessee, fue nombrado líder de grupo de investigación en el Centro de Supercomputación de Jülich. Entre 2009 y 2015, fue director del Laboratorio de Programación Paralela en la Escuela Alemana de Investigación para Ciencias de la Simulación en Aquisgrán (Aachen) y catedrático en la Universidad RWTH Aachen. Felix ha realizado importantes contribuciones a varias herramientas de rendimiento de código abierto para programas paralelos, incluyendo Scalasca, Score-P y Extra-P. Además, ha impulsado el Instituto Virtual de Supercomputación de Alta Productividad, una iniciativa internacional de desarrolladores de herramientas de programación HPC orientada a mejorar, integrar y desplegar sus productos. Ha publicado más de 150 artículos arbitrados sobre computación paralela, varios de los cuales han recibido premios.

La arquitectura de Arm: como trasladar ideas y conceptos a millones de dispositivos

Daniel Piso Fernández

Resumen

La arquitectura de Arm es la piedra angular del mayor ecosistema de computación del planeta, compuesto por 325 mil millones de dispositivos. Abarca tres areas diferentes para el diseño de SoCs (System-on-Chip): 'Arm CPU Architecture', 'Architecture Security Features' y 'Arm System Architectures'. La arquitectura CPU es una familia de ISAs (Instruction Set Architecture) definida como un estándar para la interacción hardware-software organizada en tres perfiles diferenciados A-profile (altas prestaciones), R-profile (tiempo real), M-profile (microcontroladores). El principio de interacción hardware-software garantiza la portabilidad de todo software que obedezca a los comportamientos abstractos definidos en la arquitectura.

En esta charla describiremos como la incorporación de nuevas características a la arquitectura posibilita la innovación tecnológica y la mejora continua de las prestaciones de los diferentes diseños. Este proceso requiere de un delicado balance entre distintos requerimientos, costes de implementación y desarrollo software, así como el alineamiento con las necesidades del mercado consiguiendo que una determinada solución sea factible en un rango amplio de microarquitecturas.

A medida que una arquitectura se desarrolla a lo largo de décadas, la especificación se convierte en un gran corpus de datos. El simple hecho de tener que proporcionar una definición consistente y correcta viene acompañada de una serie de retos tecnológicos que requiere del desarrollo de estándares y tecnologías que permiten el prototipado de ideas y la verificación del contenido. La calidad de dicho contenido es crucial para facilitar el consumo de la especificación (en muchos caos de manera automatizada) para minimizar el traslado de defectos a los diseños y minimizar los tiempos de implementación.

El desarrollo del Architecture Specification Language - ASL es un ejemplo exitoso para resolver a escala el problema de la especificación de una arquitectura. Estos son algunos de los ingredientes que en Arm creemos necesarios para que este despliegue de tecnología a gran escala haya sido posible.

Biografía

Daniel Piso Fernández es Director de Ingeniería en el Architecture Technology Group (ATG) en Arm, donde lidera un equipo internacional de más de setenta profesionales distribuidos entre Cambridge, Austin, Bangalore. Su labor se centra en las especificaciones de arquitectura de Arm, el diseño de la estrategia tecnológica y la evolución del modelo operativo asociado a su publicación y consumo. Uno de los mayores exponentes de este trabajo es el Arm Architecture Reference Manual (Arm ARM), que contiene la especificación completa de la arquitectura de Arm para procesadores de alto rendimiento, y que en su última versión PDF roza las diecisiete mil páginas.

Con una trayectoria previa en el proyecto European Spallation Source, en el mundo de los aceleradores de partículas, donde creó y dirigió equipos multidisciplinares y gestionó proyectos tecnológicos a gran escala, Daniel acumula amplia experiencia en gestión organizativa, ingeniería de sistemas y desarrollo de hardware y software. Su formación científica, Doctor en Física por la Universidad de Santiago de Compostela y PostDoc en la George Mason University en Virginia, USA, complementa más de dos décadas de trabajo en arquitectura de computadores, diseño electrónico, control de sistemas y programación. Daniel se ha especializado en combinar liderazgo, innovación y una sólida base técnica para impulsar tecnologías complejas y transformar organizaciones.

Parte 1.- XXXVI Jornadas de Paralelismo

Arquitecturas del procesador, multiprocesadores y chips multinúcleo

Memoria transaccional *hardware in-core* de bajo coste

Álvaro Rubira-García[1], Eduardo José Gómez-Hernández[1], Rubén Titos-Gil[1] y Alberto Ros[1]

Resumen— **La memoria transaccional ofrece una alternativa sencilla para la sincronización en programación paralela: permite definir transacciones (grupos de instrucciones) que se ejecutan de manera atómica. Con la memoria transaccional *hardware* (HTM), los procesadores incorporan *hardware* especializado para ejecutar transacciones eficientemente, principalmente a través de la ejecución especulativa. Estas modificaciones suelen requerir cambios importantes en las cachés de primer nivel o *hardware* adicional para realizar *checkpoints* de los registros.**

Motivados por el creciente número de instrucciones en vuelo en los procesadores actuales, revisitamos y extendemos un enfoque alternativo que reutiliza el soporte para la ejecución fuera de orden para implementar HTM con cambios mínimos en la microarquitectura.

Posteriormente, evaluamos el rendimiento de esta HTM *in-core* con el simulador gem5, comparando nuestra implementación con un sistema de referencia basado en cerrojos y un modelo de Intel TSX con *checkpointing* de registros y control de versiones en caché. A pesar de su simplicidad, para 32 núcleos, nuestra propuesta alcanza un rendimiento similar al de TSX en *benchmarks* de estructuras de datos concurrentes.

Palabras clave— **Memoria transaccional hardware, paralelismo, ejecución especulativa**

I. Introducción

La memoria transaccional *hardware* (HTM) intenta facilitar la programación de algoritmos paralelos permitiendo delimitar grupos de instrucciones (transacciones) que serán ejecutados atómicamente, siendo el sistema de HTM el encargado de garantizar la atomicidad de manera eficiente. En los sistemas de HTM comerciales, esto se suele conseguir ejecutando transacciones de manera especulativa, activando un sistema de detección de conflictos que generará un *abort* si otros hilos realizan accesos que violen la atomicidad. El *abort* descarta todas las modificaciones realizadas por la transacción, permitiendo posteriormente reintentar la transacción de manera especulativa u optar por un camino alternativo con sincronización tradicional (con cerrojos, por ejemplo) que asegure el progreso de la transacción.

Una desventaja de las implementaciones convencionales de HTM es que requieren modificaciones en componentes críticos del procesador, incluyendo *hardware* específico para realizar un *checkpoint* del estado de los registros [1], [2], [3] o modificaciones en las cachés más cercanas al núcleo [1], [2]. Las modificaciones en estos componentes críticos deberían ser evitadas para reducir el esfuerzo de verificación al diseñar nuevas arquitecturas [4], [5].

Buscando reducir esta complejidad, nos fijamos en la propuesta de *speculative lock elision* (SLE) [6]. SLE presenta dos implementaciones alternativas para HTM: una más similar a la HTM comercial de la actualidad, que requiere un *checkpoint* de registros y modificaciones en las líneas de caché; y otra *in-core*, que reutiliza mecanismos de los procesadores fuera de orden modernos. La alternativa *in-core* no fue investigada en profundidad (no encontramos ninguna evaluación experimental), lo cual puede deberse a que el tamaño de las transacciones (en cuanto a número de instrucciones y accesos a memoria) de esta alternativa está limitado por la capacidad de la ventana de instrucciones del procesador. Sin embargo, con el paso de los años, el tamaño de las estructuras que permiten la ejecución fuera de orden ha aumentado considerablemente (basta con comparar el *reorder buffer* (ROB) de 128 entradas de un Pentium 4 [7] con las 576 entradas de Lion Cove [8]). Se siguen aplicando límites al tamaño de las transacciones, pero ya permiten realizar operaciones atómicas de gran utilidad en la programación de estructuras de datos concurrentes y aplicaciones científicas.

En este artículo, proponemos la memoria transaccional *hardware in-core* de bajo coste (LHTM). Extendemos el mecanismo *in-core* de SLE para evaluar una implementación completa y centrada en la simplicidad. Simulando *benchmarks* con estructuras de datos concurrentes en gem5 [9], demostramos que esta alternativa *in-core* permite simplificar la programación a la vez que mantiene un rendimiento similar al de Intel TSX sin requerir modificaciones tan significativas en el núcleo.

II. Estado del arte

Nuestra propuesta reutiliza mecanismos originalmente creados para optimizar el rendimiento de los procesadores fuera de orden modernos. En esta sección resumimos brevemente cada uno de estos mecanismos y presentamos SLE [6] con más detalle.

A. Procesadores fuera de orden

A.1 *Reorder buffer*

Los procesadores superescalares dinámicos aprovechan el paralelismo a nivel de instrucción ejecutando instrucciones de forma especulativa y fuera de orden, tan pronto como las unidades y los operandos necesarios estén disponibles. Solo el *front-end* y *commit* operan en orden, siendo esto suficiente para garantizar un resultado coherente con la ejecución en orden. Además, estos procesadores pueden ejecutar instruc-

[1]Dpto. de Ingeniería y Tecnología de Computadores, Universidad de Murcia, e-mail: {alvaro.r.g,eduardojose.gomez,rtitos}@um.es, aros@ditec.um.es.

ciones basándose en la especulación sobre información no disponible, almacenando los resultados en el *reorder buffer* (ROB) [10]; si una predicción acaba siendo errónea, las entradas afectadas se eliminan para descartar el estado incorrecto.

A.2 *Load queue* y *store queue*

Las instrucciones de lectura y escritura mantienen una entrada en la *load queue* (LQ) y *store queue* (SQ), respectivamente, desde *issue* hasta *commit*. Las lecturas acceden a memoria cuando su instrucción es ejecutada, mientras que las escrituras aplazan sus modificaciones hasta hacer *commit*. Tras salir del ROB y la SQ en *commit*, las escrituras se guardan en el *store buffer* (SB) hasta completar el acceso a caché.

A.3 Ejecución especulativa de lecturas

Los modelos de consistencia de memoria estrictos (p. ej., x86-TSO [11]) no permiten reordenar las lecturas entre sí. Sin especulación, esto supondría retrasar todas las lecturas posteriores (en el orden establecido por el programa) a lecturas bloqueadas por fallos de caché de alta latencia. En la práctica, las lecturas se ejecutan de forma especulativa, sin esperar a que se completen lecturas anteriores.

Una lectura especulativa es correcta siempre que, cuando se complete la primera lectura (la más antigua, pero que en este caso se ha completado después), el valor del bloque accedido por la segunda lectura no haya cambiado [12]. La mayoría de procesadores optan por garantizar esta condición descartando y reejecutando todas las lecturas en vuelo a un bloque cuando éste recibe una invalidación en caché de primer nivel (L1). Como consecuencia, L1 debe notificar al núcleo cada vez que se invalide una línea de caché.

B. Cerrojos en líneas de caché

Los procesadores que incorporan operaciones atómicas suelen incluir una manera de activar cerrojos en las líneas de L1. Estos cerrojos permiten retrasar todas las peticiones externas a la línea de caché mientras se llevan a cabo operaciones atómicas [13] (el cerrojo no retrasa el procesamiento de las peticiones locales).

C. Speculative lock elision

Propuesta por Rajwar y Goodman [6], *speculative lock elision* (SLE) detecta el uso de cerrojos e intenta sustituirlo dinámicamente por secciones atómicas de HTM. La ejecución especulativa a través de HTM se intenta un número limitado de veces hasta llegar a un máximo de reintentos, tras lo cual se puede adquirir el cerrojo original para asegurar el progreso del programa.

Si bien en este artículo no tratamos la idea de introducir automáticamente HTM en programas originalmente escritos con cerrojos, nos centramos en los mecanismos que SLE propone para implementar la HTM. SLE propone almacenar las modificaciones a memoria de la transacción evitando que las escrituras salgan del SB hasta que se haya validado la sección atómica, y presentan dos alternativas para poder recuperar el estado de los registros en caso de *abort*: 1) con un *checkpoint* en *hardware* o 2) evitando que los cambios especulativos sean retirados del ROB. Nos centramos en la segunda alternativa, que no requiere *hardware* adicional.

III. LHTM

En este artículo buscamos conseguir una HTM eficiente mientras minimizamos el esfuerzo de verificación introducido. Mantenemos los mecanismos para almacenar el estado especulativo, preparar las líneas de caché para *commit* y detectar conflictos de Rajwar [14]; pero completamos un desarrollo más detallado de la alternativa *in-core* e incluimos un método sencillo para aplicar atómicamente las modificaciones especulativas, con el objetivo de poder evaluar una HTM *in-core*. Además, a diferencia de SLE, no pretendemos detectar el uso de cerrojos. En su lugar, asumimos que el ISA dispone de soporte para delimitar las transacciones.

En LHTM, el ROB y la SQ tienen que almacenar todas las modificaciones transaccionales al estado del procesador. Antes de la fase de *commit* de la transacción, las instrucciones se ejecutan y cargan las líneas de caché accedidas en la transacción. Mientras tanto, realizamos búsquedas en la LQ y la SQ para detectar conflictos cada vez que se pierden permisos de acceso de líneas válidas en L1. Una vez la transacción está preparada para aplicar todos sus cambios de manera atómica, activamos un cerrojo para todas las líneas de L1 y permitimos que los cambios especulativos modifiquen en estado arquitectural del procesador.

A. Interfaz del ISA

LHTM aprovecha el soporte para definir transacciones del ISA x86-64. Específicamente, utilizamos las instrucciones `XBEGIN`, `XEND` y `XABORT` de las *Intel Transactional Synchronization Extensions* (TSX).

`XBEGIN` marca el inicio de la transacción, y devuelve en el registro `rax` un código de estado indicando si la transacción se ejecutó de manera correcta, o un código con la causa del *abort* para reintentar la transacción. `XEND` marca el final de la transacción y `XABORT` genera un *abort* explícito. Para asegurar el progreso aunque haya *aborts* recurrentes (por excepciones o por conflictos con otros hilos), el manejador de *aborts* puede optar por adquirir un cerrojo convencional, denominado *fallback*, tras realizar un número específico de reintentos. Los *aborts* explícitos se suelen utilizar para reiniciar la transacción en el manejador de *aborts* cuando se detecta que otro hilo ha adquirido el cerrojo de *fallback*.

Finalmente, nuestra implementación requiere que las instrucciones `XBEGIN` y `XEND` sean tratadas como barreras de memoria. Esto permite aislar de manera sencilla los accesos a memoria de dentro de la transacción de los accesos externos.

B. Propuesta de diseño

B.1 Almacenamiento del estado especulativo

Dado que las instrucciones solamente pueden hacer *commit* a través de la cabeza del ROB y que lo deben hacer en orden, LHTM simplemente bloquea el *commit* de `XBEGIN`. De esta manera, retrasamos el *commit* de todas las instrucciones de la transacción, evitando modificaciones en el estado de la arquitectura. Como resultado, también retrasamos las modificaciones a caché, porque las escrituras solamente acceden a memoria después de hacer *commit*. Las lecturas, en cambio, pueden acceder a memoria al ser ejecutadas, permitiendo que instrucciones que dependan de sus resultados también puedan ser ejecutadas con los datos leídos.

Si bien el uso del ROB para almacenar las modificaciones de la transacción evita la necesidad de *hardware* adicional para crear *checkpoints* o marcar líneas de caché, supone la desventaja de no permitir que las instrucciones individuales empiecen a hacer *commit* hasta que se inicie el proceso de *commit* de la transacción en sí.

B.2 Acceso a las líneas de caché de la transacción

Nuestro protocolo de *commit* de la transacción requiere que todas las líneas accedidas estén presentes en L1 antes de activar el cerrojo de caché. Estas líneas de caché deben estar presentes con los permisos necesarios para poder completar la transacción sin necesidad de generar o esperar la respuesta de mensajes de coherencia (p.ej., para un protocolo MESI, las líneas sólo accedidas por lecturas tienen que estar en estado M, E o S; las líneas accedidas por escrituras deben estar en estado M o E).

Las instrucciones de lectura acceden a caché al ser ejecutadas. Las escrituras, en cambio, solamente acceden a memoria tras *commit*. Sin embargo, existen estrategias de prebúsqueda que generan peticiones para acceso exclusivo por adelantado [12]. Muchos procesadores modernos generan estas peticiones en *commit* [15]. En nuestra propuesta, generamos peticiones de acceso exclusivo cuando las instrucciones de escritura son ejecutadas, con el objetivo de preparar las líneas de caché necesarias para el *commit* de la transacción.

B.3 Detección y resolución de conflictos

Los conflictos pueden aparecer cuando la transacción está siendo ejecutada, mientras se cargan las líneas necesarias en L1. El núcleo puede detectar conflictos al recibir notificaciones de invalidaciones (las mencionadas en la Subsubsección II-A.3) a direcciones accedidas en la transacción. En *hardware*, esto se puede conseguir comparando todas las notificaciones de invalidación con las lecturas y escrituras de la LQ y SQ, respectivamente. Como `XBEGIN` y `XEND` son barreras de memoria, todas las lecturas y escrituras de la LQ y SQ pertenecen a la transacción.

Las lecturas de la transacción sólo pueden generar un conflicto al recibir notificaciones de invalidaciones. Para resolver el conflicto, seguimos el camino más simple, que imita el comportamiento de muchos procesadores modernos [16], y descartamos todas las instrucciones en el cauce del procesador (incluyendo todas las instrucciones de la transacción).

Las escrituras, por otra parte, se ven afectadas tanto por invalidaciones como *downgrades*. Los *downgrades* eliminan el permiso de acceso exclusivo pero no invalidan la línea, de manera que provocan la transición de una línea en estado M o E al estado S. En cualquier caso, una petición externa que entre en conflicto con lecturas o escrituras de la transacción provocará un *abort* que descarta todas las instrucciones del cauce.

B.4 *Commit* de la transacción

Las siguientes condiciones deben cumplirse antes de poder iniciar el *commit* de la transacción:

1. Debe existir un `XEND` *no especulativo* [17] en la ventana de instrucciones. Es decir, debe existir un `XEND` para el que ninguna de las instrucciones anteriores (en orden de programa) pueda causar una excepción o fallo de especulación [17]. Esto garantiza que no se tendrá que manejar ninguna excepción que rompa la atomicidad al hacer *commit* de la transacción.
2. Todas las líneas de caché accedidas deben estar presentes en L1 con los permisos de acceso necesarios para ejecutar la transacción sin tener que generar o esperar a la respuesta de mensajes de coherencia adicionales. Para líneas de caché accedidas solamente por lecturas, esto ya se garantiza si se cumple la primera condición (que exista un `XEND` no especulativo), pues todas las lecturas de la transacción se habrán ejecutado, obteniendo el dato que requieren sin generar ninguna excepción. Para las líneas de caché accedidas por escrituras, se debe comprobar que todas las peticiones de acceso exclusivo (Subsubsección III-B.2) generadas al ejecutar escrituras se han completado.

Durante el proceso de *commit* de la transacción retrasamos todas las peticiones externas a L1 (similar a activar un cerrojo en caché, mencionado en la Subsección II-B, pero para toda L1 en vez de líneas individuales) y permitimos que `XBEGIN` haga *commit*, seguido por el resto de instrucciones de la transacción. Realmente, solamente es necesario retrasar las peticiones externas a las líneas escritas en la transacción. Sin embargo, para evitar tener que marcar el conjunto de líneas escritas en L1 con bits adicionales, optamos por retrasar las peticiones a cualquier línea de L1. Además, sabemos que la fase de *commit* debería ser rápida, dado que todas las instrucciones han sido ejecutadas y ya se ha pagado la latencia requerida para traer los bloques accedidos a L1.

Cuando `XEND` llega a la cabeza del ROB, se comporta como una barrera de memoria, por lo que solamente puede hacer *commit* cuando todas las escrituras de la transacción se han completado. Tras hacer commit, se pueden dejar de retrasar las peticiones

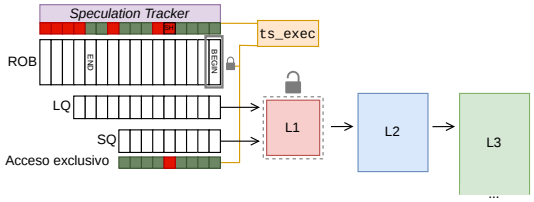


Fig. 1: Implementación de LHTM en un procesador fuera de orden

externas.

B.5 *Aborts*

En LHTM, llevamos a cabo *aborts* descartando todas las entradas del ROB desde `XBEGIN`, que se encuentra en la cabeza de ROB. Además de los *aborts* por conflicto, podemos encontrar *aborts* explícitos (producidos por `XABORT`), para manejar excepciones, para manejar interrupciones o por capacidad. Los fallos por capacidad se generan cuando 1) la transacción no cabe en la ventana de instrucciones del procesador o 2) no se puede acceder a la vez a todas las líneas de caché de la transacción en L1.

C. Cambios a la microarquitectura

A continuación listamos todas las modificaciones necesarias para añadir soporte para LHTM en un procesador fuera de orden. Incluimos un esquema que resume los cambios propuestos en la Figura 1.

En primer lugar, se debe retrasar el *commit* de `XBEGIN` hasta que se cumplan las condiciones necesarias para iniciar la fase de *commit* de la transacción. Es necesario dedicar un pequeño registro de 2 bits para guardar el estado de la transacción: `ts_none` cuando no se está en una transacción, `ts_exec` cuando `XBEGIN` se encuentra en la cabeza del ROB y `ts_commit` en la fase de *commit* de la transacción.

Se debe permanecer en `ts_exec` hasta encontrar un `XEND` no especulativo. En [17] proponen un *Speculation Tracker* (ST) que permite identificar las instrucciones no especulativas en el ROB. El ST requiere un bit adicional por entrada del ROB (para marcar las instrucciones *resueltas*) y un puntero a la *Speculation Head* (SH) (la siguiente instrucción candidata a ser no especulativa) [17]. En LHTM utilizamos el mismo mecanismo para recorrer las entradas del ROB conforme se convierten en no especulativas hasta encontrar un `XEND`, encontrar una excepción o agotar el número de entradas del ROB. En estos dos últimos casos, se debe abortar la transacción prematuramente. También se debe abortar la transacción cuando se detenga la etapa de *fetch* del procesador (para manejar una interrupción o por falta de capacidad en las estructuras del núcleo) si todas las instrucciones en vuelo están en estado no especulativo y no se ha encontrado un `XEND`. Al haberse detenido *fetch*, sabemos que no se va a insertar un `XEND` que permita iniciar el *commit* de la transacción y que desbloquee la cabeza del ROB.

Además, se debe añadir un bit por entrada de la SQ para comprobar que todas las peticiones de acceso exclusivo de las escrituras se han completado antes de iniciar el *commit* de la transacción.

Ancho del cauce	5 *fetch*/5 *decode*/5 *rename*/10 *dispatch*/ 10 *issue*/10 *commit*
Registros físicos	180 enteros + 180 punto flotante
ROB	352 entradas
LQ	128 entradas
SQ	72 entradas
Predictor de saltos	LTAGE [18]
Predicción de dependencias de memoria	*Store sets* [19]
L1 instrucciones	32KiB, 8 vías, 1 ciclo de latencia
L1 datos	48KiB, 12 vías, 1 ciclo de latencia
L2	512KiB, 8 vías, 10 ciclos de latencia de acceso
L3	Compartida, 4MiB por núcleo, 16 vías, 45 ciclos de latencia de acceso
Memoria	80 ciclos de latencia de acceso
Coherencia	Protocolo MESI

Tabla I: Configuración del sistema simulado

El conjunto de líneas de caché accedidas se puede calcular gracias a la LQ y SQ, para generar un *abort* por capacidad al exceder el límite de asociatividad de L1.

Es necesario que la caché genere notificaciones de *downgrade* para detectar conflictos cuando un procesador remoto intente leer una dirección escrita por la transacción local.

Finalmente, se debe poder activar un estado en L1 que retrase el procesamiento de todas las peticiones externas en el *commit* de la transacción.

D. Limitaciones

LHTM requiere contener la transacción al completo en la ventana de instrucciones del procesador, por lo que introduce varias restricciones que resumimos a continuación. Si una transacción no cumple estas restricciones, se generará un *abort*.

- No pueden encontrarse llamadas al sistema, excepciones o interrupciones que impidan iniciar la fase de *commit* de la transacción.
- La ventana de instrucciones del procesador (ROB, LQ, SQ, ...) debe tener capacidad para todas las instrucciones desde `XBEGIN` hasta `XEND`.
- Para asegurar que todas las líneas de caché accedidas se pueden cargar a la vez en L1, el número de líneas de caché accedidas no debe superar el límite de asociatividad de L1.

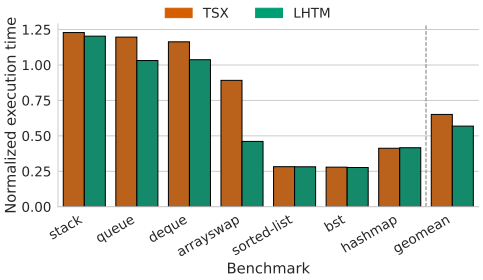


Fig. 2: Tiempo de ejecución normalizado sobre la referencia con cerrojos

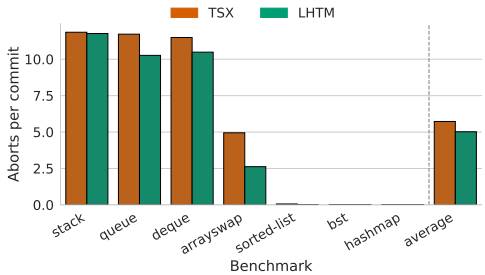


Fig. 3: *Aborts* por transacción completada

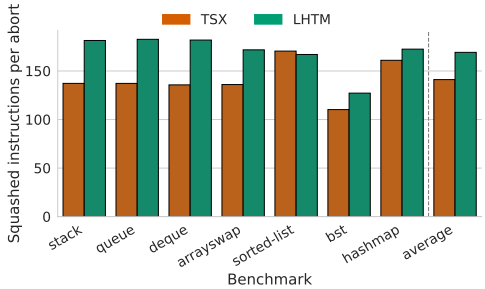


Fig. 4: Media de instrucciones descartadas del ROB en cada *abort*

IV. Metodología

Desarrollamos la implementación en gem5 [9] en modo *full-system*, con el sistema operativo Ubuntu 24.04 para el ISA x86-64. Definimos un procesador con 32 núcleos, cada uno configurado siguiendo los parámetros de un Intel Ice Lake [20] (Tabla I). Simulamos la jerarquía de memoria con Ruby y la red de interconexión con GARNET [21].

Para evaluar nuestra propuesta, ejecutamos *benchmarks* de estructuras de datos concurrentes [22]. Mostramos los resultados tras ejecutar una mezcla de operaciones (10 % inserciones, 10 % eliminaciones y 80 % búsquedas) para la lista enlazada, *hash map* y el árbol de búsqueda. Para la pila, cola, *deque* y arrayswap solamente se realizan modificaciones.

Para cada benchmark, normalizamos el tiempo de ejecución obtenido dividiendo entre el tiempo requerido por una ejecución con cerrojos. Comparamos una implementación de memoria transaccional con características similares a las del TSX de Intel [1] contra nuestra propuesta. El código del manejador que inicia y termina las transacciones en TSX y LHTM es idéntico. Este manejador incluye un tiempo de espera entre reintentos para reducir el número de *aborts* y adquiere un cerrojo *fallback* tras alcanzar el límite de reintentos. Establecemos el límite de reintentos en 6 para TSX y 5 para LHTM (con estos parámetros se obtiene el menor tiempo de ejecución medio). Mostramos la media de 10 ejecuciones para cada configuración, cada una iniciada desde un *checkpoint* distinto tras una espera de duración aleatoria, para introducir variabilidad en la simulación.

V. Resultados experimentales

En esta sección comparamos la configuración de referencia (basada en cerrojos), la configuración que imita a TSX y nuestra propuesta.

A. Tiempo de ejecución

La Figura 2 muestra el tiempo de ejecución normalizado de todos los *benchmarks*. Los *benchmarks* de *deque*, cola y pila constituyen pruebas con un alto número de conflictos para las HTM, pues es difícil conseguir una mejora a través del paralelismo en estas estructuras. En *arrayswap*, donde se permiten más accesos en paralelo, las HTM consiguen superar el rendimiento de la versión con cerrojos de grano fino. En todos los *benchmarks*, LHTM consigue un tiempo de ejecución menor que TSX. Esta mejora del tiempo de ejecución (muy notable en *arrayswap*) se debe a las prebúsquedas exclusivas que LHTM genera al ejecutar escrituras. En estos *benchmarks* con transacciones muy cortas (especialmente las de *arrayswap*), al pedir la línea de caché al ejecutar las escrituras, en vez de hacerlo en *commit* (siendo este el comportamiento convencional, que sigue TSX), se reduce el número final de *aborts*, y esto a su vez se ve reflejado en el tiempo de ejecución.

B. Aborts por transacción completada

En la Figura 3 mostramos la media de *aborts* por transacción completada en cada *benchmark*. Dado que ejecutamos muchas operaciones de lectura en los *benchmarks* con pocos conflictos y que incluimos un periodo de *backoff* entre reintentos, apenas encontramos *aborts* en la lista ordenada, árbol de búsqueda binario y *hashmap* (una media de 0,065; 0,008 y 0,008 *aborts* por transacción, respectivamente, para TSX). En los *benchmarks* con más conflictos, TSX genera un 13,87 % más de *aborts* al realizar las peticiones de las escrituras en *commit*. La gran mayoría de *aborts* son causados por conflictos, ya que las transacciones cortas que empleamos no suelen fallar por *interrupt* o por superar el límite de capacidad.

C. Instrucciones descartadas

La Figura 4 muestra el número medio de instrucciones que son descartadas del ROB en cada *abort*. Aunque TSX dispone de un *checkpoint* para recuperar el estado de los registros, tiene que descartar todas las instrucciones del cauce antes de poder utilizarlo, por lo que sigue necesitando descartar un número considerable de instrucciones similar al de LHTM. Sin embargo, TSX puede empezar a hacer *commit* de las instrucciones en el ROB antes de iniciar la fase de *commit* de la transacción. Como consecuencia, la ocupación del ROB suele ser menor en TSX cuando se produce un *abort*.

VI. Trabajo relacionado

En una investigación inicial para desarrollar la *Advanced Synchronization Facility* (ASF) de AMD, se consideró una implementación *in-core* similar a nuestra propuesta, pero acabó siendo descartada [23]. El objetivo de ASF era garantizar el progreso del programa en escenarios sin conflictos ni excepciones, por lo que no podía permitirse *aborts* por falta de capacidad en el *hardware* para ejecución fuera de orden del procesador [23].

Nagabhiru y Byrd [24] defienden la importancia de la HTM para la programación con *multi-address compare-and-swap* (MCAS). Su propuesta garantiza el progreso del programa sin requerir una alternativa *fallback*, pero introduce cambios importantes en el protocolo de coherencia de caché.

Finalmente, Kafousis [25] también aboga por la HTM limitada como una primitiva atómica universal para varias direcciones de memoria. A diferencia de LHTM, proponen modificaciones centradas en la caché de primer nivel.

VII. Conclusiones

Cuando las propuestas iniciales para la HTM empezaron a demostrar beneficios prácticos, varias compañías incorporaron soporte para HTM en sus procesadores. Con el tiempo, el interés ha disminuido y varias de estas implementaciones han sido desactivadas. Es probable que la desventaja que supone una implementación *hardware* compleja, que beneficia a un número limitado de casos de uso, haya influido en gran medida en esta decisión.

Con LHTM demostramos que, con pocas modificaciones, se pueden obtener muchas de las ventajas de la HTM, especialmente en el caso de transacciones pequeñas. Además, al reutilizar el soporte para la especulación dentro del núcleo, reducimos la superficie de ataque de posibles vulnerabilidades que podrían afectar a otras implementaciones centradas en la caché.

Agradecimientos

Este trabajo ha sido financiado por el Consejo Europeo de Investigación (ERC) bajo el programa de investigación e innovación Horizonte 2020 de la Unión Europea (ECHO, acuerdo de ayuda N. 819134), bajo MCIN/AEI/10.13039/501100011033/ y "FEDER: Una manera de hacer Europa", UE (DAMAS, ayuda PID2022-136315OB-I00). Álvaro Rubira-García ha sido financiado mediante la ayuda 22932/FPI/25 de la Fundación Séneca, Agencia de Ciencia y Tecnología de la Región de Murcia.

Referencias

[1] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual*, Intel Corporation, 2025.

[2] ARM holdings plc, "Arm® Architecture Reference Manual Supplement, Transactional Memory Extension (TME), for A-profile architecture," 2022, Accessed: 2025-10-26.

[3] Amy Wang, Matthew Gaudet, Peng Wu, José Nelson Amaral, Martin Ohmacht, Christopher Barton, Raul Silvera, and Maged Michael, "Evaluation of Blue Gene/Q hardware support for transactional memories," in *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2012, pp. 127–136.

[4] Luis Ceze, James Tuck, Calin Cascaval, and Josep Torrellas, "Bulk disambiguation of speculative threads in multiprocessors," in *33rd Int'l Symp. on Computer Architecture (ISCA)*, June 2006, pp. 227–238.

[5] Dave Christie, Jae-Woong Chung, Stephan Diestelhorst, Michael Hohmuth, Martin Pohlack, Christof Fetzer, Martin Nowack, Torvald Riegel, Pascal Felber, Patrick Marlier, and Etienne Rivière, "Evaluation of AMD's advanced synchronization facility within a complete transactional memory stack," in *Proceedings of the 5th European conference on Computer systems*, Paris France, Apr. 2010, pp. 27–40, ACM.

[6] Ravi Rajwar and James R. Goodman, "Speculative lock elision: Enabling highly concurrent multithreaded execution," in *34th Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2001, pp. 294–305.

[7] Chester Lam, "Intel's Netburst: Failure is a Foundation for Success," `https://chipsandcheese.com/p/intels-netburst-failure-is-a-foundation-for-success`, June 2022, Accessed: 2026-03-29.

[8] George Cozma, "Intel's lion cove architecture preview," `https://chipsandcheese.com/p/intels-lion-cove-architecture-preview`, Apr. 2025, Accessed: 2026-03-29.

[9] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, et al., "The gem5 simulator: Version 20.0+," *arXiv preprint arXiv:2007.03152*, 2020.

[10] James E. Smith and Andrew R. Pleszkun, "Implementing precise interrupts in pipelined processors," *IEEE Transactions on computers*, vol. 37, no. 5, pp. 562–573, 1988.

[11] Peter Sewell, Susmit Sarkar, Scott Owens, Francesco Zappa Nardelli, and Magnus O. Myreen, "x86-TSO: A rigorous and usable programmer's model for x86 multiprocessors," *Communications of the ACM*, vol. 53, no. 7, pp. 89–97, July 2010.

[12] Kourosh Gharachorloo, Anoop Gupta, and John Hennessy, "Two techniques to enhance the performance of memory consistency models," in *20th Int'l Conf. on Parallel Processing (ICPP)*, Aug. 1991, pp. 355–364.

[13] Ashkan Asgharzadeh, Juan M Cebrian, Arthur Perais, Stefanos Kaxiras, and Alberto Ros, "Free atomics: hardware atomic operations without fences," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 14–26.

[14] Ravi Rajwar, *Speculation-based techniques for transactional lock-free execution of lock-based programs*, Ph.D. thesis, University of Wisconsin-Madison, 2002.

[15] Intel Corporation, *Intel® 64 and IA-32 Architectures Optimization Reference Manual*, Intel Corporation, 2025.

[16] Hany Ragab, Enrico Barberis, Herbert Bos, and Cristiano Giuffrida, "Rage against the machine clear: A systematic analysis of machine clears and their implications for transient execution attacks," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1451–1468.

[17] Carlos Escuin, Paolo Salvatore Galfano, Davide Basilio Bartolini, Leeor Peled, and Mehdi Alipour, "Tempranillo: Non-Speculative Early Register Release," in *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Jan. 2026, pp. 1–17, ISSN: 2378-203X.

[18] André Seznec, "The L-TAGE branch predictor," *The Journal of Instruction-Level Parallelism*, vol. 9, pp. 1–13, May 2007.

[19] George Z. Chrysos and Joel S. Emer, "Memory dependence prediction using store sets," in *25th Int'l Symp. on Computer Architecture (ISCA)*, June 1998, pp. 142–153.

[20] Agner Fog, "Instruction tables: Instruction latencies, throughputs and micro-operation breakdowns," `http://www.agner.org/optimize/instruction_tables.pdf`, 2023, Accessed: 2025-05-07.

[21] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*, Apr. 2009, pp. 33–42.

[22] Nodari Kankava, "Exploring the efficiency of multi-word

compare-and-swap," Master's thesis, Uppsala University, 2020.
[23] Stephan Diestelhorst, Martin Pohlack, Michael Hohmuth, Dave Christie, Jae-Woong Chung, and Luke Yen, "Implementing AMD's Advanced Synchronization Facility in an out-of-order x86 core," 2010.
[24] Mahita Nagabhiru and Gregory T Byrd, "Achieving forward progress guarantee in small hardware transactions," *IEEE Computer Architecture Letters*, 2024.
[25] Konstantinos Kafousis, "Limited Read-Write/Set Hardware Transactional Memory without modifying the ISA or the Coherence Protocol," Bachelor's thesis, University of Crete, Sept. 2025.

Organización del Backend y Rendimiento en Procesadores RISC-V Fuera de Orden

Esther Alonso[1], Pablo Abad[1], Pablo Prieto[1] y Valentín Puente[1]

Resumen—
La función principal del *backend* es planificar las instrucciones del *frontend* sobre los recursos de ejecución, preservando la semántica secuencial. Optimizar este proceso es clave en arquitecturas superescalares fuera de orden, pues exige gestionar compromisos en la lógica de *issue* y en la organización de las unidades funcionales.

Los procesadores RISC-V recientes de altas prestaciones tienden a adoptar *backends* distribuidos, con múltiples puertos de ejecución, cada uno con cola de *issue* privada y planificador propio. Este enfoque contrasta con diseños convencionales basados en planificación unificada o parcialmente compartida entre puertos. Motivado por esta divergencia, este trabajo analiza el impacto en rendimiento del alcance de planificación y de la organización de recursos de ejecución. Se evalúan tres organizaciones de *backend*: (i) una cola de *issue* unificada con planificador global, como cota superior idealizada de rendimiento; (ii) una organización completamente distribuida con planificación por puerto; y (iii) un *backend* replicado y segmentado en clústeres homogéneos, cada uno con cola de *issue* independiente y un conjunto completo de unidades funcionales. Mediante simulación sobre suites representativas de *benchmarks*, se explora el espacio de diseño bajo supuestos ideales y latencias realistas.

Los resultados muestran que, aunque la planificación unificada puede aportar hasta un 20 % de mejora frente a colas distribuidas si tienen latencias equivalentes, esta ventaja es muy sensible al retardo de planificación y casi desaparece al añadir un ciclo al bucle de *wakeup-select*. En contraste, los *backends* replicados muestran competitividad robusta: igualan a los diseños distribuidos con menos puertos de ejecución y se mantienen dentro de una degradación del 5-10 % incluso con penalizaciones conservadoras. Estos hallazgos sugieren que la replicación del *backend* es una alternativa viable y debería considerarse junto con organizaciones distribuidas convencionales en microarquitecturas RISC-V.

Palabras clave— **RISC-V, Arquitectura Fuera de Orden, Planificación de Instrucciones.**

I. Introducción

TRAS el fin del escalado de frecuencia hace aproximadamente dos décadas, la evolución de los procesadores de propósito general se orientó a arquitecturas *multicore*. No obstante, su aprovechamiento efectivo ha sido limitado por la complejidad creciente del *software* y la dificultad de explotar paralelismo a nivel de aplicación. En consecuencia, el paralelismo a nivel de instrucción (ILP) sigue siendo fundamental para el rendimiento y central en el diseño de microarquitecturas de altas prestaciones.

El auge del *hardware* abierto ha potenciado el interés en este ámbito, con RISC-V [1] como máximo exponente, impulsando un entorno que favorece la experimentación arquitectónica y la transferencia tecnológica entre la academia y la industria. Pese a su reciente introducción, el ecosistema RISC-V ha avanzado rápidamente y evoluciona hacia niveles de rendimiento equiparables a los de las arquitecturas comerciales establecidas. Además, su naturaleza abierta fomenta tanto la cooperación internacional en el diseño de arquitecturas de alta complejidad como la asimilación de metodologías de desarrollo ágiles, inspiradas en las prácticas de la ingeniería de *software* [2], [3].

Esta evolución se observa en procesadores superescalares RISC-V recientes de SiFive [4], Xuantie (Alibaba) [5] y proyectos abiertos como XiangShan [3]. En todos ellos, la complejidad microarquitectónica es comparable a la de *cores* de altas prestaciones de proveedores tradicionales como Intel y AMD. Estos procesadores incorporan *pipelines* fuera de orden agresivos (con 6-8 vías), predictores de saltos sofisticados (p.ej. TAGE-SC-L [6]), numerosos puertos de ejecución (12 o más) y soporte escalar y vectorial. En este contexto, el diseño del *backend* depende de unas pocas decisiones con alto impacto. Entre ellas, destacan la lógica de *issue* y la organización de las unidades funcionales (FU).

La lógica de *issue* explota el ILP en procesadores fuera de orden y suele incluir dos bloques: la *Issue Queue* (IQ), donde las instrucciones esperan operandos y recursos, y la lógica de planificación, que selecciona instrucciones listas para enviarlas a unidades funcionales libres. Estas estructuras suelen estar en el camino crítico [7], [8], [9], [10], por lo que el equilibrio complejidad-rendimiento es clave para maximizar IPC y frecuencia. Las unidades funcionales se agrupan en puertos de ejecución, cuyo número fija cuántas instrucciones puede emitir la lógica de *issue* por ciclo. Esta organización también limita la complejidad de la lógica de *scheduling*, las redes de *bypass/forwarding* y los puertos del banco de registros [7].

El análisis de implementaciones RISC-V de altas prestaciones muestra una tendencia hacia el diseño de *backends* distribuidos. Por ejemplo, la microarquitectura P870 [4] implementa 13 puertos de ejecución; cada uno recibe instrucciones desde una cola de *issue* independiente (10-12 entradas) y dispone de un planificador dedicado con alcance restringido. Con variaciones de implementación, el C930 de Xuantie [5] y el Kunminghu de XiangShan [3] siguen una organización similar. Dada la complejidad de la lógica de *wakeup* y *select* en planificadores modernos [9], [10] y su impacto en el camino crítico [7], este enfoque ofrece ventajas de escalabilidad y control de complejidad.

[1]Dpto. de Ingeniería Informática y Electrónica, Universidad de Cantabria, e-mail: {alonsogae, abadp, prietop, vpuente}@unican.es

En contraste, proveedores consolidados como Intel y AMD parecen favorecer organizaciones de *backend* distintas en diseños recientes. La familia Zen de AMD ha evolucionado desde colas de *issue* distribuidas en generaciones tempranas [11] hacia un *backend* estructurado con tres colas dedicadas según el tipo de operación (enteros, memoria y punto flotante/vectorial), cada una con un planificador capaz de emitir múltiples instrucciones por ciclo en Zen 5 [12]. Arquitecturas recientes de Intel como Meteor Lake [13] y Lunar Lake [14] muestran una filosofía afín.

Más recientemente, Andes Technology introdujo su procesador RISC-V de altas prestaciones Cuzco [15], punto alternativo en este espacio de diseño. Aunque mantiene una cola de *issue* y un planificador por puerto, incrementa su funcionalidad. En particular, cada puerto replica el conjunto completo de operaciones de la ISA y reduce el número total de puertos.

La coexistencia de estas organizaciones de *backend*, desde diseños con planificación global hasta estructuras distribuidas o segmentadas con replicación, plantea preguntas sobre los compromisos arquitectónicos que guían su adopción. Aunque implementan la misma ISA, procesadores de distintos proveedores presentan organizaciones diferentes, lo que sugiere compromisos entre el tamaño de la ventana de planificación, la latencia del camino crítico y el coste de implementación.

Motivado por esta divergencia, este trabajo analiza cómo la organización del *backend* afecta al rendimiento de procesadores superescalares fuera de orden. Estudiamos el impacto del alcance de planificación y de la organización de recursos de ejecución en un conjunto representativo de aplicaciones. Primero, cuantificamos la distancia respecto a un *backend* idealizado con una única cola de *issue* unificada y alcance completo. Después, estimamos el coste de implementación modelando incrementos en la latencia de planificación asociados a mayor complejidad de estructuras centralizadas. Finalmente, evaluamos arquitecturas alternativas basadas en segmentación y replicación de recursos de ejecución, y situamos su rendimiento relativo bajo supuestos tanto ideales como más representativos.

Las principales contribuciones de este trabajo son las siguientes:

1. Una exploración del espacio de diseño del alcance de planificación en *backends* superescalares modernos, cuantificando la diferencia de rendimiento entre organizaciones de *issue* unificadas y distribuidas bajo supuestos ideales y sensibles a latencia.
2. Una evaluación de segmentos de *backend* completamente replicados como alternativa a organizaciones distribuidas por puerto, identificando cuándo la replicación iguala o supera diseños distribuidos con más vías de ejecución.
3. Un análisis de sensibilidad a la latencia de etapas críticas del *backend*, que demuestra que la ventaja de la planificación unificada es frágil ante retardos adicionales en el mecanismo de *wakeup-select*, mientras las organizaciones replicadas muestran mayor robustez frente a variaciones moderadas en la latencia de *writeback*.
4. Un análisis arquitectónico de los compromisos del escalado de recursos de ejecución en *cores* RISC-V de altas prestaciones, destacando la replicación como vía viable junto a *backends* distribuidos.

II. Background: Backend superescalar

Dentro del *backend* del procesador, algunos componentes afectan de forma no lineal al rendimiento y a la complejidad. Esta sección se centra en dos: la lógica de *issue* y las estructuras de *writeback* y *forwarding*, responsables de la planificación dinámica, la propagación de resultados y la resolución de dependencias. Se analiza su organización, escalabilidad y compromisos de diseño.

A. Issue

La lógica de *issue* despacha instrucciones desde la *Issue Queue* hacia las unidades funcionales disponibles. En procesadores fuera de orden, este proceso es dinámico: las instrucciones se emiten cuando sus operandos están listos. Esta etapa se gobierna por dos mecanismos acoplados, *wakeup* y *select*. *Wakeup* detecta qué entradas de la IQ ya tienen operandos disponibles tras la finalización de instrucciones productoras; *select* elige cuáles se emiten en el ciclo actual según la disponibilidad de recursos y política de planificación.

La literatura describe dos esquemas dominantes para implementar la lógica de *issue*. El primero, y más extendido (figura 1), combina RAM para almacenar el estado de instrucciones y CAM para comparar etiquetas. Cuando finalizan instrucciones productoras, sus etiquetas destino se difunden a toda la IQ y se comparan en paralelo para detectar operandos disponibles. El segundo enfoque usa una matriz de bits proporcional al número de instrucciones en vuelo y de registros físicos. En este caso, la finalización de instrucciones limpia los bits de la fila correspondiente, salvo los asociados a operandos fuente aún no disponibles. En ambos enfoques, el tamaño de la *Issue Queue* (IQ) y el ancho de emisión elevan la complejidad: los diseños CAM aumentan comparaciones y difusión de etiquetas, y los matriciales escalan mal por tamaño e interconexión. Por ello, las operaciones de *wakeup-select* se sitúan con frecuencia en el camino crítico y hacen de la planificación una etapa especialmente sensible en retardo crítico y energía [7].

Un amplio cuerpo de trabajo ha abordado estos desafíos. Abella et al. [16] presentan un estudio que recoge la mayoría de optimizaciones clásicas propuestas a comienzos de los 2000, cuando el aumento de anchos superescalares impulsó la investigación en lógica de *issue* escalable. Entre ellas destacan: filtrado de entradas vacías o ya preparadas para reducir comparaciones [17], desactivación selectiva de por-

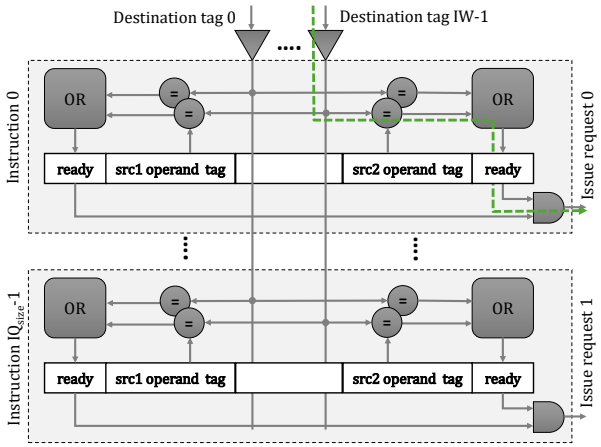


Fig. 1: Etapa de *issue*: camino crítico de *wakeup* en la IQ.

ciones de la *Issue Queue* (IQ) [17], [18], combinación de estructuras CAM con colas simples en orden [19], y optimización de implementaciones matriciales mediante particionado o reutilización estructural [20], [21]. Trabajos recientes retoman estas ideas con énfasis en limitar costes de temporización y energía, por ejemplo con colas particionadas [22], planificadores matriciales fragmentados [23] o lógica de *wakeup* optimizada [10], [24].

Si bien estas técnicas mejoran la escalabilidad y eficiencia de la lógica de *issue*, son en gran medida ortogonales a las decisiones arquitectónicas exploradas aquí. La mayoría se aplica tanto a colas con planificación global como a organizaciones distribuidas por puerto.

B. Ejecución (+ Writeback)

El número de instrucciones que pasa de *issue* a ejecución por ciclo impacta directamente en el dimensionamiento y coste del *backend*. Un caso claro es el banco de registros: al aumentar la emisión, crece el número de operandos leídos por ciclo, y con ello área y consumo energético. Algo similar ocurre al final del *backend*, donde el ancho de banda de *writeback* y la complejidad de las redes de *bypass* aumentan con las unidades funcionales activas de forma concurrente. Estas redes reenvían resultados entre unidades funcionales para reducir *stalls* por dependencias de datos.

Dado que el ILP de la mayoría de aplicaciones es limitado, existe un punto a partir del cual aumentar unidades funcionales simultáneas deja de mejorar rendimiento y solo incrementa el coste del *backend*. En procesadores fuera de orden contemporáneos, este compromiso se aborda agrupando unidades funcionales en puertos de ejecución, de modo que como máximo se emite una instrucción por puerto y ciclo. La distribución entre puertos busca maximizar el ancho de banda de ejecución y se ajusta al perfil de instrucciones de cargas de trabajo representativas [25]. Aun así, en diseños modernos con un alto grado de paralelismo la red de *bypass* sigue siendo crítica por su impacto en área, consumo y camino crítico. Por ello, muchas arquitecturas agrupadas restringen el reenvío a un subconjunto de caminos, normalmente dentro del mismo clúster, sin reenvío completo entre clústeres.

Mientras el banco de registros permanezca centralizado y compartido por todo el *backend* (o replicado en copias equivalentes), las limitaciones en la red de *bypass* afectan sobre todo a la latencia de resolución de dependencias. Si no hay camino de reenvío, las instrucciones consumidoras esperan a la escritura de operandos en el banco de registros y a su lectura posterior. Cuando el agrupamiento se extiende al propio banco de registros, se requieren mecanismos adicionales de comunicación entre clústeres para que un clúster consuma valores producidos en otro. Estos mecanismos pueden implementarse de forma explícita con búferes entre clústeres [26], [27] o de forma implícita mediante instrucciones especializadas de copia de registros [28].

En este trabajo se asume un único banco de registros centralizado en todas las configuraciones evaluadas. Bajo esta hipótesis, la ausencia de ciertos caminos de *bypass* se manifiesta solo como penalizaciones adicionales de latencia en casos específicos. Se evalúa una implementación idealizada con red de *bypass* completamente conectada y de un ciclo, y un diseño más realista en el que la latencia de reenvío aumenta progresivamente.

Hasta donde alcanza nuestro conocimiento, la literatura no ofrece una exploración sistemática del espacio de diseño de organizaciones de *backend* que contraste alcances de planificación unificados y particionados en procesadores fuera de orden contemporáneos; ese es precisamente el foco de este estudio.

III. Marco de Evaluación

A. Metodología

Utilizamos gem5 [29], [30] como herramienta principal, partiendo del *commit* 7a2b0e4 (v25.1.0.0) del repositorio oficial. Durante el proyecto identificamos una limitación en la política de *steering* por defecto para *backends* con IQ divididas, que producía asignaciones subóptimas a puertos. Implementamos un *steering* corregido, con una mejora media de aproximadamente el 5 % de IPC (hasta 10 % en SPECint).

Evaluamos cargas de trabajo de tres conjuntos de *benchmarks*, seleccionados para estresar núcleo y memoria: SPEC CPU2017 [31] (etiquetas SINT y SFP), SPLASH-4 [32] (SH4) y NAS Parallel Benchmarks [33] (NPB). SPEC CPU2017 incluye 23 aplicaciones (modo *rate*) de enteros y punto flotante, y cubre cargas representativas de escritorio, servidor y computación científica. SPLASH-4 se diseñó para multiprocesadores de memoria compartida; aunque ejecutamos 14 aplicaciones en modo secuencial, siguen siendo representativas de cargas científicas, de ingeniería y gráficas. Además, evaluamos 9 aplicaciones de NAS, derivadas de núcleos de dinámica de fluidos computacional. En cada aplicación, adelantamos la ejecución hasta su región de interés y realizamos simulación detallada ciclo a ciclo para un número fijo de instrucciones.

Consideramos dos configuraciones arquitectónicas

de referencia. La primera, BigO3, es un núcleo fuera de orden agresivo, comparable a diseños RISC-V recientes de alto rendimiento discutidos en la sección I [4], [5], [3]. Sus características principales, incluida la distribución de puertos, se resumen en las tablas I y II.

Tabla I: Configuración del procesador bajo prueba.

Parámetro	BigO3	SmallO3
Cache L1I	64 KB, 4 vías	32 KB, 4 vías
Pred. de saltos	TAGE-SC-L	TAGE-SC-L
Dec/Ren/Ejec/Ret	6/6/12/8	3/3/6/4
Tamaño del ROB	700 entradas	192 entradas
Banco de registros	228 (int), 240 (FP)	128 (int), 119 (FP)
Tamaño LSQ	196 (LD) / 64 (ST)	36 (LD) / 18 (ST)
Cache L1D	64 KB, 8 vías, acierto 4 ciclos	32 KB, 4 vías, acierto 4 ciclos
Cache L2	1 MB, 8 vías, acierto 12 ciclos	256 KB, 8 vías, acierto 12 ciclos
LLC	16 MB, 16 vías, acierto 40 ciclos	2 MB, 16 vías, acierto 40 ciclos

Tabla II: Organización de FUs para cada configuración.

Puerto	Tipo	BigO3	SmallO3
1	INT	ALU, Div	ALU, Div
2	INT	ALU, Mul	ALU, Mul
3	INT	ALU, Mul	ALU, Br
4	INT	ALU	-
5	INT	Br, ALU	-
6	INT	Br, ALU	-
7	FP	Add, Mul, MAC, Div, Sqrt	Add, Mul, MAC, Div, Sqrt
8	FP	Add, Mul, MAC	-
9	FP	Add, Mul, MAC	-
10	MEM	AGU, Ld	AGU, Ld
11	MEM	AGU, Ld, St	AGU, Ld, St
12	MEM	AGU, Ld, St	-

Por otro lado, definimos un segundo núcleo fuera de orden con menos recursos de ejecución, representativo de un diseño orientado a reducir complejidad y consumo. Esta configuración, SmallO3, está inspirada en la microarquitectura SiFive P550 [34]. Su diseño prioriza la eficiencia y reduce el paralelismo a nivel de instrucción, por lo que resulta representativo de núcleos de propósito general energéticamente eficientes. Las tablas I y II resumen sus características y distribución de puertos.

B. Organizaciones de backend evaluadas

Como se discutió en la Introducción, los procesadores contemporáneos exhiben una amplia variedad de organizaciones de *backend*. Por ello, agrupamos las alternativas en tres organizaciones, esquematizadas en la figura 2, que corresponden a las tres filosofías de diseño introducidas.

- IQ Única + Puertos Distribuidos: Inspirado en la planificación centralizada de muchos procesadores comerciales de altas prestaciones, este modelo emplea una única cola de *issue* global, de modo que cada puerto planifica sobre toda la cola. Las unidades funcionales se distribuyen entre puertos para maximizar el ancho de banda de ejecución, y el planificador puede despachar instrucciones preparadas a cualquier puerto elegible.
- IQ Dividida + Puertos Distribuidos: Representativa de procesadores RISC-V contemporáneos, esta configuración implementa una cola de *issue* independiente por puerto. La política de selección es intencionadamente sencilla: primero distribuye las instrucciones por tipo de operación y, cuando varios puertos ofrecen la unidad funcional requerida, selecciona según ocupación entre los puertos elegibles.
- IQ Dividida + Puertos Replicados: Esta configuración modela la organización con *backend* modular del procesador Cuzco [15]. Mantiene la IQ dividida del diseño anterior, pero replica un conjunto completo de unidades funcionales en cada puerto. Como consecuencia, la selección de puerto depende solo de la ocupación de la cola, ya que el tipo de instrucción deja de restringir la elegibilidad.

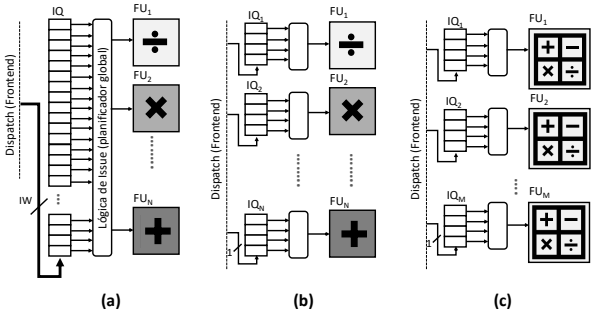


Fig. 2: Organizaciones de *backend* evaluadas. (a) IQ Única + Puertos Distribuidos, (b) IQ Dividida + Puertos Distribuidos, (c) IQ Dividida + Puertos Replicados.

Cada organización de *backend* presenta ventajas y desventajas. Las colas de *issue* distribuidas reducen complejidad y coste de planificación, pero pueden limitar rendimiento al restringir el número de instrucciones expuestas al planificador. En cambio, la replicación de unidades funcionales simplifica la planificación al eliminar restricciones por tipo de instrucción en la selección de puerto, pero introduce sobrecarga de área y puede aumentar coste y complejidad de otras estructuras del *backend*, como redes de *bypass* y *forwarding*. Nuestro objetivo es caracterizar cuantitativamente estos compromisos y su impacto práctico sobre el conjunto de aplicaciones.

IV. Resultados

Esta sección presenta la evaluación experimental de las organizaciones de *backend* introducidas anteriormente. Analizamos el impacto de la parametrización de la cola de *issue*, el tamaño de la ventana de planificación y la organización de los recursos de ejecución sobre el rendimiento. Cada subsección estudia una dimensión del espacio de diseño, primero bajo supuestos idealizados y después incorporando consideraciones de coste, para evaluar la robustez de las tendencias observadas.

A. Parametrización de la cola de issue

Un punto de partida sólido es esencial en una evaluación por simulación detallada. Aunque la simulación ciclo a ciclo ofrece una aproximación sólida, puede introducir pequeñas discrepancias frente a resultados de RTL o hardware real. Por ello, primero determinamos tamaños adecuados de cola de *issue* (IQ) para las implementaciones fuera de orden eva-

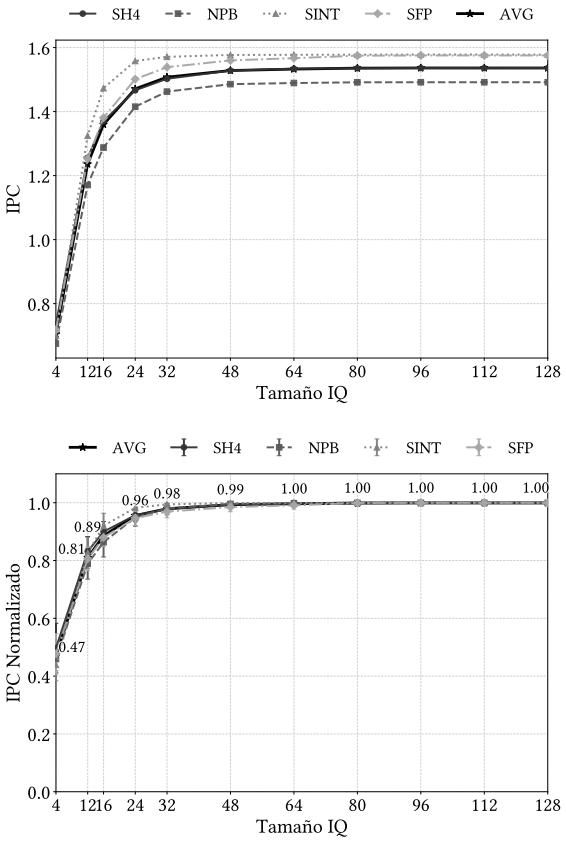


Fig. 3: Evolución del rendimiento con IQ única en función del tamaño de la IQ, configuración SmallO3. (arriba) IPC absoluto. (abajo) IPC normalizado al máximo.

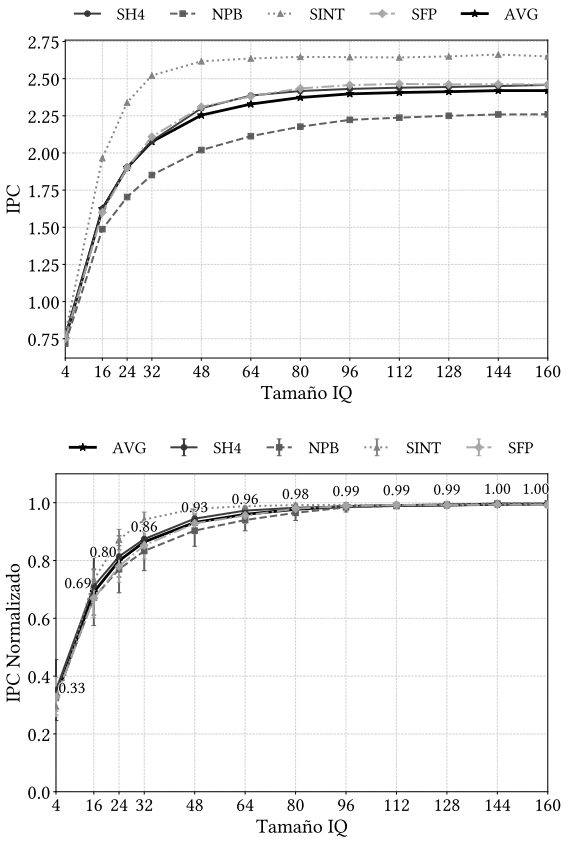


Fig. 4: Evolución del rendimiento con IQ única en función del tamaño de la IQ, configuración BigO3. (arriba) IPC absoluto. (abajo) IPC normalizado al máximo.

luadas y las configuraciones base de la sección III. Para ello, incrementamos progresivamente el tamaño de la IQ y analizamos la evolución del IPC hasta identificar la saturación de la curva IPC-tamaño de IQ.

Para cubrir distintos regímenes de esa curva, seleccionamos tres tamaños representativos: uno en la zona de mayor pendiente, otro en el codo, donde decrece la ganancia de rendimiento, y un tercero al inicio de la saturación, donde los *benchmarks* se estabilizan en torno al 99 % del IPC máximo. Considerar varios puntos permite obtener conclusiones robustas en un rango razonable de tamaños y evita depender de un solo diseño. Estos tres puntos se usan en el resto de la evaluación y representan compromisos cercanos al óptimo rendimiento-coste.

Las figuras 3 y 4 muestran los resultados para SmallO3 y BigO3. Para facilitar la lectura, el IPC se agrupa en cuatro series, cada una con el IPC medio de una suite de *benchmarks*; en SPEC CPU2017 se separan cargas enteras y de punto flotante. En ambas figuras se presentan IPC absoluto (arriba) e IPC normalizado al máximo (abajo). En los datos normalizados se incluyen barras de error con intervalo de confianza del 95 % entre aplicaciones de cada *benchmark*.

Los resultados confirman varias tendencias. Primero, el tamaño de la IQ es crítico para el rendimiento global. Segundo, BigO3 alcanza mayor IPC absoluto que SmallO3. Tercero, la sensibilidad al tamaño de IQ varía entre suites. En particular, NAS muestra mayor dependencia de la capacidad de IQ y mayor variabilidad interna, posiblemente por su menor IPC base y su naturaleza numérica, donde una ventana de planificación mayor permite explotar ILP adicional. En el extremo opuesto, SPECint alcanza su IPC máximo con IQ más pequeñas, probablemente por menor densidad relativa de dependencias de datos, lo que facilita encontrar instrucciones planificables incluso entre un conjunto de instrucciones menor.

Además, observamos una divergencia respecto a los tamaños de IQ usados en procesadores comerciales de referencia. Las capacidades sugeridas por nuestras curvas IPC-tamaño de IQ tienden a ser algo menores que las empleadas en la práctica. Lejos de cuestionar el análisis, esto sugiere que existen factores arquitectónicos adicionales que condicionan las decisiones reales. Como veremos más adelante, esta discrepancia se explica principalmente por la organización del *backend* y el tamaño de la ventana de planificación.

Siguiendo esta política de selección, los tamaños de IQ usados en el resto de la evaluación son:

Para BigO3, 96, 48 y 24 entradas, equivalentes aproximadamente al 99 %, 93 % y 80 % del IPC máximo observado.

Para SmallO3, 48, 24 y 12 entradas, equivalentes aproximadamente al 99 %, 96 % y 81 % del IPC máximo observado.

Con estos tamaños parametrizados, la siguiente sección cuantifica la pérdida de rendimiento al pa-

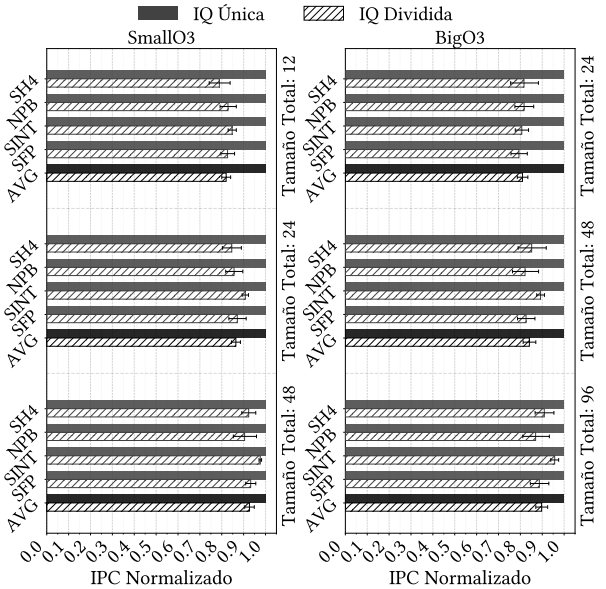


Fig. 5: Degradación de IPC para implementaciones de IQ distribuida.

sar de una IQ unificada idealizada a un modelo distribuido con una cola de *issue* por puerto de ejecución, correspondiente a la organización IQ dividida + Puertos Distribuidos de la sección III-B.

B. *Evaluación de la cola de issue única frente a dividida*

Como se discutió en la sección IV-A, usamos la configuración con IQ única como cota superior de rendimiento, ignorando sus implicaciones físicas. El primer experimento cuantifica la pérdida causada exclusivamente por particionar la IQ en segmentos independientes asociados a puertos de ejecución. Aunque implementaciones reales pueden usar capacidades distintas según el tipo de operaciones de cada puerto, nuestro análisis muestra que ese ajuste aporta mejoras menores frente a los efectos estructurales estudiados. Por claridad, asumimos una partición uniforme de la capacidad total entre puertos.

Repetimos las configuraciones de la sección anterior y recogemos métricas para los tres tamaños de IQ seleccionados. La figura 5 presenta los resultados para SmallO3 (izquierda) y BigO3 (derecha). Cada barra representa el rendimiento medio de una suite de *benchmarks*, normalizado respecto al obtenido con IQ única.

Aparece una tendencia consistente en todos los *benchmarks* y tamaños de IQ: la partición por puerto introduce una pérdida significativa frente a la cola unificada. En ambas configuraciones, la degradación es mayor con IQs pequeñas, donde la capacidad es más crítica para sostener el IPC. Al aumentar el tamaño total de IQ, la brecha entre organización unificada y distribuida disminuye gradualmente: pasa de alrededor del 20 % en la configuración más pequeña evaluada (2 entradas por cola) a aproximadamente el 10 % en la mayor. De forma destacable, SPECint prácticamente iguala la cola unificada cuando las colas distribuidas alcanzan capacidad suficiente, coherente con su rápida saturación en la sección anterior.

Como se anticipó, esta pérdida ayuda a explicar la discrepancia entre tamaños de saturación de IQ en simulación y tamaños implementados en procesadores comerciales. Una interpretación plausible es que los diseños reales compensan el menor alcance de planificación de las colas divididas sobredimensionando su capacidad para recuperar parte del rendimiento. Para evaluarlo, realizamos un experimento adicional que determina el incremento de capacidad necesario para cerrar la brecha; en ese escenario, los tamaños resultantes se aproximan a los observados en procesadores contemporáneos.

Los resultados anteriores muestran que, si se ignoran costes de implementación, el mayor alcance de planificación de una IQ unificada puede aportar hasta un 10 % de mejora cuando la cola se dimensiona cerca del punto de saturación IPC-tamaño. Sin embargo, la lógica de planificación está entre los componentes más críticos en temporización de un procesador fuera de orden [7], y el tamaño de la IQ es uno de los factores que más influye en su latencia. Por tanto, asumir latencias similares para colas unificadas y distribuidas no es realista.

A la vez, estimar con precisión el sobrecoste de una IQ unificada es complejo, porque la latencia depende de decisiones de bajo nivel (optimizaciones microarquitectónicas, *place-and-route*, tecnología, etc.) fuera del alcance de este trabajo. En lugar de fijar un coste específico, realizamos un análisis de sensibilidad para estudiar cómo evoluciona el rendimiento al incrementar la latencia de planificación en la cola unificada. Esto permite identificar la rapidez con que se erosiona su ventaja y los rangos donde sigue siendo beneficiosa.

Modelamos incrementos progresivos en la latencia de planificación de la cola unificada. Primero, asumimos latencias idénticas de uno y dos ciclos para organizaciones unificada y distribuida. Después añadimos ciclos adicionales solo al *pipeline* de planificación unificado, para determinar el punto de convergencia con la cola distribuida.

Las figuras 6 y 7 muestran resultados para SmallO3 y BigO3. El rendimiento se normaliza al diseño con mejor comportamiento en cada escenario (cola unificada con latencia mínima de planificación: 1 ciclo en las gráficas izquierdas y 2 ciclos en las derechas).

En SmallO3, el impacto de la latencia de planificación es claro: un único ciclo adicional en la cola unificada elimina casi toda su ventaja frente al diseño distribuido. Para IQ de 12 entradas, el mayor alcance de planificación aún compensa parcialmente, con alrededor de un 5 % de ganancia. Pero ese beneficio cae al aumentar la IQ. Con 24 entradas, ambas organizaciones son casi indistinguibles y la distribuida llega a superar a la unificada en algunas suites (Splash y SPECint). Con 48 entradas, esta inversión se vuelve consistente en todas las suites. Al aumentar más la penalización, el diseño unificado deja rápidamente de ser competitivo. Estas tendencias se mantienen con

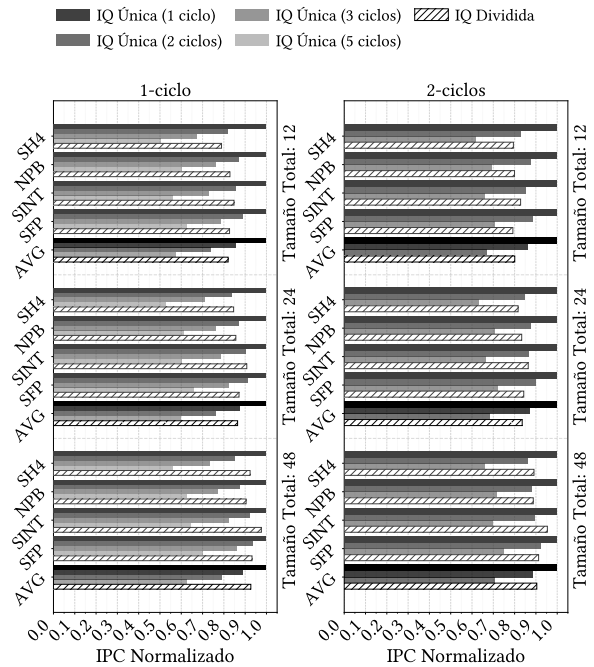


Fig. 6: Degradación de IPC para la IQ con retardos crecientes. Configuración SmallO3.

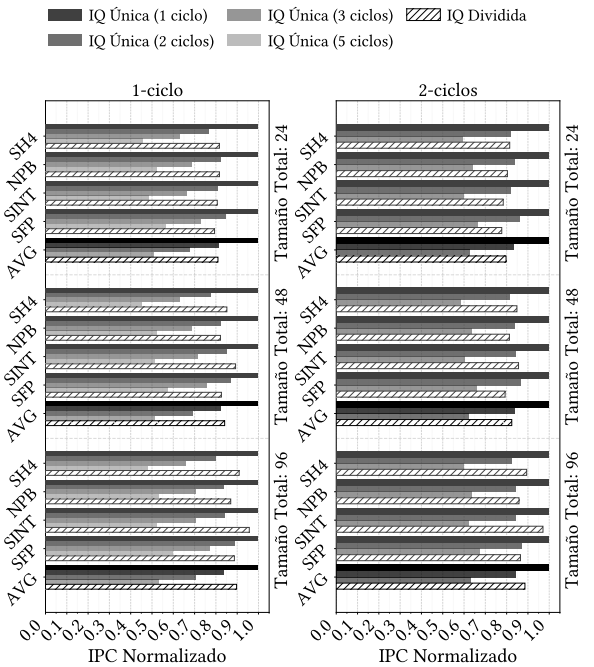


Fig. 7: Degradación de IPC para la IQ con retardos crecientes. Configuración BigO3.

latencia base de 1 o 2 ciclos, lo que indica los beneficios de un mayor alcance de planificación no logran compensar el impacto del retardo adicional.

En BigO3, el impacto es ligeramente mayor. En IQs pequeñas, la cola unificada apenas explota su ventaja teórica tras añadir un ciclo adicional. Desde el tamaño mínimo de IQ, la tendencia es más acusada que en SmallO3: un ciclo extra ya produce ligera degradación media en el tamaño intermedio y supera el 5 % en el tamaño mayor. Penalizaciones adicionales hacen claramente inferior al planificador unificado. Como en SmallO3, la tendencia es casi independiente de la latencia inicial asumida.

En conjunto, estos resultados indican que una IQ unificada ofrece hasta un 10 % de ventaja frente a colas distribuidas cuando se dimensiona cerca de saturación y se asume igualdad de costes. No obstante, esa ventaja es frágil: un incremento mínimo en la latencia de planificación contrarresta gran parte de las ganancias del mayor alcance. Esto evidencia un compromiso central del *backend*: aunque la planificación global maximiza teóricamente el ILP, su beneficio práctico depende de mantener un tiempo de ciclo estricto en el *wakeup-select*. Con recursos limitados de optimización a nivel de circuito, optar por planificación unificada puede introducir riesgo de implementación y beneficios netos inciertos.

Dado que las colas distribuidas emergen como alternativa robusta y competitiva bajo hipótesis realistas en cuanto a restricciones temporales, como sugiere la transición de ARM Neoverse V1 a V2 [35], resulta natural explorar si organizaciones alternativas de unidades funcionales mejoran aún más el equilibrio entre rendimiento y complejidad.

C. *Puertos de ejecución replicados frente a distribuidos*

Esta sección amplía el análisis a organizaciones de *backend* segmentadas con unidades funcionales replicadas [15], [36], [37], para evaluar si ofrecen un compromiso más favorable entre alcance de planificación, sensibilidad a latencia y coste. Como antes, dividimos la exploración en dos etapas. Primero, evaluamos el rendimiento al aumentar el número de *backends* replicados, sin modelar explícitamente los costes de implementación. De esta forma cuantificamos los segmentos necesarios para igualar a un *backend* heterogéneo convencional. Después incorporamos hipótesis de coste para estudiar la evolución del rendimiento.

Añadir segmentos impacta en el área dedicada al *backend* y a las redes de *writeback* y *bypass*. El coste final depende de decisiones de bajo nivel, restricciones físicas y política de direccionamiento de instrucciones. Para capturar esa variación de complejidad, analizamos la sensibilidad del rendimiento frente a retardos en *forwarding* y *writeback*.

La figura 8 muestra resultados para ambas configuraciones, normalizados al diseño de referencia (IQ Única + Puertos Distribuidos). SmallO3 y BigO3 muestran tendencias similares. Con IQ pequeñas, donde la organización distribuida deja colas por puerto poco profundas, la configuración replicada alcanza el rendimiento de referencia con pocos segmentos. Esto puede explicarse por dos factores: (i) la replicación reduce contención en las unidades funcionales y (ii) con menos segmentos que puertos, cada segmento mantiene una cola local más profunda para igual capacidad agregada, aumentando la capacidad efectiva de planificación por segmento.

Fuera de esa región, al crecer la capacidad total

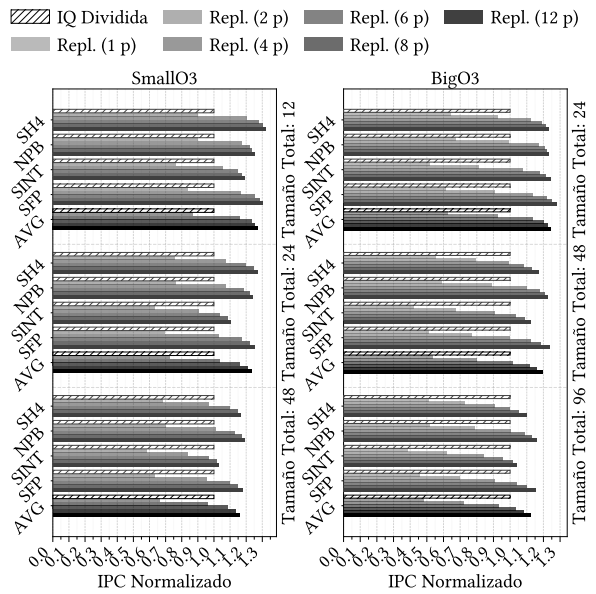


Fig. 8: Evolución del IPC al incrementar el número de segmentos replicados en el *backend*.

de IQ y diluirse la ventaja por profundidad, emerge un patrón claro: el *backend* replicado supera la configuración de referencia distribuida cuando el número de segmentos iguala el ancho del *frontend*. En nuestras configuraciones, esto ocurre con tres segmentos en SmallO3 y seis en BigO3, con mejoras del 5-10 % respecto a la referencia. Estos valores son menores que el ancho de *retire* del *backend* distribuido (4 y 8) y bastante menores que su número de puertos (6 y 12). Una explicación posible es que los *backends* distribuidos convencionales sobredimensionan puertos para minimizar riesgos estructurales y evitar cuellos de botella en ejecución. En contraste, la replicación completa elimina contención intra-segmento. Cuando los segmentos igualan la tasa de suministro del *frontend*, el *backend* puede sostener rendimiento comparable sin requerir tantos puertos ni ancho de retirada. Finalmente, si el número de segmentos replicados iguala el número de puertos de la línea base, IQ Única + Puertos Replicados aumenta la disponibilidad de unidades funcionales y reduce la contención, lo que lleva a mejoras del 15-25 %.

Como en la sección anterior, el coste de implementación es clave para interpretar estos resultados. A diferencia de la comparación entre cola global y distribuida, donde el sobrecoste del planificador unificado es evidente, aquí el balance es menos claro. Por un lado, la replicación de puertos conlleva un mayor consumo de área en el *backend*. Por otro, igualar el rendimiento de referencia suele requerir menos segmentos replicados que puertos distribuidos, lo que potencialmente reduce el tamaño y complejidad de las redes de *writeback* y *bypass*, cuyo coste crece típicamente de forma cuadrática con el número de puertos. Como una estimación precisa requeriría implementación física detallada, adoptamos de nuevo una exploración del espacio de diseño bajo distintas hipótesis de coste para la organización replicada.

En la fase final consideramos, para IQ Única + Puertos Replicados, tres anchos de ejecución:

- Número de segmentos igual al ancho de *decode/-rename* (tabla I): seis para BigO3 y tres para SmallO3.
- La configuración inmediatamente inferior: cuatro segmentos en BigO3 y dos en SmallO3.
- Número de segmentos equivalente al ancho de *retire* (tabla I): ocho para BigO3 y cuatro para SmallO3.

Para estos casos analizamos la evolución del rendimiento bajo dos hipótesis de latencia opuestas:

1. El retardo de *writeback* y *forwarding* es mayor en el *backend* replicado, asumiendo que el área del segmento domina la latencia.
2. El retardo de *writeback* y *forwarding* es menor en el *backend* replicado, asumiendo que la complejidad de la red de *bypass* y *forwarding*, es mayor en la organización distribuida por su mayor número de puertos, domina la latencia.

Para modelar estos escenarios, variamos la latencia de *writeback* y *forwarding* de la configuración replicada en ±1 ciclo, manteniendo fija la organización distribuida. Es una aproximación conservadora, pues el retardo adicional se aplica uniformemente a todas las instrucciones, mientras que en la práctica afectaría solo a un subconjunto.

La figura 9 muestra los resultados, normalizados al *backend* distribuido convencional. Para la configuración replicada se muestran tres curvas por *benchmark*, correspondientes a las tres variantes de latencia de *writeback*: un ciclo menos (−1), referencia y un ciclo adicional (+1). Los marcadores distinguen el número de puertos en la versión replicada.

Al aumentar la latencia de *writeback*, el IPC normalizado tiende a reducirse, mientras que al disminuirla se recupera rendimiento de forma consistente. En términos cualitativos, esta sensibilidad es comparable a la observada al penalizar la latencia de planificación, lo cual resulta coherente: en ambos casos se retrasa un ciclo la resolución de dependencias, ya sea en la selección/emisión (planificación) o en la difusión del resultado (*writeback*).

Analizando por separado cada configuración de segmentos, comenzando por el menor número (inferior al ancho de *fetch/decode*), el *backend* replicado permanece competitivo siempre que su latencia de *writeback* no supere a la de la organización convencional. Además, si logra completarlo un ciclo antes, esta configuración reducida puede ofrecer mayor rendimiento incluso con tamaños de IQ elevados, donde la organización segmentada suele perder competitividad. Estas observaciones sugieren que la replicación de recursos de ejecución altera la dinámica de escalado del *backend* de forma más profunda que una simple comparación basada en el número de puertos.

Cuando el número de segmentos coincide con el ancho del frontend, la figura 8 ya mostró que el *backend* replicado iguala el rendimiento de la línea base con latencias equivalentes. Si el *writeback* se comple-

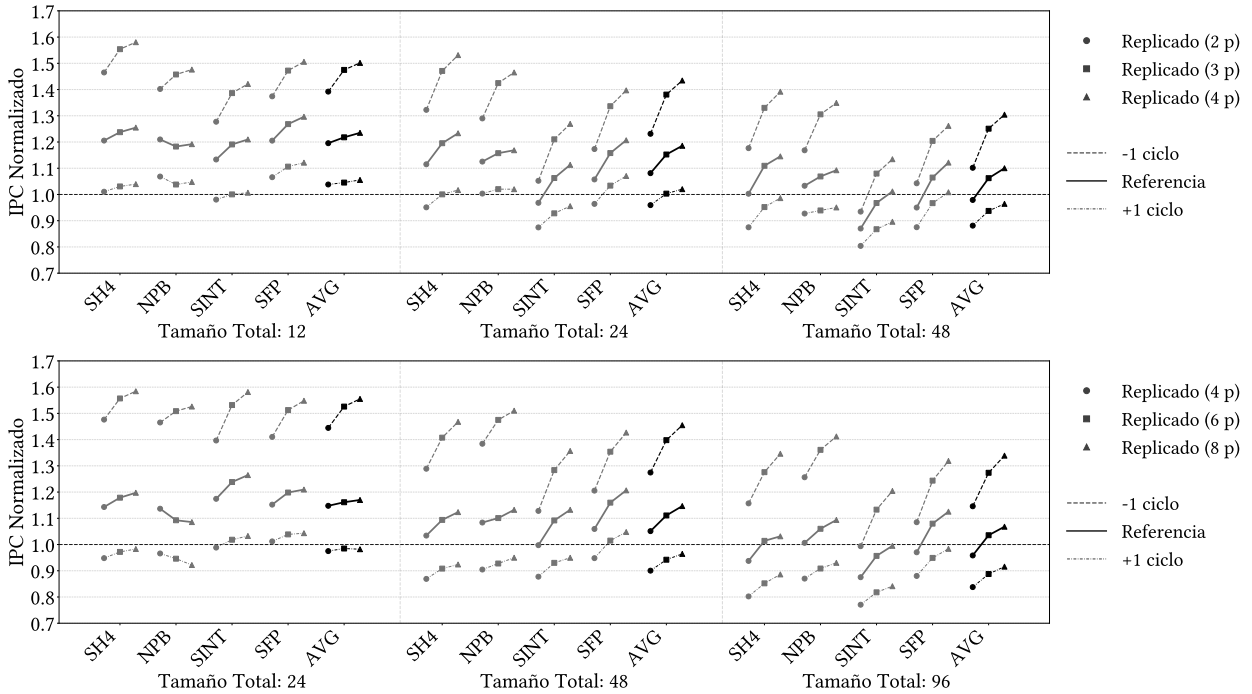


Fig. 9: Evolución del IPC con latencia variable de *writeback* para IQ Única + Puertos Replicados. Configuración SmallO3 (arriba) y BigO3 (abajo).

ta un ciclo antes, las ganancias alcanzan el 25–50 %, un límite optimista que pone de relieve el impacto cuadrático del escalado de puertos sobre los caminos críticos. El caso más informativo es la hipótesis opuesta, donde asumimos que el incremento de área por la réplica de recursos implica un ciclo adicional en el *backend* segmentado. Con tamaños de IQ reducidos, la configuración replicada sigue siendo competitiva, ya que la combinación de mayor profundidad por segmento y menor contención compensa la latencia añadida. Al aumentar la capacidad total de IQ y reducirse esa ventaja, el rendimiento replicado se degrada: en SmallO3 la pérdida permanece por debajo del 5 % incluso en la mayor IQ, mientras que en BigO3 alcanza alrededor del 5 % en el tamaño intermedio y algo más del 10 % en el mayor.

Finalmente, cuando los segmentos replicados igualan el ancho de *retire* del *backend* distribuido (4 en SmallO3 y 8 en BigO3), IQ Única + Puertos Replicados sigue siendo competitivo incluso con mayor latencia de *writeback*, con pérdidas medias inferiores al 5 % en SmallO3 y al 10 % en BigO3, incluidas las IQ más grandes.

En conjunto, estos resultados indican que una organización IQ Única + Puertos Replicados puede constituir una alternativa viable y competitiva frente al *backend* distribuido convencional, incluso bajo hipótesis de sobrecoste moderado. Aunque la replicación incrementa el área estructural, también reconfigura el comportamiento de los caminos críticos y los patrones de contención de manera que puede compensar parcialmente dicho coste. Además, *backends* replicados aportan ventajas arquitectónicas adicionales, como mayor regularidad estructural y posible simplificación de la lógica de planificación. En el contexto de microarquitecturas RISC-V orientadas a diseños escalables de alto rendimiento, la replicación del *backend* puede considerarse viable para continuar el escalado de prestaciones.

V. Conclusiones

Este trabajo analiza el impacto del tamaño de la ventana de planificación y de la organización de los recursos de ejecución sobre la escalabilidad del rendimiento en *backends* superescalares fuera de orden de gran anchura. Motivado por el contraste entre planificación unificada y organizaciones distribuidas, cada vez más presentes en núcleos RISC-V de alto rendimiento, exploramos un espacio de diseño que abarca *backends* monolíticos, distribuidos y segmentados con replicación completa.

Los resultados muestran que, aunque las colas de *issue* unificadas pueden ofrecer ventaja de rendimiento si las latencias son equivalentes, este beneficio es muy sensible a incrementos en el retardo del proceso de *wakeup-select*. Incluso penalizaciones moderadas compensan gran parte de las ganancias del mayor alcance en la planificación, lo que evidencia las limitaciones prácticas en diseños fuertemente centralizados. En contraste, los *backends* segmentados con replicación presentan un compromiso más equilibrado entre disponibilidad de recursos y sensibilidad a la latencia. Al reducir la contención en ejecución y mantener caminos críticos más cortos, la replicación puede igualar, o aproximarse estrechamente, al rendimiento de diseños distribuidos de mayor anchura, incluso con penalizaciones moderadas en *writeback*.

En conjunto, los resultados sugieren que la replicación del *backend* es una estrategia de escalado viable, especialmente donde las restricciones temporales y físicas dificultan implementar planificadores centralizados de gran tamaño.

Referencias

[1] A. Waterman, Y. Lee, D. Patterson, K. Asanovic, V. I. U. level Isa, A. Waterman, Y. Lee, and D. Patterson, "The risc-v instruction set manual," *Volume I: User-Level ISA', version*, vol. 2, 2014.

[2] Y. Lee, A. Waterman, H. Cook, B. Zimmer, B. Keller, A. Puggelli, J. Kwak, R. Jevtic, S. Bailey, M. Blagojevic, P. F. Chiu, R. Avizienis, B. Richards, J. Bachrach, D. Patterson, E. Alon, B. Nikolic, and K. Asanovic, "An agile approach to building risc-v microprocessors," *IEEE Micro*, vol. 36, 2016.

[3] Y. Xu, Z. Yu, D. Tang, G. Chen, L. Chen, L. Gou, Y. Jin, Q. Li, X. Li, Z. Li, J. Lin, T. Liu, Z. Liu, J. Tan, H. Wang, H. Wang, K. Wang, C. Zhang, F. Zhang, L. Zhang, Z. Zhang, Y. Zhao, Y. Zhou, Y. Zhou, J. Zou, Y. Cai, D. Huan, Z. Li, J. Zhao, Z. Chen, W. He, Q. Quan, X. Liu, S. Wang, K. Shi, N. Sun, and Y. Bao, "Towards developing high performance risc-v processors using agile methodology," in *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, vol. 2022-October, 2022.

[4] "P870 high-performance risc-v processor," in *2023 IEEE Hot Chips 35 Symposium, HCS 2023*, 2023.

[5] C. Chen, X. Xiang, C. Liu, Y. Shang, R. Guo, D. Liu, Y. Lu, Z. Hao, J. Luo, Z. Chen, C. Li, Y. Pu, J. Meng, X. Yan, Y. Xie, and X. Qi, "Xuantie-910: A commercial multi-core 12-stage pipeline out-of-order 64-bit high performance risc-v processor with vector extension : Industrial product," in *Proceedings - International Symposium on Computer Architecture*, vol. 2020-May, 2020.

[6] A. Seznec, "Tage-sc-l branch predictors again," *JWAC-4: Championship Branch Prediction*, 2016.

[7] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Complexity-effective superscalar processors," in *Conference Proceedings - Annual International Symposium on Computer Architecture, ISCA*, 1997.

[8] H. Wong, V. Betz, and J. Rose, "High performance instruction scheduling circuits for out-of-order soft processors," in *Proceedings - 24th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2016*, 2016.

[9] I. Jeong, J. Lee, M. K. Yoon, and W. W. Ro, "Reconstructing out-of-order issue queue," in *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, vol. 2022-October, 2022.

[10] K. Mori, S. Kosugi, H. Yoshida, H. Shimada, and H. Ando, "Localizing the tag comparisons in the wakeup logic to reduce energy consumption of the issue queue," in *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, 2024.

[11] D. Suggs, M. Subramony, and D. Bouvier, "The amd 'zen 2' processor," *IEEE Micro*, vol. 40, 2020.

[12] T. Singh, S. Oliver, S. Rangarajan, S. Southard, C. Henrion, A. Schaefer, B. Johnson, S. B. Tower, K. Hoover, D. John, T. Antoniadis, S. Lakshman, V. Mittal, B. Kasprzyk, R. McCoy, K. Mohlman, A. Mohan, H. H. Wong, D. Lieu, R. Schreiber, S. Singh, N. Lance, D. Prudich, J. Coppin, T. Jackson, A. Karegar, R. Miller, S. Balagangadharan, J. Pistole, W. Li, and M. McCabe, "'zen 5': The amd high-performance 4nm x86-64 microprocessor core," in *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, 2025.

[13] W. Gomes, S. Morgan, B. Phelps, T. Wilson, and E. Hallnor, "Meteor lake and arrow lake intel next-gen 3d client architecture platform with foveros," in *2022 IEEE Hot Chips 34 Symposium, HCS 2022*, 2022.

[14] N. Bonen, A. Gihon, L. Polishuk, Y. Aizik, Y. Okunev, T. Kurts, and N. Bashyam, "Lunar lake an intel mobile processor: Soc architecture overview (2024)," *IEEE Micro*, vol. 45, 2025.

[15] T. Garibay and S. Nemawarkar, "Cuzco: A high-performance risc-v rva23 compatible cpu ip," 2025.

[16] J. Abella, R. Canal, and A. González, "Power- and complexity-aware issue queue designs," *IEEE Micro*, vol. 23, 2003.

[17] D. Folegnani and A. González, "Energy-effective issue logic," in *Conference Proceedings - Annual International Symposium on Computer Architecture, ISCA*, 2001.

[18] S. Dropsho, A. Buyuktosunoglu, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, G. Semeraro, G. Magklis, and M. L. Scottt, "Integrating adaptive on-chip storage structures for reduced dynamic power," in *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, vol. 2002-January, 2002.

[19] T. Moreshet and R. I. Bahar, "Effects of speculation on performance and issue queue design," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, 2004.

[20] M. Goshima, K. Nishino, Y. Nakashima, S. I. Mori, T. Kitamura, and S. Tomita, "A high-speed dynamic instruction scheduling scheme for superscalar processors," in *Proceedings of the Annual International Symposium on Microarchitecture*, 2001.

[21] M. D. Brown, J. Stark, and Y. N. Patt, "Select-free instruction scheduling logic," in *Proceedings of the Annual International Symposium on Microarchitecture*, 2001.

[22] S. Sakai, T. Suenaga, R. Shioya, and H. Ando, "Rearranging random issue queue with high ipc and short delay," in *Proceedings - 2018 IEEE 36th International Conference on Computer Design, ICCD 2018*, 2019.

[23] H. Ando, "Performance improvement by prioritizing the issue of the instructions in unconfident branch slices," in *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, vol. 2018-October, 2018.

[24] ——, "Segmenting age matrices to improve instruction scheduling without increasing delay and area," in *Proceedings - IEEE International Conference on Computer Design: VLSI in Computers and Processors*, vol. 2022-October, 2022.

[25] P. Prieto, P. Abad, J. A. Gregorio, and V. Puente, "Performance characterization of popular dnn models on out-of-order cpus," in *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, 2023.

[26] R. Canal, J. M. Parcerisa, and A. Gonzalez, "Cost-effective clustered architecture," *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, 1999.

[27] R. Canal, J. M. Parcerisa, and A. González, "Dynamic cluster assignment mechanisms," in *IEEE High-Performance Computer Architecture Symposium Proceedings*, 2000.

[28] C. S. Deshpande, A. Perais, and F. Pétrot, "Address/data instruction steering in clustered general purpose processors," *ACM Transactions on Architecture and Code Optimization*, vol. 22, 2025.

[29] J. Lowe-Power, A. M. Ahmad, A. Akram *et al.*, "The gem5 simulator: Version 20.0+," *CoRR*, vol. abs/2007.03152, 2020. [Online]. Available: https://arxiv.org/abs/2007.03152

[30] N. L. Binkert, B. M. Beckmann, G. Black *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011. [Online]. Available: https://doi.org/10.1145/2024716.2024718

[31] "Spec cpu 2017," 2017. [Online]. Available: https://www.spec.org/

[32] E. J. Gomez-Hernandez, J. M. Cebrian, S. Kaxiras, and A. Ros, "Splash-4: A modern benchmark suite with lock-free constructs," in *Proceedings - 2022 IEEE International Symposium on Workload Characterization, IISWC 2022*, 2022.

[33] H. Jin, M. Frumkin, and J. Yan, "The openmp implementation of nas parallel benchmarks and its performance," *National Aeronautics and Space Administration (NASA), Technical Report NAS-99-011, Moffett Field, USA*, 1999. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.1321&rep=rep1&type=pdf

[34] SiFive, "Sifive hifive premier p550 development board (2025)."

[35] M. Bruce, "Arm neoverse v2 platform: Leadership performance and power efficiency for next-generation cloud computing, ml and hpc workloads," in *2023 IEEE Hot Chips 35 Symposium, HCS 2023*, 2023.

[36] W. J. Starke, B. W. Thompto, J. A. Stuecheli, and J. E. Moreira, "Ibm's power10 processor," *IEEE Micro*, vol. 41, 2021.

[37] S. K. Sadasivam, B. W. Thompto, R. Kalla, and W. J. Starke, "Ibm power9 processor architecture," *IEEE Micro*, vol. 37, 2017.

Coste computacional de una red de aprendizaje profundo basada en EfficientNet sobre dispositivos basados en GPU para computación en el borde

Nicolás Vilela-Pérez[1,2], Dora B. Heras[1,2] y Francisco Argüello[2]

Resumen— **El despliegue de redes de aprendizaje profundo para la teledetección en entornos de computación en el borde sigue siendo un gran reto, ya que estas plataformas operan bajo restricciones de energía y de recursos de cómputo. Este desafío exige aplicar técnicas de optimización de redes y realizar un trabajo especialmente cuidadoso en el caso de redes diseñadas para operar con un coste computacional reducido. El problema se agrava en el caso de redes de tamaño medio, ya que la mayoría de las técnicas de optimización están orientadas a redes de gran escala. En este trabajo se analizan el coste computacional y la eficiencia energética generados por técnicas de optimización aplicadas a la red EffBaGAN, una red de tamaño medio basada en EfficientNet, orientada a la clasificación de imágenes multiespectrales de alta resolución espacial para la cobertura terrestre en escenarios de escasez y desequilibrio de datos. Se evalúa el impacto de la precisión mixta automática (AMP), la cuantización durante y después del entrenamiento (QAT y PTQ, respectivamente) y la poda, tanto de forma aislada como combinada, en una plataforma NVIDIA Jetson AGX Orin. Además, se analiza el rendimiento computacional en función de los límites de potencia del dispositivo. Los resultados revelan que la combinación del entrenamiento sensible a la cuantización (QAT) con la poda representa el enfoque más efectivo para lograr reducciones conjuntas en el tamaño de la red y en el número de parámetros y FLOPs, al mismo tiempo que se mantienen métricas de precisión competitivas.**

Palabras clave— **clasificación de imágenes, teledetección, computación en el borde, coste computacional, eficiencia energética, precisión mixta automática, cuantización, poda**

I. Introducción

En el ámbito de la teledetección, uno de los problemas fundamentales es la clasificación de imágenes, una tarea que consiste en asignar una etiqueta o clase a cada píxel para identificar diferentes tipos de elementos en problemas de cobertura terrestre como, por ejemplo, ecosistemas vegetales [1]. En la actualidad, las redes de aprendizaje profundo han desplazado a los algoritmos clásicos de aprendizaje automático para la clasificación, superando sus métricas de precisión. No obstante, la mejora algorítmica conlleva una penalización: una alta exigencia de recursos computacionales y la necesidad de grandes cantidades de datos etiquetados para su entrenamiento adecuado [2]. Por otro lado, en aplicaciones reales, como el etiquetado de componentes vegetales y artificiales en imágenes de cobertura terrestre de teledetección, la obtención de datos etiquetados resulta costosa y compleja, ya que se trata de zonas de difícil acceso y, dependiendo de la resolución espacial, las imágenes pueden representar elementos de más de una clase en un mismo píxel [3]. Esto da lugar a escenarios de escasez de datos y desequilibrio de clases, donde las redes neuronales tradicionales sufren de sobreajuste, ya que aprenden a representar los patrones y características espaciales y espectrales presentes en los datos de entrenamiento en lugar de patrones generales [4]. La red EffBaGAN (Efficient Balancing Generative Adversarial Network) [5] aborda estos dos problemas al integrar una arquitectura generativa para sintetizar muestras artificiales y estabilizar el aprendizaje. Además, esta red ha sido concebida bajo la premisa de reducir el coste computacional, heredando la alta eficiencia de su arquitectura base EfficientNet [6].

A pesar de estas optimizaciones a nivel algorítmico, el despliegue de redes para teledetección en entornos de computación en el borde para su uso en tiempo cercano al real sigue siendo un reto [7]. Las plataformas computacionales heterogéneas basadas en GPU para la computación en el borde son muy comunes actualmente para ejecutar algoritmos de aprendizaje profundo en teledetección [8]. No obstante, estas operan bajo estrictas restricciones de energía, de memoria y de cómputo. Para facilitar la ejecución de redes neuronales complejas en este tipo de hardware, resulta indispensable aplicar técnicas de optimización del coste computacional [9]. Entre estas estrategias, se encuentran en la literatura las siguientes: la precisión mixta automática [10], que combina operaciones en punto flotante de 32 bits (FP32) con operaciones en 16 bits (FP16) para aliviar el uso del ancho de banda de memoria; la cuantización, que proyecta los pesos y activaciones en representaciones enteras de menor precisión (INT8), pudiendo aplicarse simulando el ruido de cuantización durante el entrenamiento [11], o tras finalizarlo [12]; y la poda [13], que transforma la red eliminando conexiones no representativas o grupos enteros de parámetros, como canales o filtros, reduciendo así el número de operaciones necesarias.

[1]Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, 15782, Santiago de Compostela, España. E-mails: {nicolas.vilela.perez, dora.blanco}@usc.gal.

[2]Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, 15782, Santiago de Compostela, España. E-mail: francisco.arguello@usc.gal.

También existen otras técnicas de mejora enfocadas exclusivamente en el entrenamiento, como LoRA (*Low-Rank Adaptation*) [14], orientadas a la adaptación eficiente de redes preentrenadas de gran escala, lo que reduce drásticamente los requerimientos de memoria y tiempo en esta fase. Sin embargo, dado que estas técnicas no alteran la topología de la red, esta permanece completamente activa durante la inferencia, por lo que no ofrecen ninguna reducción en los costes de despliegue. Por consiguiente, estos enfoques difieren de la compresión estructural de arquitecturas diseñadas desde cero, que es el enfoque de este trabajo.

Más allá de la compresión estructural y algorítmica, el despliegue efectivo de redes exige herramientas orientadas a maximizar la eficiencia en la etapa de inferencia en el hardware de despliegue. Para ello, es una práctica estándar exportar las redes a un formato de representación intermedia interoperable como ONNX (*Open Neural Network Exchange*), lo que permite desacoplar la arquitectura de la red de la sobrecarga del *framework* de entrenamiento [15]. Posteriormente, se emplean motores de inferencia especializados como NVIDIA TensorRT, que actúan como compiladores para GPUs de computación en el borde [16]. Este motor aplica transformaciones a nivel de grafo computacional y realiza un autoajuste para seleccionar las implementaciones a bajo nivel más eficientes según la arquitectura específica de las tarjetas gráficas de destino. Estudios recientes demuestran empíricamente que el uso de estas optimizaciones en plataformas de alto rendimiento, como la NVIDIA Jetson AGX Orin, mejora la velocidad de inferencia y la eficiencia energética frente a otras alternativas [17].

En este trabajo se analiza el coste computacional de aplicar diferentes técnicas de mejora aplicadas a la red de aprendizaje profundo de bajo coste computacional EffBaGAN para la clasificación de imágenes multiespectrales de alta resolución espacial en un dispositivo basado en GPU para computación en el borde. Presentamos un análisis del coste computacional de las diferentes versiones de red optimizada en un dispositivo basado en GPU para computación en el borde: una NVIDIA Jetson AGX Orin. Asimismo, se realiza una evaluación de la eficiencia energética al variar el límite máximo de potencia. Las principales contribuciones de este trabajo son las siguientes:

- **Reducción de coste computacional:** se aplican distintas técnicas de reducción de coste computacional, como la precisión mixta automática, la cuantización durante y después del entrenamiento, y la poda.
- **Análisis del impacto de los distintos perfiles de potencia:** se evalúa cómo escala la red optimizada bajo diferentes restricciones de potencia configurables en el dispositivo, alterando la frecuencia y el número de núcleos activos de la CPU/GPU.

II. Red EffBaGAN para clasificación

Este trabajo analiza la ejecución sobre un dispositivo basado en GPU para computación en el borde de la red EffBaGAN (*Efficient Balancing Generative Adversarial Network*) [5], una red generativa específicamente diseñada para resolver tareas de clasificación en condiciones de escasez de datos y desequilibrio de clases de forma eficiente.

Para abordar la falta de datos, EffBaGAN se basa en redes generativas adversarias (GANs). Una GAN consiste en dos redes neuronales: un generador, que aprende a crear muestras sintéticas realistas a partir de ruido (o de un espacio latente), y un discriminador, que intenta distinguir entre las muestras reales y las generadas. Este proceso iterativo permite generar nuevas muestras sintéticas de alta calidad que enriquecen y equilibran el conjunto de entrenamiento, evitando así el sobreajuste del clasificador. En concreto, EffBaGAN se apoya en una variante denominada red generativa adversaria balanceada (BAGAN) [18], diseñada para estabilizar el aprendizaje en clases minoritarias.

A pesar de su capacidad para estabilizar el aprendizaje, las arquitecturas GAN conllevan una alta demanda de recursos computacionales. Por ello, a diferencia de los enfoques convencionales, el diseño de EffBaGAN ha tenido el bajo coste computacional como criterio guía, combinando las ventajas de BAGAN con la eficiencia computacional de la arquitectura EfficientNet [6]. La esencia de este bajo coste computacional radica en el uso de bloques de cuello de botella invertido móviles (MBConv) como unidad principal de procesamiento del discriminador. Estos bloques, a diferencia de las convoluciones tradicionales, que operan simultáneamente en las dimensiones espaciales y de canal, emplean convoluciones separables en profundidad. Esta técnica desacopla la extracción de características espaciales de la combinación de canales, reduciendo drásticamente las interacciones intensivas y minimizando el coste global de la red.

La figura 1 muestra la arquitectura de EffBaGAN. Para el entrenamiento, la red consta de tres pasos:

1. Primero, se entrena el *autoencoder* sin considerar la clase para aprender una representación compacta de los datos que captura su estructura y distribución subyacentes. Un *autoencoder* [19] es una red que a través de un codificador (*encoder*) y de un decodificador (*decoder*) genera vectores de características comprimidas de las muestras de entrada. Concretamente, el codificador reduce la dimensionalidad de las imágenes originales mapeándolas a un espacio latente de menor tamaño, extrayendo patrones más representativos, mientras que el decodificador toma estos vectores comprimidos y reconstruye la imagen original, ajustando sus pesos para minimizar el error de reconstrucción.
2. Después, se transfieren los parámetros del *autoencoder* que permiten reconstruir las muestras

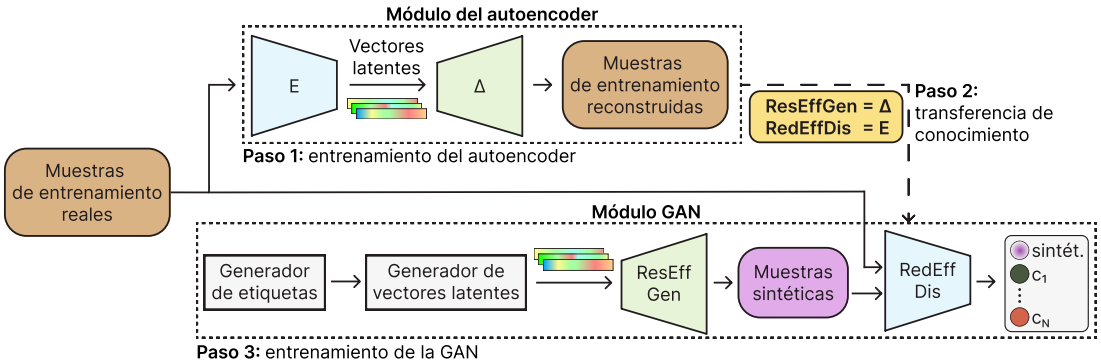


Fig. 1: Arquitectura neuronal de la red EffBaGAN [5].

originales al módulo GAN, adquiriendo este así el conocimiento del *autoencoder* y reduciendo el tiempo de entrenamiento, ya que al partir de información no aleatoria sobre los parámetros, la calidad del entrenamiento del módulo GAN mejora.

3. Por último, se entrena el módulo GAN propiamente dicho. El módulo GAN es el bloque principal de la arquitectura de la red, estando formado por dos componentes: ResEffGen, un generador que debido a la introducción de un camino residual basado en EfficientNet sintetiza muestras de alta calidad; y RedEffDis, un discriminador reducido basado en la versión EfficientNet-B0.

El coste computacional de EffBaGAN es, de partida, bajo, debido a su propia arquitectura. De las dos versiones existentes de EffBaGAN, se elige para este trabajo la de menor coste computacional: EffBaGAN-Small. A pesar de su bajo coste computacional, que no supera los 5 millones de parámetros, los recursos limitados disponibles en plataformas como la NVIDIA Jetson estudiada en este trabajo siguen haciendo necesaria la aplicación de técnicas de mejora del rendimiento.

III. Técnicas de reducción de coste computacional

Aunque la arquitectura de EffBaGAN ya es eficiente, como se explicó en la sección anterior, las restricciones de los dispositivos de computación en el borde requieren optimizaciones adicionales a nivel de software. En este trabajo se evalúa el impacto individual y combinado de las siguientes técnicas para la reducción del coste computacional de la red:

- **Precisión mixta automática** (*Automatic Mixed Precision*, AMP) [10]: esta técnica optimiza el uso de memoria durante el entrenamiento combinando dinámicamente operaciones en precisión simple (FP32), necesarias para garantizar la estabilidad numérica, con operaciones en precisión media (FP16), orientadas a reducir el uso del ancho de banda de la memoria.
- **Entrenamiento sensible a la cuantización** (*Quantization-Aware Training*, QAT) [11]: consiste en reducir la precisión de los pesos y las activaciones de la red a enteros de 8 bits (INT8) simulando los efectos de la cuantización mediante la inyección de ruido durante el entrenamiento. Esto permite que la red se adapte gradualmente a representaciones de menor precisión, logrando una mayor robustez, aunque a costa de aumentar el tiempo de entrenamiento y el uso de memoria en la GPU.
- **Cuantización posentrenamiento** (*Post Training Quantization*, PTQ) [12]: al igual que QAT, proyecta los pesos y las activaciones de la red a una menor precisión (INT8), pero se aplica una vez finalizado el entrenamiento. Su objetivo principal es comprimir la red, acelerar el tiempo de inferencia y reducir el consumo de energía en dispositivos de borde. Es un proceso rápido que requiere un paso de calibración con datos representativos, pero no sobrecarga el tiempo de entrenamiento ni modifica los pesos aprendidos.
- **Poda** [13]: transforma la red densa en una arquitectura más compacta mediante la eliminación permanente de conexiones consideradas redundantes. En este caso se aplica una poda estructurada de filtros en las capas convolucionales, utilizando un criterio basado en la magnitud para eliminar el 25 % de los canales menos significativos, tras lo cual se aplica una etapa de *fine-tuning* para tratar de conseguir la máxima precisión posible en la clasificación en comparación con la red original. Su objetivo principal es reducir el número de parámetros y de operaciones (FLOPs) de la red. Cabe comentar que este tipo de poda se realiza en anchura, al eliminar filtros y no capas completas, por lo que la red resultante tendrá el mismo número de capas y, en estas, menos filtros. Dado que se mantienen las dependencias entre capas y el número de capas, no se esperan mejoras en tiempo de ejecución notables respecto a una poda en profundidad [20].

La siguiente sección evalúa el impacto de aplicar estas técnicas, analizando su viabilidad y rendimiento en el dispositivo de computación en el borde, tanto de forma aislada como combinada.

IV. Experimentación

En esta sección presentaremos inicialmente el contexto experimental y el conjunto de imágenes multi-

espectrales de cobertura terrestre sobre el que se han realizado los experimentos. A continuación mostraremos los resultados de aplicar diferentes técnicas de optimización a la red EffBaGAN así como los resultados bajo diferentes restricciones de potencia.

A. Configuración experimental

Para evaluar el comportamiento de EffBaGAN sobre un dispositivo basado en GPU para computación en el borde se han usado tres conjuntos de datos multiespectrales de alta resolución espacial (10 cm/px) de cuencas de ríos en Galicia, usados anteriormente en [21]. En la tabla I se detallan las 10 clases presentes en la información de referencia de estos tres conjuntos de datos. Las imágenes que componen los conjuntos de datos han sido capturadas con un vehículo aéreo no tripulado a 120 m de altitud equipado con una cámara multiespectral MicaSense RedEdge-MX, que captura 5 bandas correspondientes a las longitudes de onda de 475 nm (azul), 560 nm (verde), 668 nm (rojo), 717 nm (*red-edge*) y 840 nm (infrarrojo cercano).

Tabla I: Número de muestras para cada conjunto de datos expresado en número de superpíxeles (obtenidos mediante el algoritmo de segmentación Waterpixels).

#	Color	Clases	Eiras Dam	Oitavén River	Xesta Basin
1		Agua	1 929	912	210
2		Tierra	424	484	182
3		Piedras	726	349	2 170
4		Asfalto	233	128	467
5		Hormigón	111	458	0
6		Tejados	29	294	0
7		Prados	2 989	6 896	42 821
8		Árboles nativos	8 604	7 573	15 543
9		Pinos	351	955	0
10		Eucaliptos	39	3 599	0
		Muestras totales:	15 435	21 648	61 393

En la figura 2 se puede ver la imagen en color RGB del conjunto de datos Eiras Dam, así como el detalle de una porción de dicha imagen, incluyendo la información de referencia correspondiente, que indica en diferentes colores los elementos presentes. Las zonas negras corresponden a regiones en las que la información de referencia no es concluyente.

Todos los conjuntos de datos han sido segmentados en superpíxeles usando el algoritmo Waterpixels con un tamaño medio de 400 px/superpíxel, un tamaño mínimo de 100 px/superpíxel y un factor de compacidad de 0,5 puntos, tal y como se describe en [21]. La entrada de la red son los parches centrados, cada uno en un superpíxel, con una dimensión espacial de 32×32 px. Adicionalmente, todos los datos han sido normalizados al rango $[-1, 1]$. Para cada conjunto de datos, el 15 % de las muestras corresponden al conjunto de entrenamiento, y el 5 %, al conjunto de validación, siendo el 80 % restante para el conjunto de test.

Para analizar el efecto de las técnicas evaluaremos: precisión de clasificación, coste computacional y eficiencia energética. La precisión se evalúa mediante tres métricas estándar a nivel de píxel: la precisión global (OA), la precisión media por clase (AA) y el coeficiente kappa de Cohen (κ). Los tiempos de ejecución registrados son el tiempo de entrenamiento por época (TEPE) y el tiempo total de test (inferencia). Otras métricas de coste computacional son el pico de memoria requerida por la GPU durante el entrenamiento, el tamaño de la red en disco, el número de parámetros entrenables y el número de millones de operaciones de coma flotante (MFLOPs), siendo estas tres últimas métricas tomadas durante la fase de inferencia y, en consecuencia, solo usando el discriminador. Finalmente, para evaluar la eficiencia energética, se registra la potencia media y se estima la energía total consumida tanto en la fase de entrenamiento como en la de test. Con el fin de garantizar la significancia estadística de los resultados, cada configuración experimental se ha ejecutado de forma independiente un total de 5 veces, reportando el valor medio y la desviación típica de todas las métricas.

Para la ejecución de los experimentos se ha utilizado un kit de desarrollo NVIDIA Jetson AGX Orin, un sistema en chip representativo de los dispositivos basados en GPU para computación en el borde, diseñado para operar bajo restricciones de tamaño, peso y potencia. En la tabla II se presentan las especificaciones técnicas de dicho sistema.

Tabla II: Especificaciones del kit de desarrollo NVIDIA Jetson AGX Orin.

Componente	Especificación
CPU	ARM Cortex-A78AE v8.2 de 12 núcleos
GPU	NVIDIA de arquitectura Ampere
Memoria RAM	32 GB LPDDR5 (unificada)
Almacenamiento	64 GB eMMC 5.1
Potencia	Configurable entre 15 W y 60 W

En cuanto al software, el sistema opera sobre el *software stack* NVIDIA JetPack 6.0, que incluye CUDA 12.2 y cuDNN 8.9.4. El desarrollo y la experimentación se han realizado en un entorno de Conda con Python 3.10.19, pudiéndose resaltar los siguientes paquetes: PyTorch 2.4.0, para el entrenamiento de la red; TensorRT 8.6.2, para la inferencia de alto rendimiento; y Guild AI 0.9.0, para registrar los experimentos.

B. Resultados experimentales

B.1 Impacto de las técnicas de reducción de coste computacional

En esta fase de la experimentación se evalúa el impacto de las diferentes técnicas de reducción del coste de cómputo sobre la red EffBaGAN-Small ejecutada en la GPU, utilizando la configuración de máxima potencia del dispositivo. Los resultados de precisión de clasificación y coste computacional se pueden ver en las tablas III y IV.

El análisis individual de las técnicas muestra que AMP logra reducir el pico de memoria requerida por la GPU durante el entrenamiento en un 27,96 %, así

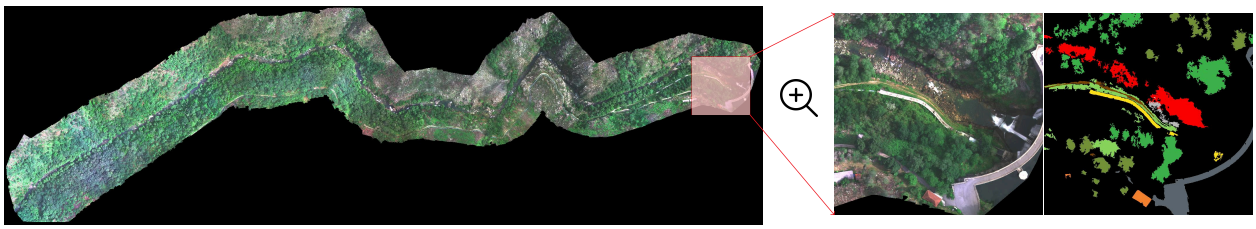


Fig. 2: Imagen en color RGB y ampliación de la misma junto con su información de referencia del conjunto de datos Eiras Dam usado en este trabajo, con una resolución de 18 221 × 5 176 px. La clase correspondiente a cada color de la información de referencia está descrita en la tabla I.

Tabla III: Métricas de precisión y tiempos de ejecución para todos los conjuntos de datos con diferentes técnicas de reducción del coste computacional aplicadas a EffBaGAN-Small en el dispositivo de computación en el borde NVIDIA Jetson AGX Orin.

Técnica	**OA** (%)	**AA** (%)	κ (%)	**TEPE** (s)	**Tiempo de test** (s)
Ninguna	97,32 ± 0,35	90,00 ± 3,10	95,50 ± 0,61	9,47 ± 0,02	265,07 ± 5,00
AMP	97,19 ± 0,36	89,41 ± 2,39	95,31 ± 0,59	9,02 ± 0,03	266,98 ± 4,66
PTQ	96,30 ± 0,70	87,84 ± 2,42	93,83 ± 1,10	9,47 ± 0,02	263,82 ± 6,28
QAT	97,19 ± 0,35	90,75 ± 1,34	95,33 ± 0,57	16,16 ± 0,05	263,43 ± 3,40
Poda	97,63 ± 0,11	91,03 ± 1,73	96,03 ± 0,18	9,48 ± 0,02	263,83 ± 5,18
PTQ + Poda	53,03 ± 23,02	44,58 ± 11,52	36,46 ± 19,72	9,47 ± 0,02	264,25 ± 2,78
QAT + Poda	88,56 ± 11,39	81,06 ± 12,22	84,44 ± 12,68	9,46 ± 0,03	264,45 ± 2,71

Tabla IV: Métricas de coste computacional para todos los conjuntos de datos al aplicar diferentes técnicas a EffBaGAN-Small en el dispositivo de computación en el borde NVIDIA Jetson AGX Orin.

Técnica	**Pico mem. GPU** (MiB)		**Tamaño red** (MiB)		**Parámetros** (#)		**MFLOPs** (#)	
Ninguna	1 314,01		3,78		972 235		27,10	
AMP	946,62	(↓ 27,96 %)	3,78	(=)	972 235	(=)	27,10	(=)
PTQ	1 315,19	(↑ 0,09 %)	1,10	(↓ 70,88 %)	972 235	(=)	26,62	(↓ 1,79 %)
QAT	1 425,14	(↑ 8,46 %)	1,10	(↓ 70,88 %)	972 235	(=)	26,62	(↓ 1,79 %)
Poda	1 315,51	(↑ 0,11 %)	3,22	(↓ 14,99 %)	677 315	(↓ 30,33 %)	18,34	(↓ 32,34 %)
PTQ + Poda	1 314,43	(↑ 0,03 %)	1,34	(↓ 64,60 %)	675 443	(↓ 30,53 %)	17,94	(↓ 33,81 %)
QAT + Poda	1 316,43	(↑ 0,18 %)	1,36	(↓ 63,94 %)	677 315	(↓ 30,33 %)	17,94	(↓ 33,81 %)

Los valores porcentuales indican la diferencia relativa respecto de la configuración sin aplicar ninguna técnica. Todas las métricas de coste computacional registradas, excepto el pico de memoria requerida por la GPU durante el entrenamiento, corresponden a la fase de inferencia; por lo tanto, solo se considera el discriminador al calcularlas.

como disminuir el TEPE de 9,47 s a 9,02 s, siendo un 4,75 % más rápido. Por su parte, ambas técnicas de cuantización (PTQ y QAT) consiguen comprimir el tamaño de la red en disco en más de un 70 %, mientras que la poda logra eliminar más de un 30 % de los parámetros y MFLOPs manteniendo las métricas de precisión estables.

Al evaluar las combinaciones de técnicas de cuantización con poda, resulta evidente la necesidad de un enfoque algorítmico cuidadoso para evitar caídas notables en las métricas de precisión. La combinación de PTQ con poda provoca una caída significativa en las métricas de precisión. Sin embargo, la integración de QAT con poda demuestra ser una estrategia con mayor robustez: consigue aplicar reducciones en el coste computacional de un 63,94 %, 30,33 % y 33,81 % en el tamaño de la red, número de parámetros y número de MFLOPs, respectivamente, a la vez que logra mitigar la caída de precisión en comparación con la combinación con PTQ. Debido a este equilibrio entre precisión y coste, la versión optimizada con QAT y poda se selecciona para el análisis de eficiencia del dispositivo que se realizará a continuación.

B.2 Análisis de perfiles de potencia

Una vez optimizado el coste computacional a nivel de software, en esta sección evaluamos cómo se comporta la red en términos de eficiencia energética. La arquitectura NVIDIA Jetson AGX Orin permite configurar dinámicamente su límite de potencia, ajustando las frecuencias de reloj y los núcleos activos de la CPU y de la GPU.

Para este análisis, se ha ejecutado la versión más eficiente de la red optimizada, la red optimizada con QAT y poda, bajo distintos perfiles de potencia: sin restricciones (MAXN), y con máximos fijados a 50 W, 30 W y 15 W. En la tabla V se muestran los correspondientes tiempos de ejecución y métricas de eficiencia energética de la red optimizada sobre el conjunto de datos Eiras Dam. Debe tenerse en cuenta que las métricas de precisión de la clasificación no se ven afectadas por la modificación de los perfiles de potencia.

Como se puede observar, los perfiles de mayor potencia máxima dan lugar a los tiempos de ejecución más bajos. Por el contrario, los perfiles más restrictivos en términos energéticos penalizan los tiempos de ejecución. Esto demuestra que, para la computación

Tabla V: Tiempos de ejecución y métricas de eficiencia energética, variando el perfil de potencia, para EffBaGAN-Small optimizada con QAT y poda (QAT + Poda) sobre el conjunto de datos Eiras Dam, en el dispositivo de computación en el borde NVIDIA Jetson AGX Orin.

Perfil de potencia	**TEPE** (s)		**Tiempo de test** (s)		**Potencia** (W) Entrenamiento		**Potencia** (W) Test		**Energía** (Wh) Entrenamiento		**Energía** (Wh) Test	
MAXN	4,73		228,18		36,25		15,54		28,59		0,98	
50 W	6,60	(↑ 39,51 %)	300,78	(↑ 31,82 %)	23,83	(↓ 34,27 %)	12,74	(↓ 18,04 %)	26,22	(↓ 8,30 %)	1,07	(↑ 8,33 %)
30 W	11,42	(↑ 141,40 %)	271,62	(↑ 19,04 %)	18,30	(↓ 49,52 %)	13,28	(↓ 14,53 %)	34,84	(↑ 21,86 %)	1,00	(↑ 1,83 %)
15 W	21,54	(↑ 355,26 %)	407,94	(↑ 78,78 %)	12,47	(↓ 65,60 %)	10,34	(↓ 33,43 %)	44,77	(↑ 56,57 %)	1,17	(↑ 19,11 %)

Los valores porcentuales indican la diferencia relativa respecto del perfil de potencia sin restricciones (MAXN).

en el borde, limitar la potencia del hardware puede ser una estrategia eficiente, siempre y cuando se pueda permitir un aumento de los tiempos de ejecución. Para el caso estudiado, si contemplamos un escenario en el que tanto el entrenamiento como el test se ejecutan en el dispositivo en el borde, el perfil de 50 W representa el único compromiso energético viable. Esto se debe a que la energía consumida en el entrenamiento se reduce en un 8,30 %, tal como se puede observar en la tabla V, siendo esta 26,22 Wh frente a 28,59 Wh para el caso MAXN, es decir, sin restricciones. Los perfiles de potencia con restricciones más agresivas (30 W y 15 W) provocan un incremento notable en los tiempos, lo que tiene un impacto directo en la energía total consumida.

Puede resultar llamativo que el perfil de potencia limitado a 50 W tenga un mayor tiempo de test que el limitado a 30 W. Esto es debido a las restricciones de frecuencia de cada perfil: para mantener el límite de consumo, el perfil de 50 W habilita los 12 núcleos de la CPU, pero restringe su frecuencia máxima a 1,50 GHz; por el contrario, el perfil de 30 W trabaja solamente con 8 núcleos, pero a una mayor frecuencia de 1,72 GHz. Dado que la etapa de posprocesado asociada al cálculo de métricas de precisión se ejecuta en la CPU de forma secuencial, esta parte del código se ejecuta más rápido en el núcleo de mayor frecuencia disponible en el escenario del perfil de 30 W.

V. Conclusiones

En este trabajo se ha analizado la aplicación de distintas técnicas para reducir el coste computacional de la red EffBaGAN-Small, una red generativa eficiente orientada a la clasificación de imágenes de teledetección en escenarios de escasez de datos y desequilibrio entre clases. El despliegue de esta red de tamaño medio en una plataforma de computación en el borde NVIDIA Jetson AGX Orin ha permitido cuantificar el impacto real de estas optimizaciones en un hardware con restricciones computacionales.

Los resultados experimentales muestran que, aplicadas de forma aislada, las técnicas evaluadas ofrecen beneficios notables: la precisión mixta automática (AMP) reduce el pico de memoria requerida por la GPU durante el entrenamiento en un 27,96 %, las estrategias de cuantización comprimen el tamaño de la red en más de un 70 %, y la poda logra reducir en más del 30 % el número de parámetros y MFLOPs manteniendo las métricas de precisión de la clasificación. No obstante, el análisis de la interacción entre técnicas revela que la pertinencia de combinar técnicas de optimización requiere un análisis cuidadoso. Por ejemplo, la integración de la técnica de cuantización PTQ con una poda posterior provoca una caída notable en el rendimiento de la clasificación. La combinación de QAT con poda es la estrategia más robusta, lo que reduce el coste de computación sin comprometer significativamente la precisión de la clasificación, que se reduce en torno a 10 puntos porcentuales. La evaluación del comportamiento en escenarios de limitación de potencia indica que restringir el límite penaliza el tiempo de ejecución, pero puede asegurar la viabilidad de la ejecución en escenarios de disponibilidad limitada de energía, como suele ser el caso de dispositivos de computación en el borde.

Resulta pertinente analizar hasta qué punto los resultados y comportamientos expuestos en este trabajo son extrapolables a otras redes. Una de las observaciones es que, si bien técnicas como la poda logran reducir más de un 30 % los parámetros y los MFLOPs para la red, estas mejoras apenas se traducen en una reducción tangible en tiempo de ejecución. Este comportamiento se debe al tamaño de la red estudiada, ya que EffBaGAN-Small es una arquitectura compacta que cuenta con menos de un millón de parámetros en el discriminador. Para la ejecución de redes de tamaño similar, que puede considerarse de tamaño medio en el entorno de aprendizaje profundo, el tiempo de procesamiento en la GPU queda enmascarado por las sobrecargas del entorno de ejecución, ya que estamos utilizando *frameworks* de alto nivel como PyTorch. Como conclusión, podemos decir que para redes con menos de un millón de parámetros, la computación deja de ser el cuello de botella, dominada por estos costes fijos, que incluyen los movimientos de datos transparentes para el usuario, y resulta difícil percibir consecuencias directas en el tiempo de ejecución.

La decisión de aplicar las técnicas analizadas a otra red debe basarse en el tamaño y no solo en el tipo de red. En redes pequeñas o altamente optimizadas desde su diseño mediante bloques eficientes, como es el caso, la aplicación de técnicas de optimización no producirá mejoras notables en los tiempos de ejecución. Por el contrario, en el caso de redes de aprendizaje profundo con un alto número de parámetros (decenas de millones), las optimizaciones se traducirán en una reducción del tiempo de ejecución.

Agradecimientos

Este trabajo ha sido financiado por el proyecto PID2022-141623NB-I00, financiado por MCIN/AEI/10.13039/501100011033, y por la Xunta de Galicia – Consellería de Educación, Ciencia, Universidades e Formación profesional (acreditación de Centro de Investigación de Galicia 2024-2027 ED431G-2023/04 y acreditación de Grupo Competitivo de Referencia ED431C-2022/16), ambos cofinanciados por el FEDER/UE. El trabajo de Nicolás Vilela-Pérez ha sido financiado por la Xunta de Galicia – Consellería de Educación, Ciencia, Universidades e Formación profesional, a través de la ayuda ED481A-2025-015.

Referencias

[1] Enpu Yu, "Review of remote sensing image classification: Technology evolution, method innovation and future challenges," *Applied and Computational Engineering*, vol. 177, pp. 110–116, 07 2025.

[2] Neil Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F. Manso, "The Computational Limits of Deep Learning," in *Ninth Computing within Limits 2023*. LIMITS, jun 14 2023, https://limits.pubpub.org/pub/wm1lwjce.

[3] José M. Bioucas-Dias, Antonio Plaza, Nicolas Dobigeon, Mario Parente, Qian Du, Paul Gader, and Jocelyn Chanussot, "Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 2, pp. 354–379, 2012.

[4] Shweta Sharma and Anjana Gosain, "Addressing class imbalance in remote sensing using deep learning approaches: a systematic literature review," *Evolutionary Intelligence*, vol. 18, no. 1, pp. 23, Jan 2025.

[5] Nicolás Vilela-Pérez, Dora B. Heras, and Francisco Argüello, "EffBaGAN: An efficient balancing GAN for Earth observation in data scarcity scenarios," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 18, pp. 2477–2496, 2025.

[6] Mingxing Tan and Quoc Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, Kamalika Chaudhuri and Ruslan Salakhutdinov, Eds. 09–15 Jun 2019, vol. 97 of *Proceedings of Machine Learning Research*, pp. 6105–6114, PMLR.

[7] Tianyu Wang, Jinyang Guo, Bowen Zhang, Ge Yang, and Dong Li, "Deploying AI on edge: Advancement and challenges in edge intelligence," *Mathematics*, vol. 13, no. 11, 2025.

[8] Javier López-Fandiño, Dora B. Heras, and Francisco Argüello, "Using heterogeneous computing and edge computing to accelerate anomaly detection in remotely sensed multispectral images," *The Journal of Supercomputing*, vol. 80, no. 9, pp. 12543–12563, jun 2024.

[9] Sebastián A. Cajas Ordóñez, Jaydeep Samanta, Andrés L. Suárez-Cetrulo, and Ricardo Simón Carbajo, "Intelligent edge computing and machine learning: A survey of optimization and applications," *Future Internet*, vol. 17, no. 9, 2025.

[10] Sharan Narang, Gregory Diamos, Erich Elsen, Paulius Micikevicius, Jonah Alben, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al., "Mixed precision training," in *Int. Conf. on Learning Representation*, 2017.

[11] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[12] Raghuraman Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *CoRR*, vol. abs/1806.08342, 2018.

[13] Song Han, Jeff Pool, John Tran, and William Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[14] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen, "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022.

[15] Lukas Stäcker, Juncong Fei, Philipp Heidenreich, Frank Bonarens, Jason Rambach, Didier Stricker, and Christoph Stiller, "Deployment of deep neural networks for object detection on edge AI devices with runtime optimization," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, October 2021, pp. 1015–1022.

[16] Omais Shafi, Chinmay Rai, Rijurekha Sen, and Gayathri Ananthanarayanan, "Demystifying TensorRT: Characterizing neural network inference engine on Nvidia edge devices," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, 2021, pp. 226–237.

[17] Ishrak Jahan Ratul, Yuxiao Zhou, and Kecheng Yang, "Accelerating deep learning inference: A comparative analysis of modern acceleration frameworks," *Electronics*, vol. 14, no. 15, 2025.

[18] Giovanni Mariani, Florian Scheidegger, Roxana Istrate, Costas Bekas, and Cristiano Malossi, "BAGAN: Data augmentation with balancing GAN," in *International Conference on Machine Learning*, 2018.

[19] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[20] Sara Elkerdawy, Mostafa Elhoushi, Abhineet Singh, Hong Zhang, and Nilanjan Ray, "To filter prune, or to layer prune, that is the question," in *Proceedings of the Asian Conference on Computer Vision (ACCV)*, November 2020.

[21] Francisco Argüello, Dora B. Heras, Alberto S. Garea, and Pablo Quesada-Barriuso, "Watershed monitoring in Galicia from UAV multispectral imagery using advanced texture methods," *Remote Sensing*, vol. 13, no. 14, 2021.

Infraestructura para el modelado de políticas de planificación multihilo SMT en el simulador gem5

Manuel A. Pecino[1], Daniel Escribano[1], Salvador Petit[1] y Julio Sahuquillo[1]

Resumen—**El auge de las aplicaciones de inteligencia artificial ha modificado notablemente la composición de las cargas de trabajo empleadas en entornos HPC y Cloud, aumentando significativamente el tiempo de cómputo dedicado a este tipo de aplicaciones. Esto plantea nuevos retos y desafíos a la hora de gestionar los recursos de *hardware*, como es el caso de la ejecución simultánea de múltiples hilos. Dicha tecnología se ha consolidado como la mejor alternativa para mejorar las prestaciones del sistema con un relativamente bajo coste en área de silicio. No obstante, las aplicaciones ejecutadas simultáneamente ven su rendimiento degradado por la competición por recursos compartidos.**

Simuladores detallados ciclo a ciclo, como gem5, suponen una herramienta esencial en la investigación en arquitectura de computadores. No obstante, los simuladores de este tipo carecen del soporte necesario para el diseño y evaluación de políticas de planificación. En este trabajo se presenta una infraestructura *software* que extiende gem5 para habilitar la planificación de hilos en tiempo de ejecución, permitiendo el análisis de políticas de planificación en este contexto.

Como caso de estudio, en este trabajo se implementa una política FIFO y se analiza su comportamiento utilizando cargas compuestas por aplicaciones de IA generadas específicamente para su ejecución eficiente en entornos de simulación ciclo a ciclo. Los resultados demuestran patrones coherentes con lo esperable en un sistema real, habilitando el estudio riguroso de este tipo de políticas en arquitecturas SMT simuladas.

Palabras clave—**gem5, planificación, SMT, multihilo, simulación.**

I. Introducción

La computación científica y de altas prestaciones (HPC) ha sido tradicionalmente objeto de investigación para la mejora de las prestaciones y de la eficiencia energética. En dichos entornos de cómputo, es crucial maximizar el rendimiento y la utilización de los recursos [1]. Con este fin, los procesadores de altas prestaciones modernos implementan técnicas como el *simultaneous multi-threading* (SMT). Esta técnica permite procesar instrucciones de varios hilos en un mismo ciclo. Así se logra, con un relativo pequeño aumento del área del chip, un gran aumento en las prestaciones del sistema y la utilización del procesador [2].

No obstante, los hilos en un procesador SMT comparten recursos durante su ejecución, lo cual conlleva una degradación de las prestaciones de las aplicaciones con respecto a su ejecución en solitario. La interferencia o contención por los recursos compartidos puede ocurrir tanto dentro del núcleo (por ejemplo, en los operadores de cómputo) como fuera del núcleo (por ejemplo, en los distintos niveles de la jerarquía de memoria). La interferencia y su impacto en las prestaciones depende de los patrones de consumo de recursos de las aplicaciones que se ejecutan simultáneamente en el procesador. De esta forma, aplicaciones con patrones de consumo similares presentarán mayor interferencia entre ellas que en el caso de ser emparejadas con aplicaciones con patrones de consumo complementarios. Cuando dos aplicaciones tienen patrones de consumo complementarios, se dice que presentan un comportamiento simbiótico.

Para maximizar las prestaciones de cargas de trabajo compuestas por un conjunto de aplicaciones, la ejecución simbiótica constituye una solución sólida. Para lograr dicha simbiosis, es clave identificar los patrones de consumo de recursos de las aplicaciones y, mediante la planificación de hilos, asignar hilos con patrones simbióticos al mismo núcleo físico SMT.

En este contexto, la planificación y la asignación de hilos a núcleos en procesadores SMT son actualmente un tema de investigación de interés tanto para la industria como para la academia. La planificación de hilos permite explotar comportamientos simbióticos para reducir al máximo la penalización sufrida por la contención de recursos compartidos.

En los sistemas actuales, compuestos por procesadores multinúcleo con soporte a dos hilos por núcleo (SMT2), la planificación de hilos abarca dos funciones fundamentales. Por un lado, determinar el intervalo de tiempo (*quantum*) donde se ejecutará cada aplicación y por otro, seleccionar las parejas de aplicaciones que se ejecutarán simultáneamente en cada núcleo, esto es, la ubicación de hilos en núcleos (*thread-to-core allocation*) [3] [4].

A la hora de desarrollar la investigación en arquitectura de computadores, la simulación constituye la metodología de diseño más empleada en el estado del arte para formular nuevas propuestas. La simulación ha demostrado ser una herramienta esencial, permitiendo explorar diseños sin sufrir los costes de implementación de caros prototipos físicos. Actualmente, uno de los simuladores más utilizados tanto por la academia como por la industria es gem5 [5]. Se trata de un simulador modular de código abierto, desarrollado en C++ y Python, que cuenta con una gran comunidad de desarrolladores. Sin embargo, para el objetivo de nuestra investigación, gem5 presenta limitaciones en lo que respecta al diseño de políticas de planificación en procesadores SMT.

[1]Departamento de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mail: {mapecfer,descrom,spetit,jsahuqui}@disca.upv.es

En gem5, el soporte a SMT sólo puede ser modelado en modo *syscall emulation*, pero este modo, al no simular un sistema operativo real, no permite el modelado de un sistema realista donde las aplicaciones llegan dinámicamente al sistema y son ubicadas en los diversos núcleos de acuerdo a una determinada estrategia o política de planificación. Este trabajo resuelve esta limitación y presenta una novedosa infraestructura de *software* integrada en gem5 capaz de gestionar e inyectar trabajos dinámicamente para la evaluación de políticas de planificación.

Además de la contribución a gem5, este trabajo también se enfoca en la evaluación del comportamiento simbiótico de cargas de Inteligencia Artificial (IA) para procesadores SMT. Dicho enfoque está motivado por el auge de este tipo de cargas, que ha transformado la realidad computacional del sector y, por tanto, la investigación. Por lo general, los modelos de aprendizaje profundo (*Deep Learning*), debido a sus características y costes computacionales, se ejecutan en aceleradores gráficos como las GPUs. No obstante, un pilar fundamental de la IA son las cargas de Machine Learning (ML) clásico, como pueden ser los árboles de decisión o los algoritmos de clusterizado. Al contrario que los grandes modelos de lenguaje (LLM), este tipo de cargas se ejecutan predominantemente en la CPU, representando una fracción masiva de los ciclos de cómputo diarios en los centros de datos modernos [6]. Actualmente, los entornos HPC y Cloud son muy utilizados para albergar este tipo de algoritmos, lo cual nos ha conducido a seleccionarlos como objeto de estudio. A este respecto, este trabajo presenta una *suite* de cargas de Machine Learning clásico transpiladas directamente a código C puro mediante la herramienta m2cgen. Esta suite permite evaluar el comportamiento simbiótico de cargas de IA modernas a nivel de ciclo, evitando el incremento del tiempo de simulación que introducen las cargas que utilizan lenguajes de alto nivel como Python.

En resumen, este trabajo presenta dos contribuciones principales: i) una infraestructura para la evaluación de políticas de planificación multihilo SMT en gem5 y ii) una suite de cargas de Machine Learning adaptadas para la simulación a nivel de ciclo. Estas contribuciones se validan mediante la implementación de una política FIFO donde las cargas de IA llegan al sistema siguiendo una distribución de Poisson. Los resultados demuestran patrones coherentes con lo esperable en un sistema real, lo que habilita estudios posteriores de propuestas de políticas de planificación avanzadas para sistemas multihilo SMT.

II. Antecedentes: limitaciones de los modos de ejecución del simulador gem5

El simulador gem5 [5] es una de las herramientas más utilizadas en la investigación sobre arquitectura de computadores para evaluar diseños a nivel de sistema y de microarquitectura. Se trata de un simulador modular y de código abierto que permite modelar distintos componentes *hardware* (CPU, memoria, interconexión, etc.) con distintos niveles de detalle, lo que lo hace especialmente adecuado para explorar el espacio de diseño sin necesidad de prototipos físicos. El simulador ofrece dos modos principales de ejecución: *syscall emulation* (SE) y *full system* (FS). En el modo SE, gem5 ejecuta aplicaciones en espacio de usuario interceptando y emulando las llamadas al sistema, sin simular un sistema operativo completo, lo que simplifica la configuración y permite simulaciones más rápidas. Por el contrario, el modo FS simula un sistema completo, incluyendo el *hardware*, el sistema operativo y sus interacciones, ejecutando un kernel real y modelando aspectos como la gestión de memoria, las interrupciones o la planificación de procesos.

En la versión actual de gem5, el único modo que ofrece capacidad de realizar y evaluar políticas de planificación es el modo FS, ya que en el modo SE las aplicaciones a ejecutar se definen estáticamente antes de comenzar la simulación y todas se inician inmediatamente al comenzar esta. Esto impide determinar los intervalos de tiempo donde una aplicación se ejecuta así como que nuevas aplicaciones entren en el sistema durante la ejecución de forma dinámica. Tampoco permite simular situaciones de *oversubscription*, donde el sistema debe planificar la ejecución de un número de hilos mayor que los soportados por el *hardware* simulado. En comparación, el modo FS simula un sistema operativo completo y, por tanto, no sufre las mismas limitaciones que el modo SE en lo que respecta a la planificación de aplicaciones.

Lamentablemente, FS no soporta actualmente la simulación de núcleos de cómputo SMT, por lo tanto, en el simulador gem5, actualmente no se cuenta con la infraestructura necesaria para diseñar y evaluar políticas de planificación y ubicación de hilos en núcleos SMT.

Teniendo todo esto en cuenta, en este trabajo proponemos una modificación de gem5 que permite realizar este tipo de estudios. Como se verá más adelante, la propuesta explota el soporte a la emulación de llamadas al sistema de SE para introducir la capacidad de ejecutar aplicaciones en cualquier contexto *hardware* simulado (ya sea de núcleos con soporte monohilo o multihilo). Esta propuesta resulta mucho menos compleja que lo que sería la implementación de SMT en FS y ofrece, con mínimas modificaciones en el código de gem5, la capacidad de seleccionar y ubicar las aplicaciones dinámicamente en un sistema SMT. A su vez, se presenta un estudio y caracterización de cargas de ML clásico sacadas de una suite de *benchmarks* del estado del arte, demostrando la utilidad del sistema desarrollado.

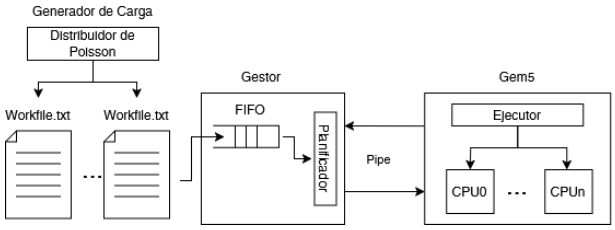


Fig. 1: Diagrama de flujo de la infraestructura desarrollada.

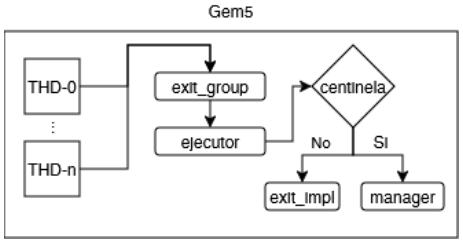


Fig. 2: Flujo de llamadas al sistema gem5.

III. Infraestructura desarrollada

La Figura 1 muestra un diagrama de la infraestructura desarrollada. Esta se compone de 4 partes, descritas a continuación.

A. El ejecutor

El ejecutor es la parte esencial de la infraestructura. El código del ejecutor está integrado dentro del propio simulador gem5. Dentro de gem5, existen tantas instancias del ejecutor como contextos *hardware* soporta la máquina simulada. Así, por ejemplo si se simula un procesador con 4 núcleos SMT2, se dispondrá de 8 (4×2) ejecutores.

Como se ha mencionado anteriormente, en modo SE, las llamadas al sistema se emulan mediante código del simulador, el cual está implementado en C++. Cuando un hilo de ejecución finaliza, realiza una llamada al sistema que lleva a la ejecución del método de gem5 denominado `exit_group`, que se encarga de la finalización correcta de la simulación del hilo.

Para habilitar la ejecución dinámica de aplicaciones (es decir, que nuevas aplicaciones lleguen al sistema a lo largo de la simulación) y posibilitar la planificación de estas, se ha interceptado la llamada a `exit_group`, donde se ha añadido el código del ejecutor, como se muestra en la Figura 2.

Lo primero que realiza el ejecutor es comprobar si en el sistema existe un archivo al cual llamamos centinela, que habilita la planificación dinámica. En caso de que la planificación dinámica no esté habilitada, la ejecución del hilo finaliza de forma normal. En caso contrario, el ejecutor utiliza tuberías de comunicación entre procesos (*named pipes*) para establecer una comunicación con el gestor de la planificación o *manager*, descrito posteriormente. En dicha comunicación, el ejecutor informa al gestor de que la aplicación ha finalizado su ejecución y queda a la espera de que el gestor le envíe el comando para ejecutar una nueva aplicación. Este comando se compone de la ruta al binario de la nueva aplicación y las opciones con las que debe ejecutarse.

Una vez se recibe el comando, el ejecutor realiza una llamada al método `execve`, el cual emula en gem5 la llamada al sistema utilizada para reemplazar la aplicación actual por una nueva aplicación. De esta forma, la llamada inicial a `exit_group` realizada por la aplicación que finaliza, se sustituye por una llamada a `execve`, que no solo finaliza la aplicación actual, sino que además lanza una nueva aplicación de acuerdo al algoritmo de planificación implementado en el gestor.

En la implementación desarrollada, el ejecutor no realiza ninguna lógica de planificación, sino que simplemente se encarga de llevar a cabo las decisiones del gestor. Hemos optado por esta implementación modular porque reduce al máximo las modificaciones sobre gem5 y permite la implementación de políticas de planificación en código externo implementado en lenguajes de más alto nivel como Python, efectivamente limitando la inclusión de nuevas dependencias en un código ya bastante complejo como es el de gem5.

B. El gestor o manager

El gestor es el encargado de implementar las políticas de planificación, así como de configurar la infraestructura subyacente de tuberías y de comunicación entre procesos. El gestor recibe trabajo mediante uno o más ficheros de entrada denominados `workfiles`. En dicho fichero se encuentran las cargas que se van a ejecutar, sus opciones y los instantes de tiempo en los que llegan al sistema. Nótese que para trabajar, el gestor emplea el tiempo interno de gem5 como referencia. Dicho tiempo de simulación, ajeno al tiempo real, se mide en picosegundos y es reportado por los ejecutores cada vez que una aplicación finaliza.

A partir de los `workfiles`, el gestor carga los trabajos en una cola interna, la cual emplea para realizar la planificación. El funcionamiento es el siguiente. El gestor lee los `workfiles` y extrae la configuración inicial de la cola (trabajos con instante de llegada 0). Posteriormente, lanza gem5 a ejecución configurando la carga inicial en el instante 0. A partir de ese momento, el gestor espera a que una aplicación de la carga finalice, lo cual ocurrirá cuando reciba el mensaje correspondiente del ejecutor asociado al contexto de la aplicación que termina. En ese momento, el gestor actualiza el tiempo interno con la marca de tiempo de gem5 proporcionada por el ejecutor, carga en la cola de planificación los trabajos de los `workfiles` que hayan llegado antes de ese momento y planifica la siguiente aplicación para indicársela al ejecutor.

C. Generador de carga

Para simular la ejecución de cargas en un entorno real, se ha empleado un generador *open-loop* que utiliza una lambda por carga y un intervalo de tiempo por experimento para generar una traza. Siguiendo las metodologías más recientes para la caracterización de estrés en colas de microservicios [7], se utiliza una distribución de Poisson para generar dicha

Tabla I: Parámetros de la microarquitectura.

i9-12900k	
Componentes	Número de entradas
CPU width	6
ROB	512
IQ	248
LQ/SQ	192/114
Register File Int/FP	288/248

Tabla II: Parámetros de la jerarquía de memoria.

Caché	**Parámetros**	
	Tamaño/vías	Tiempo de acceso
L1D	48KiB/12	5 ciclos
L1I	32/8 KiB	5 ciclos
L2	1,28MiB/10	15 ciclos
DDR4 3200	32 GiB	41,25 ns

Tabla III: Algoritmos de IA evaluados agrupados por los parámetros del *dataset* sintético.

Workload (M × F)	**Algoritmos Evaluados**
100k × 20	`elastic_net`, `lasso`, `lin_reg`, `log_reg`, `pca`, `ridge`
100k × 10	`dtree`, `dtree_reg`, `extra_trees`, `mnb_class`, `n_centroid`
50k × 10	`rf_class`
10k × 10	`gnb_class`, `kmeans`, `qda`
2k × 10	`svc_rbf`
1k × 10	`knn`, `radius_nn`

traza, lo que permite evaluar el sistema bajo ráfagas de estrés realistas.

D. Planificador

La planificación de hilos para procesadores multihilo se divide en dos decisiones esenciales: qué hilos se planifican juntos y qué núcleo se les asigna. Dichas decisiones dependen en gran medida de la máquina en la que se implementan, ya que cada procesador cuenta con una microarquitectura y una jerarquía de memoria propias, entre otros elementos. En topologías clusterizadas, no todos los núcleos comparten la caché de último nivel (LLC). Como evidencian los modelos analíticos más recientes sobre la contención de recursos en arquitecturas multinúcleo modernas [8], ignorar esta compartición asimétrica al asignar hilos provoca una degradación severa e impredecible del rendimiento global.

El criterio empleado para seleccionar hilos puede basarse en muchas métricas distintas, como, por ejemplo, el tiempo que lleve la aplicación en la cola, el uso de la CPU, el comportamiento simbiótico con aplicaciones en ejecución o con políticas de prioridad, entre otras. También la decisión de a qué núcleo enviar puede abordarse desde muchas estrategias distintas.

En este trabajo, para validar la infraestructura se ha optado por una políticica FIFO simple. Esta política implementa una única cola de entrada, a la que se añaden dinámicamente las aplicaciones en el orden de llegada descrito en los `workfiles`. Cuando un contexto de ejecución queda libre, esta política selecciona la aplicación situada a la cabeza de la cola para ejecutarla.

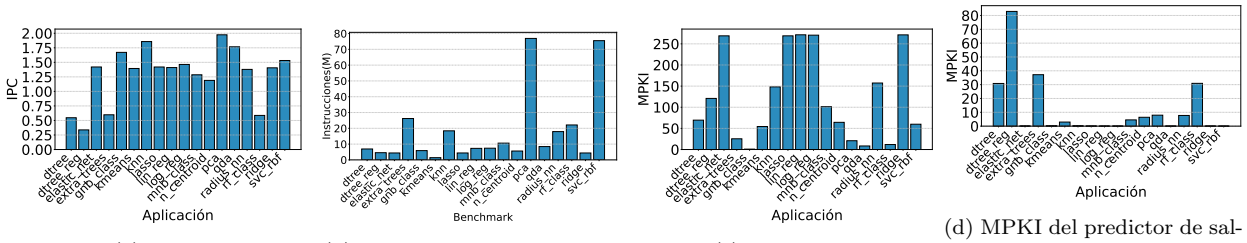
IV. Entorno experimental y metodología

Los experimentos se han llevado a cabo en un clúster x86 equipado con un procesador Intel(R) Xeon(R) Gold 5120. Se ha utilizado el simulador gem5. El simulador se ha configurado para imitar un procesador de estado del arte para computación de servidores, como el Intel i9-12900K. Los parámetros de la microarquitectura se han configurado tal y como se detalla en la Tabla I. Asimismo, se ha implementado la jerarquía de memoria de un sistema real, como se muestra en la Tabla II. La ejecución se ha configurado en 1 GHz.

Para la evaluación experimental, se han empleado cargas de ML clásico provenientes de scikit-learn, enumeradas en la Tabla III. Dichas cargas abarcan un gran número de modelos distintos con características distintas. Dichos modelos, para optimizar su ejecución en el simulador, se han entrenado en Python y posteriormente transpilado a C mediante la herramienta de código abierto m2cgen para ejecutarlos en el simulador. Una ventaja diferencial de esta infraestructura es el aislamiento del *kernel* algorítmico puro. Como evidencian estudios sobre la caracterización de cargas de inteligencia artificial [9], la evaluación directa de modelos en lenguajes de alto nivel introduce una severa sobrecarga derivada de características propias de dichos lenguajes, como puede ser el recolector de basura. Además, los lenguajes interpretados requieren de otro programa que corra simultáneamente para traducir dinámicamente el *software* a código máquina. Al transpilar los modelos de scikit-learn a C estático, eliminamos esta sobrecarga, lo que permite que gem5 simule únicamente el estrés que el cómputo del modelo ejerce sobre la microarquitectura.

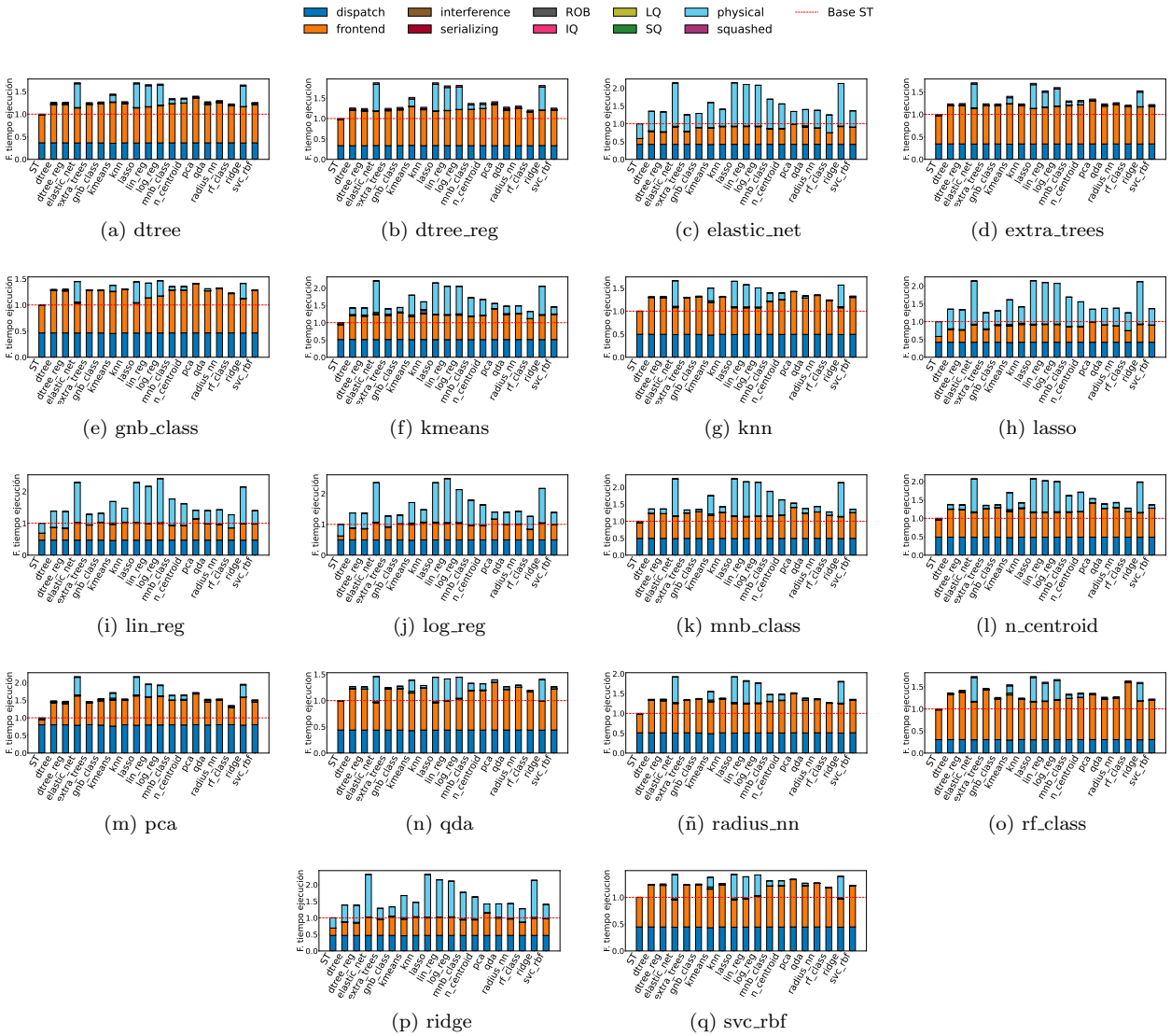
La metodología de evaluación consta de dos experimentos. El primer experimento contempla la caracterización en solitario mediante la ejecución de las cargas, en el que se han medido su IPC, MPKI y el número total de instrucciones. En línea con las metodologías modernas de caracterización de contención de recursos [10], este perfilado base busca estudiar la huella microarquitectónica de cada modelo para predecir posibles comportamientos simbióticos y de interferencia durante su ejecución multiprograma.

Para el segundo experimento, que replica las metodologías estándar de evaluación de servicios interactivos y cargas con una duración de microsegundos [11], se ha empleado un generador de carga *open-loop* basado en un proceso de Poisson. Manteniendo una longitud fija de experimento (20 ms), se ha variado la frecuencia de llegada de las peticiones para estudiar métricas a nivel de sistema bajo distintos niveles de estrés. Se ha monitorizado el tamaño de la cola durante la ejecución, el tiempo en cola (tiempo transcurrido desde que una aplicación entra al sistema hasta que es atendida) y el tiempo de ejecución. Para capturar con precisión la latencia de cola (*tail latency*), crítica en entornos de centros de datos, se han reportado la media, la mediana y el percentil 97,5 de cada métrica. Finalmente, para cada frecuencia de llegada



(a) IPC. (b) Millones de Instrucciones. (c) MPKI$_{L1D}$. (d) MPKI del predictor de saltos.

Fig. 3: Métricas microarquitectónicas de las aplicaciones sk-learn.



(a) dtree (b) dtree_reg (c) elastic_net (d) extra_trees

(e) gnb_class (f) kmeans (g) knn (h) lasso

(i) lin_reg (j) log_reg (k) mnb_class (l) n_centroid

(m) pca (n) qda (ñ) radius_nn (o) rf_class

(p) ridge (q) svc_rbf

Fig. 4: Degradación del rendimiento de las cargas en solitario y en compañía.

se han generado 40 trazas únicas, lo que garantiza la solidez estadística de los resultados experimentales.

V. Evaluación

Al analizar las prestaciones de un sistema de colas como el representado en la Figura 9, es necesario definir las métricas a medir y evaluar. Este tipo de sistemas presenta una arquitectura que se rige por las siguientes normas. Un conjunto de aplicaciones independientes llega al sistema a distintas frecuencias; dichas aplicaciones se almacenan en una cola hasta que el planificador las lanza a la ejecución. Una vez ejecutadas, estas aplicaciones salen del sistema con una frecuencia predeterminada. La frecuencia de llegada se conoce como λ; el tiempo que dicha aplicación espera en la cola se conoce como tiempo en cola; el tiempo que tarda la aplicación en ejecutarse una vez planificada se conoce como tiempo de servicio. La frecuencia con la que salen las aplicaciones se conoce como productividad.

Para llevar a cabo la evaluación experimental, se han empleado dos perspectivas totalmente complementarias: i) comportamiento a nivel microarquitectónico de las aplicaciones en solitario y degradación de rendimiento en ejecución SMT y ii) un análisis profundo del comportamiento a nivel de sistema de una política de planificación FIFO evaluada bajo distintas condiciones de carga simulando un entorno

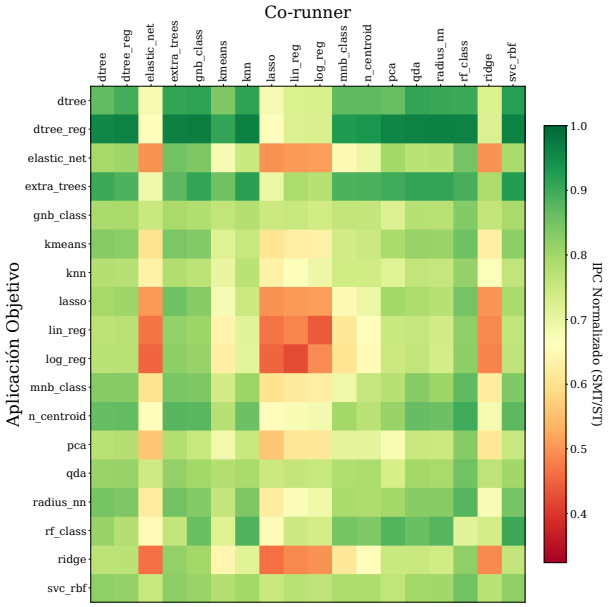


Fig. 5: Mapa de calor de la degradación de rendimiento en multihilo.

real. Cabe destacar que el objetivo de esta sección no es proponer ni evaluar la idoneidad de la politica FIFO frente a planificadores comerciales, sino utilizarla como un estudio base para validar la infraestructura desarrollada.

A. Ejecución en solitario

Como podemos observar en la Figura 3a, las cargas analizadas presentan tres comportamientos diferenciados en relación al IPC; cargas como `dtree`,dtree_reg, `extra_trees` y `rf_class` presentan un IPC bajo, inferior a 0,5. Por otro lado, observamos un segundo grupo formado por cargas como `elastic_net`, cuyo IPC se podría considerar medio, con valores entre 1,25 y 1,5. Para finalizar, observamos un tercer grupo de aplicaciones, como `pca` y `qda`, que alcanzan valores de IPC altos, superando 1,75. Dicha variabilidad del IPC no es aleatoria, sino que se debe a características intrínsecas de las aplicaciones y a sus patrones de consumo de recursos. La Figura 3d muestra los fallos del predictor de salto por cada mil instrucciones; como podemos observar, los algoritmos basados en árboles son los que presentan un IPC más bajo debido a la alta tasa de fallos de predicción de salto, inherente a este tipo de algoritmos.

El análisis revela comportamientos heterogéneos entre las cargas estudiadas, lo cual habilita la explotación de dichas diferencias de comportamiento, permitiendo políticas de planificación que hagan uso de comportamientos simbióticos.

Otro análisis importante es el de la cantidad de instrucciones ejecutadas por carga; la Figura 3b muestra que, en este aspecto, tampoco existe homogeneidad entre las cargas. Este análisis es muy útil ya que, sumado a las frecuencias de llegada al sistema, supone una gran diferencia en la utilización de la CPU presentada por las cargas, lo cual puede servir de base para políticas de planificación que tengan en cuenta la utilización.

Finalmente, la Figura 3c muestra los fallos cada mil instrucciones (*Misses Per Kilo-Instruction* o MPKI) de la cache de datos de primer nivel. Esta figura nos muestra patrones de consumo y estrés distintos, lo que supone un factor a tener en cuenta a la hora de emparejar aplicaciones.

B. Degradación de prestaciones

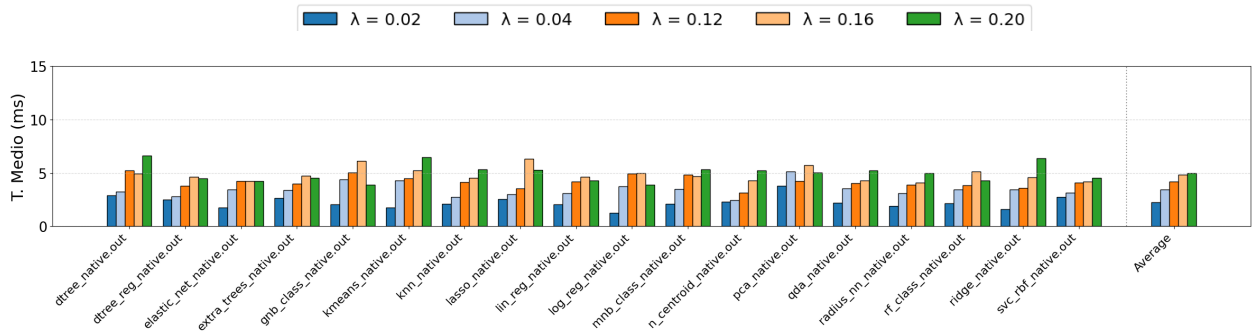
La Figura 4 muestra la pila de prestaciones (*performance stack*) para cada aplicación cuando se ejecuta en compañía de otras aplicaciones en el mismo núcleo normalizada con respecto a su ejecución en solitario. Como podemos observar, la competición por recursos compartidos y los heterogéneos patrones de acceso conducen a que cada aplicación sufra una degradación distinta cuando se ejecuta en compañía dependiendo de su pareja en el núcleo. De esta forma, existen combinaciones con distintos grados de simbiosis que se pueden explotar a la hora de realizar una política de planificación. Este comportamiento se puede ver más claramente en el mapa de calor presentado en la Figura 5. Para obtener el mapa, la degradación (D) normalizada del IPC se calcula siguiendo la ecuación:

$$D_{i,j} = \frac{IPC_{i,j}^{SMT}}{IPC_{i}^{ST}}, \tag{1}$$

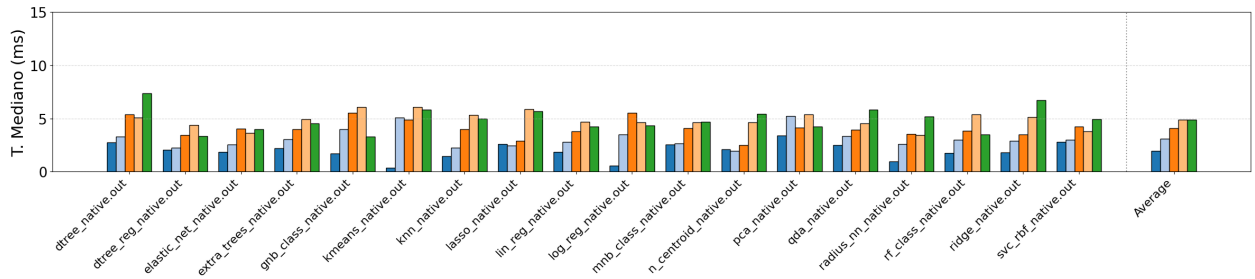
donde IPC_i^{ST} representa el IPC de la aplicación i en solitario (*Single Thread* o ST) e $IPC_{i,j}^{SMT}$ representa su IPC cuando se ejecuta en compañía de la aplicación j en un núcleo SMT.

Atendiendo al mapa de calor, se pueden observar tres zonas claramente diferenciadas:

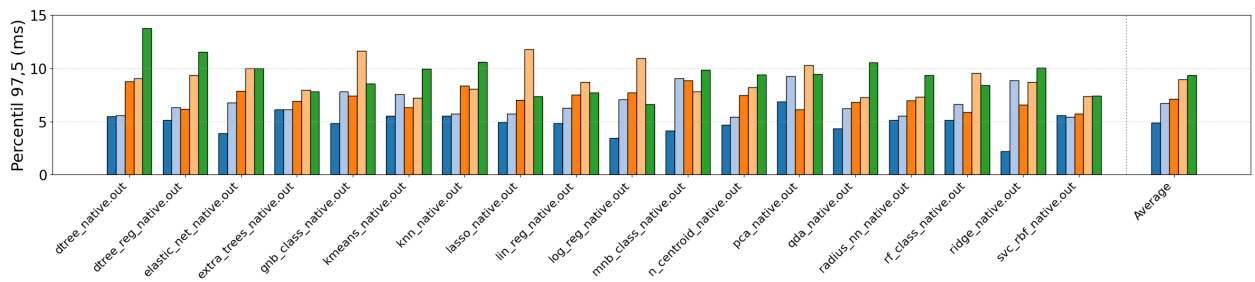
- Zona de alta interferencia: Se identifica un clúster de alta degradación representado por los colores naranja y rojo, en la ejecución simultánea de modelos lineales, como es el caso de `lasso`, `lin_reg` o `log_reg`. Esto se debe a que dichos algoritmos comparten patrones de consumo de recursos similares, específicamente en la jerarquía de memoria, como se observa en la Figura 3c. De esto se desprende que su ejecución simultánea produce una degradación del rendimiento normalizado cercana al 0,5, lo que prácticamente anula las ventajas del SMT.
- Zona de alta simbiósis: En el mapa de calor, se observan grandes regiones de parejas cuya ejecución simultánea apenas repercute en pérdidas de prestaciones (región verde), con degradaciones del rendimiento cercanas al 0,1 normalizado. Un claro ejemplo de esto es la mezcla de aplicaciones con un alto IPC (`gnb_class`) con apenas acceso a la jerarquía de memoria, con aplicaciones con un IPC menor (`n_centroid`) pero con mayor acceso a memoria. Dichos patrones de consumo complementarios permiten que la ejecución simultánea apenas degrade el rendimiento de dichas aplicaciones.



(a) Tiempo medio en cola.



(b) Tiempo mediano en cola.



(c) Percentil 97,5 del tiempo en cola.

Fig. 6: Tiempo de cola por aplicación con distintas frecuencias de llegada para 20 ms.

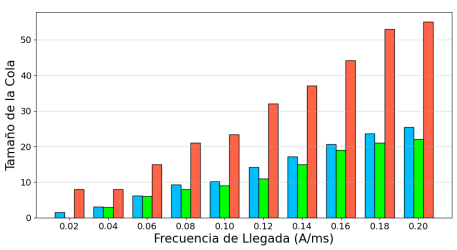


Fig. 7: Media, Mediana y Percentil 97.5 % del tamaño de la cola de jobs bajo distintas condiciones de carga.

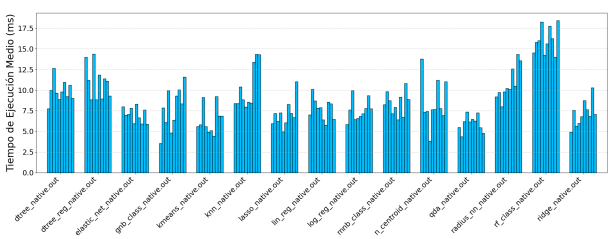


Fig. 8: Tiempo de ejecución medio.

Este análisis microarquitectónico demuestra la utilidad de nuestro sistema para satisfacer las necesidades de análisis y estudio de las cargas y, así, desarrollar políticas de planificación inteligentes.

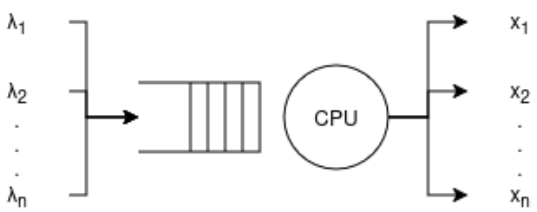


Fig. 9: Diagrama de flujo de un sistema.

C. Caso de estudio: política FIFO

La Figura 6 muestra el tiempo de cola obtenido por cada aplicación bajo distintas frecuencias de llegada (λ) durante el mismo intervalo de tiempo, en este caso, 20 milisegundos. La política, bajo estas condiciones de carga, es capaz de mantener también una baja cantidad de aplicaciones en la cola, como se muestra en la Figura 7.

También podemos observar que el tiempo medio de cola presenta una fuerte degradación paulatina a medida que aumenta la frecuencia de llegada (de la Figura 6a a la Figura 6c). Al mismo tiempo, también observamos cómo se presenta mayor dispersión en las medianas, lo que muestra un sistema cercano a la saturación.

En la Figura 8, se representan los tiempos medios de ejecución de las aplicaciones a lo largo de las distintas configuraciones. Cada barra corresponde al promedio del tiempo de ejecución obtenido por dicha aplicación en 40 ejecuciones con las mismas condiciones iniciales. Como se puede observar, existe una gran variabilidad y no se muestra una tendencia clara de aumento del tiempo de ejecución. De esta forma, se puede observar cómo la carga o saturación del sistema no afecta de manera tan importante al tiempo de ejecución de una carga como otros factores, como la simbiosis. Nótese que la política FIFO no explota el potencial comportamiento simbiótico de las aplicaciones.

Los resultados experimentales muestran resultados coherentes con lo esperado en un sistema real bajo las mismas condiciones de carga. A través de la evaluación experimental, podemos observar cómo nuestra infraestructura es capaz de caracterizar el rendimiento de un sistema, y demostrar su capacidad para capturar fenómenos de contención, saturación o rendimiento de las cargas en entornos SMT.

VI. Trabajos relacionados

Nuestra infraestructura de planificación de hilos se enmarca en la convergencia en la investigación entre arquitecturas de computadores y sistemas operativos. Para contextualizar la propuesta, nos centramos en tres ámbitos clave: el diseño de políticas de planificación, los estudios microarquitectónicos basados en simulación y la evaluación de cargas de ML clásico.

A. Politicas de planificación

Para maximizar la productividad del *hardware* subyacente, las tendencias actuales se centran en reducir la contención entre hilos. Típicamente, esto se aborda mediante la partición estricta de recursos y la gestión de QoS a nivel de nodo, como demuestran Heracles [12] y PARTIES [13]. Algunas propuestas, como Arachne [14], ante los elevados costes de los cambios de contexto en la ejecución de cargas modernas, proponen trasladar la planificación al espacio de usuario. En la frontera de este enfoque de baja latencia surge el *blind scheduling* de microsegundos para centros de datos, propuesto en Tiny Quanta [15]. Al gestionar tareas extremadamente cortas, este sistema sustituye las interrupciones de *hardware* por corrutinas ligeras y una planificación jerárquica que evita bloqueos. El uso de nuestro entorno de simulación beneficiaría enormemente a todas estas propuestas, ya que permite validar su eficacia y heurísticas utilizando estadísticas detalladas de bajo nivel que es muy difícil obtener en sistemas reales.

B. Estudios basados en simulación

Evaluar escenarios modernos de planificación en simuladores precisos es complejo: el modo SE de gem5 carece de flexibilidad para cargas dinámicas y el modo FS resulta inasumible por su sobrecoste. Soluciones como TaskPoint [16] resuelven el tiempo de ejecución mediante muestreo estadístico, mientras que alternativas como ZSim [17] o la simulación basada en trazas sacrifican la fidelidad arquitectónica. Por otro lado, el desacoplo de la lógica mediante sockets (dist-gem5 [18]) introduce latencias de red ajenas al sistema simulado. Nuestra infraestructura elimina estos compromisos, permitiendo un control directo y determinista del simulador sin perder la precisión de un modelo ciclo a ciclo.

C. Cargas de ML

La evaluación microarquitectónica del ML clásico enfrenta dos grandes retos: las suites actuales, como Fathom [19] o MLPerf Tiny [20], se enfocan predominantemente en arquitecturas de aprendizaje profundo (Deep Learning), y el uso de frameworks de alto nivel introduce un ruido métrico severo que dificulta el perfilado de *hardware*, especialmente debido al 'impuesto del intérprete' inherente a muchos de sus entornos de inferencia. Alineándonos con la filosofía de aislar el kernel puro (bare-metal) — similar a los principios de diseño de MachSuite [21], que emplea implementaciones en un subconjunto estático de C, libre de asignaciones dinámicas y punteros complejos, para una evaluación precisa del *hardware* — utilizamos m2cgen para transformar modelos predictivos en código C estático. Esto permite generar un espectro ilimitado de cargas deterministas con distintos hiperparámetros. El resultado es un banco de pruebas de alta fidelidad que aísla el impacto real de la algoritmia compleja sobre la jerarquía de memoria, libre de las capas de abstracción del *software*.

VII. Conclusiones y trabajo futuro

Con este trabajo hemos presentado el desarrollo e implementación de una infraestructura total para habilitar la creación y estudio de políticas de planificación de hilos en procesadores multihilo, extendiendo y mejorando el simulador gem5. A diferencia de las aproximaciones tradicionales, la solución propuesta permite modelar la llegada de aplicaciones en tiempo de ejecución, su gestión mediante colas y su asignación a los recursos del sistema de forma controlada y reproducible.

La evaluación experimental, basada en el estudio detallado de una política FIFO actuando como un caso de uso funcional mediante cargas de trabajo de IA generadas por distribuciones de Poisson, ha permitido validar el correcto funcionamiento de la infraestructura y caracterizar el comportamiento del sistema bajo distintas condiciones de carga, demostrando así la utilidad de la propuesta para realizar los análisis necesarios de este tipo de cargas para entornos HPC y Cloud.

Como trabajo futuro, se plantea emplear la infraestructura desarrollada para el desarrollo de políticas de planificación de hilos, las cuales se beneficiarán sobremanera de la infraestructura desarrollada. En conclusión, este trabajo establece una base sólida para la investigación en planificación de hilos en entornos multihilo en gem5.

Agradecimientos

Este trabajo ha sido financiado por MICIU/AEI/10.13039/501100011033/ FEDER, UE, a través de los proyectos PID2024-158682OB-C31 y PID2021-123627OB-C51, junto con la Generalitat Valenciana a través del proyecto CIPROM/2024/7. Manuel A. Pecino está financiado por la beca PREP2024-002924.

Referencias

[1] Svilen Kanev, Juan Pablo Daraghmeh, Alexander Bowden, and Parthasarathy Ranganathan, "Profiling a warehouse-scale computer," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 158–169.

[2] Dean M Tullsen, Susan J Eggers, and Henry M Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," in *Proceedings of the 22nd annual international symposium on Computer architecture*, 1995, pp. 392–403.

[3] Marta Navarro, Josué Feliu, Salvador Petit, María E Gómez, and Julio Sahuquillo, "Synpa: Smt performance analysis and allocation of threads to cores in arm processors," in *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2024, pp. 1–11.

[4] Marta Navarro, Josué Feliu, Salvador Petit, María E Gómez, and Julio Sahuquillo, "Thread-to-core allocation in arm processors building synergistic pairs," in *2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2023, pp. 1–2.

[5] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Alitheia Rollins, Jeff Rosenfeld, Eric Leary, Lisa Moore, Ismail Hamadeh, Eno Adegbore, and Ali Saidi, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, may 2011.

[6] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, et al., "Machine learning at facebook: Understanding inference at the edge and in the cloud," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 331–344.

[7] Ratnadeep Bhattacharya and Timothy Wood, "Dynamically balancing load with overload control for microservices," *ResearchGate*, 2024.

[8] Yongju Lee, Jaewon Kwon, Cheolhwan Kim, Enhyeok Jang, Jiwon Lee, Hyunwuk Lee, and Won Woo Ro, "Cosmos: An llc contention slowdown model for heterogeneous multi-core systems," in *2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2025.

[9] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, et al., "Applied machine learning at facebook: A datacenter infrastructure perspective," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 620–629.

[10] Ennan Zhai et al., "Dirigent: Lightweight serverless orchestration for datacenter resource efficiency," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.

[11] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Aasheesh Janjua, Michael Kelley, Timothy Osborn, Charlotte Slavin, et al., "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019, pp. 3–18.

[12] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis, "Heracles: Improving resource efficiency at scale," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA '15)*, Portland, OR, USA, 2015.

[13] Shuang Chen, Christina Delimitrou, and José F. Martínez, "Parties: Qos-aware resource partitioning for multiple interactive services," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*, Providence, RI, USA, 2019.

[14] Henry Qin, Qian Li, Jacqueline Speiser, Peter Kraft, and John Ousterhout, "Arachne: Core-aware thread management," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI '18)*, Carlsbad, CA, USA, 2018.

[15] Zhihong Luo, Sam Son, Dev Bali, Emmanuel Amaro, Amy Ousterhout, Sylvia Ratnasamy, and Scott Shenker, "Efficient microsecond-scale blind scheduling with tiny quanta," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*, La Jolla, CA, USA, 2024.

[16] Thomas Grass, Alejandro Rico, Marc Casas, Miquel Moreto, and Eduard Ayguade, "Taskpoint: Sampled simulation of task-based programs," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2016, pp. 222–232.

[17] Daniel Sanchez and Christos Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA '13)*, Tel-Aviv, Israel, 2013.

[18] Mohammad Alian, Gabor Dozsa, Umur Darbaz, Stephan Diestelhorst, Daehoon Kim, and Nam Sung Kim, "distgem5: Distributed simulation of computer clusters," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2017, pp. 276–285.

[19] Robert Adolf, Saketh Rama, Brandon Reagen, Gu-Yeon Wei, and David Brooks, "Fathom: Reference workloads for modern deep learning methods," 2016.

[20] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, Urmish Thakker, Antonio Torrini, Peter Warden, Jay Cordaro, Giuseppe Di Guglielmo, Javier Duarte, Stephen Gibellini, Videet Parekh, Honson Tran, Nhan Tran, Niu Wenxu, and Xu Xuesong, "Mlperf tiny: A benchmark suite for machine learning on edge devices," *Preprint. Under review.*, 2021.

[21] Brandon Reagen, Robert Adolf, Yakun Sophia Shao, Gu-Yeon Wei, and David Brooks, "Machsuite: Benchmarks for accelerator design and customized architectures," 2014.

[22] Shridhar Joshi, Sumant Kalra, et al., "Simulation-driven design of large-scale systems architecture," in *2024 IEEE 31st International Conference on High Performance Computing (HiPC)*. IEEE, 2024.

Redes de comunicaciones (1/2)

Análisis del impacto de la comunicación intra-nodo en el rendimiento de las redes de interconexión de sistemas para HPC e IA

Joaquin Tarraga-Moreno[1], Jesus Escudero-Sahuquillo[1], Pedro Javier Garcia[1] y Francisco J. Quiles[1]

Resumen— **En la última década, se han incorporado dispositivos de cómputo y de almacenamiento con fines específicos, como GPU, TPU o almacenamiento de alta velocidad, a los nodos de servidor de superordenadores y centros de datos. El desarrollo de la memoria de alto ancho de banda (HBM) ha permitido un formato mucho más compacto para estos dispositivos, lo que ha facilitado la interconexión de varios de ellos dentro de un nodo de servidor, normalmente mediante una red de interconexión intra-nodo (por ejemplo, PCIe, NVLink o Infinity Fabric). Estas redes permiten aumentar el número de dispositivos informáticos y de almacenamiento específicos por nodo. Además, las redes inter-nodo comunican miles de estos dispositivos colocados en diferentes nodos de servidor en un superordenador o centro de datos. Desgraciadamente, las redes intra e inter-nodo pueden convertirse en el cuello de botella del sistema debido a la creciente demanda de comunicación entre los aceleradores de aplicaciones como la IA generativa. Aunque los diseños actuales de redes intra-nodo alivian este cuello de botella al aumentar el ancho de banda de la red intra-nodo, en este artículo demostramos que un ancho de banda tan alto para la comunicación intra-nodo puede obstaculizar el rendimiento de la comunicación inter-nodo cuando el tráfico procedente del exterior del nodo llega a los dispositivos intra-nodo, lo que da lugar a competición con el tráfico intra-nodo. Para analizar el impacto de esta interferencia, hemos estudiado las operaciones de comunicación de patrones de tráfico realistas que explotan la comunicación intra-nodo. Hemos desarrollado un modelo genérico de simulación intra e inter-nodo basado en OMNeT++ y hemos modelado las operaciones de comunicación mencionadas. También hemos realizado extensos experimentos de simulación que confirman que aumentar el ancho de banda de la red intra-nodo y el número de dispositivos informáticos por nodo (es decir, aceleradores) es contraproducente para el rendimiento de la comunicación inter-nodo.**

Palabras clave— **Redes de interconexión de alto rendimiento, comunicación intra-nodo, aceleradores y GPU.**

I. Motivación

LOS sistemas para HPC e IA están compuestos por miles de nodos que incrementan su capacidad de cómputo, memoria y almacenamiento para afrontar aplicaciones como computación científica e IA generativa. La potencia de cálculo se incrementa mediante procesadores y aceleradores (GPUs, TPUs), mientras que tecnologías como HBM, NVMe o SCM aumentan la capacidad y el ancho de banda de acceso a datos, permitiendo situarlos cerca del procesamiento y reduciendo la latencia inter-nodo.

Los dispositivos dentro del nodo se interconectan mediante una red intra-nodo que debe evitar convertirse en cuello de botella. PCIe es la solución *de facto* (1, 2 y 4 GB/s por línea en Gen3, Gen4 y Gen5), junto con alternativas como NVLink (NVIDIA) o Infinity Fabric (AMD). El aumento de velocidad y heterogeneidad de dispositivos complica el diseño intra-nodo.

Millones de operaciones de comunicación se generan en redes intra- e inter-nodo. Una operación P2P consta de tres fases: fragmentación y envío intra-nodo, agregación y transmisión inter-nodo, y nueva fragmentación en destino. Si el destino es local, la comunicación permanece intra-nodo.

Las operaciones colectivas (por ejemplo, *AllReduce* o *Broadcast*) empleadas en el entrenamiento distribuido de DNNs pueden involucrar múltiples nodos cuando el modelo no cabe en uno solo [1]. En este caso, el modelo se distribuye en memoria local (por ejemplo, HBM) y el intercambio de gradientes genera altas tasas de tráfico. Aunque las redes de interconexión han sido ampliamente estudiadas, el impacto de las fases intra-nodo inicial y final en el rendimiento global sigue siendo un campo abierto [2]. Es necesario analizar los patrones de tráfico y su efecto en posibles cuellos de botella.

Este trabajo aborda la siguiente cuestión: *¿Hasta qué punto el tráfico inter-nodo impacta en el tráfico intra-nodo y viceversa?* Para responderla, modelamos una arquitectura genérica intra-nodo y patrones realistas en un simulador basado en OMNeT++. Existen simuladores como *gem5* [3] o TraceR-CODES [4], [5], pero carecen de granularidad o escalabilidad para modelar conjuntamente comunicación intra- e inter-nodo. Validamos el modelo mediante micro-benchmarks (InfiniBand Perftest) en un clúster real.

Evaluamos configuraciones de 32, 128 y 512 nodos (con 256, 1 024 y 4 096 aceleradores) utilizando patrones inspirados en entrenamiento de LLMs. Los resultados muestran que la red intra-nodo puede convertirse en cuello de botella por interferencia entre tráfico intra- e inter-nodo y por el overhead de agregación y fragmentación de paquetes. Estos efectos pueden mitigarse incorporando mecanismos específicos en los dispositivos de red.

El resto del artículo se organiza como sigue: la Sección II presenta los antecedentes; la Sección III describe el modelo; la Sección IV muestra los resultados; y la Sección V expone las conclusiones.

[1]Dpto. de Sistemas Informáticos, Universidad de Castilla-La Mancha, España, e-mail: antonioj.tarraga@uclm.es

II. Estado del arte

A. Dispositivos de un nodo

La CPU en los nodos de sistemas para HPC e IA gestiona procesos y coordina operaciones de software y hardware. En aplicaciones de IA generativa basadas en LLMs, prepara y preprocesa los datos antes de enviarlos a GPUs o TPUs para su entrenamiento.

La GPU es un procesador especializado en paralelismo masivo, con miles de núcleos para operaciones simultáneas, esencial en entrenamiento e inferencia de LLMs. Fabricantes como NVIDIA, AMD e Intel desarrollan aceleradores para cargas intensivas de IA. La NVIDIA Hopper GPU [6], integrada en el Grace Hopper Superchip [7], combina HBM, alta capacidad de cálculo (hasta 989,4 TFLOPS FP32) y comunicación NVLink, con hasta 144GB HBM3e. La Intel Gaudi GPU [8] ofrece hasta 229 TFLOPS y 128GB HBM.

Las SSD y tecnologías como NVMe o SCM proporcionan almacenamiento de alto rendimiento para IA y HPC. NVMe-oF mejora el acceso remoto a datos reduciendo la distancia entre cómputo y almacenamiento.

B. Redes de interconexión inter-nodo

Las redes inter-nodo permiten comunicación de alto rendimiento entre miles de nodos. Aunque históricamente diferían entre sistemas para HPC e IA, su arquitectura ha convergido [9]. Comparten diseño en topología, encaminamiento, control de flujo, QoS y gestión energética. La era post-Exascale y la IA generativa introducen nuevos desafíos [10].

Tecnologías como InfiniBand (NVIDIA), Slingshot (HPE/Cray) o iniciativas del Ultra Ethernet Consortium impulsan redes de alto rendimiento. RoCE habilita RDMA sobre Ethernet reduciendo latencia y sobrecarga de CPU, y se emplea en sistemas como Intel Gaudi3. Las velocidades actuales alcanzan 200, 400 y 800 Gbps, con previsión de Terabit Ethernet.

C. Redes de interconexión intra-nodo

La red intra-nodo debe proporcionar baja latencia, alto ancho de banda y eficiencia energética para comunicación entre CPU, aceleradores y memoria. La heterogeneidad actual introduce nuevos patrones de tráfico, superando diseños tradicionales basados en bus o anillo.

Soportan cargas como IA generativa con alto paralelismo [11], incorporando optimizaciones de hardware y técnicas avanzadas de control y encaminamiento.

PCIe es la interconexión principal (hasta 3.9 y 7.5 GB/s por línea en Gen5 y Gen6). Se utiliza tanto para escalar aceleradores como para conectar NICs inter-nodo. NVLink ofrece mayor ancho de banda (hasta 900 GB/s bidireccionales) y se emplea en sistemas como NVIDIA DGX. Otras soluciones incluyen Infinity Fabric (AMD), Quick Path Interconnect (Intel), UALink y CXL, que buscan unificar memoria y cómputo.

D. Patrones de Comunicación

Las aplicaciones de IA generativa requieren paralelización, ya que los LLMs no caben en un único dispositivo. El modelo de programación combina paralelismo de datos y de modelo.

Data Parallelism (DP) [12] divide el conjunto de datos entre dispositivos con copias idénticas del modelo, sincronizando gradientes mediante *AllReduce*.

Model Parallelism (MP) distribuye el modelo cuando excede la memoria de un dispositivo. Incluye *Tensor Parallelism* (TP) [13], con comunicación frecuente mediante *AllReduce* o *AllGather*, y *Pipeline Parallelism* (PP) [14], que divide el modelo en etapas y utiliza comunicación punto a punto entre dispositivos.

III. Modelado de la comunicación intra e inter-nodo

En esta sección se describe el modelo de simulación de redes intra- e inter-nodo desarrollado para estudiar el impacto de la comunicación intra-nodo en el rendimiento de las redes de interconexión intra- e inter-nodo, la caracterización de la comunicación entre dispositivos PCIe, el modelo genérico de simulación intra- e inter-nodo y el modelo de tráfico de red que reproduce operaciones de comunicación realistas en el entrenamiento paralelo de modelos de lenguaje grandes (LLM).

A. Arquitectura del nodo de referencia

En la herramienta de simulación, hemos modelado una arquitectura de nodo final basada en un sistema real que utiliza PCIe 3.0 para interconectar una GPU, un disco NVMe y una NIC InfiniBand, que se utilizará con fines de validación. La Figura 1 ilustra esta infraestructura conocida como *Cluster for the Evaluation of Low-Latency Architectures* (CELLIA) [1].

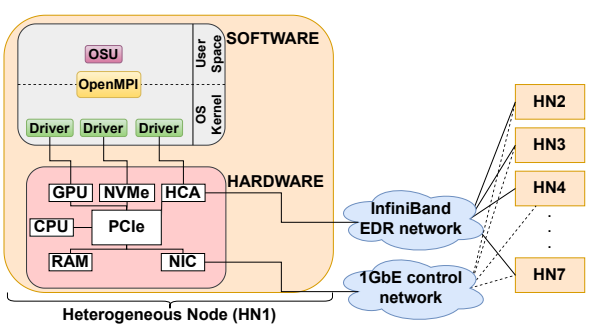


Fig. 1: Configuración de nodos en el clúster CELLIA.

El clúster CELLIA consta de varios nodos; cada nodo tiene dos CPU Intel Xeon Silver 4116 conectadas a PCIe Gen3 y 192 GB de RAM. Estos nodos también tienen una interfaz de red EDR InfiniBand (también conocida como HCA) MCX556A-ECAT ConnectX-5 de doble puerto. La tecnología InfiniBand EDR proporciona una velocidad de datos de 100 Gbps por puerto HCA (es decir, 12,5 GB/s),

[1]Más detalles en: `https://www.i3a.uclm.es/raap/?page_id=1650`.

con paquetes de 4 KiB de MTU. Estos paquetes contienen una cabecera de 64 B y una carga útil de 4032 B. CELLIA tiene una red de gestión de 1 GbE, por lo que cada nodo tiene una interfaz de red (NIC) para fines de gestión. Además, cada nodo tiene instalada una GPU NVIDIA Tesla T4 sobre PCIe Gen3 x16 y un disco NVMe Ultrastar SN200 Series NVMe SSD sobre PCIe Gen3 x8. Ten en cuenta que esta arquitectura de nodos heterogéneos tiene varios dispositivos conectados al PCIe (por ejemplo, la CPU, la GPU, el NVMe y el HCA), por lo que la red intra-nodo podría ser un cuello de botella cuando la comunicación inter-nodo involucra a otros dispositivos ubicados en diferentes nodos.

B. Modelo genérico de red intra-nodo

En nuestro simulador SAURON [15], hemos ampliado esta herramienta para modelar una red intra-nodo genérica y patrones de comunicación realistas. SAURON es un simulador altamente escalable que modela patrones de comunicación inter-nodo, topologías y configuraciones de enrutamiento. La Figura 2 muestra la arquitectura intra- e inter-nodo modelada en el simulador SAURON.

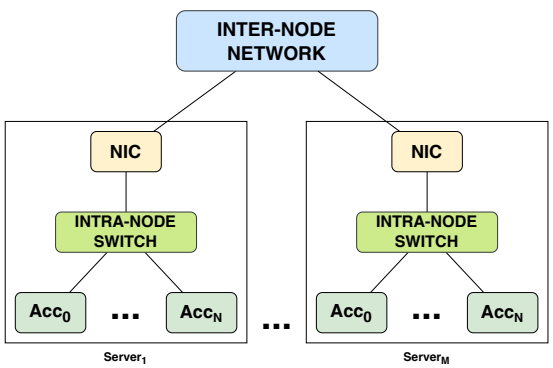


Fig. 2: Arquitectura genérica de red intra- y inter-nodo.

Suponemos un número configurable de aceleradores y otras unidades funcionales (por ejemplo, CPU o dispositivos de almacenamiento) interconectados ≪todos con todos≫ mediante un switch en la red intra-nodo. Este switch se puede configurar para proporcionar el ancho de banda deseado, de modo que la red intra-nodo pueda procesar el tráfico entrante dirigido a los dispositivos del nodo y el tráfico saliente que generan. De hecho, también podemos configurar el ancho de banda entre este switch y la NIC del nodo, es decir, el ancho de banda entre la red intra-nodo y la red inter-nodo. Cabe señalar que este modelo genérico permite configurar la red intra-nodo de acuerdo con las tecnologías actuales y emergentes (por ejemplo, PCIe 5.0, 6.0, 7.0, NVLink, UALink, etc.). De hecho, podemos configurar el número de líneas, el ancho de banda para los diferentes dispositivos de los nodos, su tamaño máximo de carga útil (MPS), etc.

Suponemos que los mensajes se generan en aceleradores (GPU, TPU, etc.) y unidades funcionales (CPU, NVMe, etc.), que deben dividirlos en paquetes (según el MPS intra-nodo) que se inyectan en la red intra-nodo. Los paquetes intra-nodo pueden dirigirse a otros dispositivos dentro del mismo nodo o colocarse en diferentes nodos. En este último caso, las NIC de los nodos convierten los paquetes intra-nodo en paquetes inter-nodo y los inyectan en la red inter-nodo. Esta conversión implica rellenar los paquetes inter-nodo con los encabezados y las cargas útiles de los paquetes de la red intra-nodo, lo que introduce una latencia y una sobrecarga adicionales.

Como se describe más adelante, en nuestro modelo de simulación podemos configurar el tamaño de los mensajes en los aceleradores y las unidades funcionales, la distribución de destinos y la tasa de generación. Esta flexibilidad permite reproducir patrones de comunicación realistas, como los observados en la fase de entrenamiento de los LLM, en la que se utilizan varios tipos de paralelismo.

C. Modelado genérico de la comunicación LLM

En nuestro simulador, hemos incluido un modelo genérico de tráfico de red basado en las operaciones de comunicación generadas por los modelos de grandes de lenguaje (LLM), que se utilizan habitualmente en sistemas para HPC e IA, tal y como se describe en la Sección II-D. Concretamente, el entrenamiento en aplicaciones basadas en IA sigue un patrón cíclico, en el que las operaciones de comunicación se repiten en los lotes de datos proporcionados al LLM al dividir un conjunto de datos en fragmentos. Durante el entrenamiento, los aceleradores (por ejemplo, las GPU) realizan tareas de cálculo hasta llegar a diferentes bloques (por ejemplo, Multi-Head Attention o Feed-forward Neural Network), lo que desencadena nuevas operaciones de comunicación. Suponemos que los aceleradores tienen capacidades computacionales idénticas, procesan la misma cantidad de datos a la misma velocidad e inician su siguiente operación de comunicación simultáneamente.

Nuestro modelo también distingue entre paralelismo de datos (DP) y paralelismo de modelos (MP), tal y como se describe en la Sección II-D. Suponemos que el modelo LLM inicial ya está almacenado en la memoria de los aceleradores, lo que elimina la necesidad de intercambiar mensajes con recursos de memoria externos (por ejemplo, discos NVMe). En su lugar, toda la comunicación se produce entre los aceleradores para compartir información durante las fases de avance y retroceso en MP o para actualizar los gradientes en DP. Suponemos cinco configuraciones de patrones de tráfico que pueden generarse en la red durante el proceso de entrenamiento de LLM: $C1$, $C2$, $C3$, $C4$ y $C5$. De $C1$ a $C4$, modelamos un LLM masivo que requiere MP para distribuir el modelo entre múltiples aceleradores. La diferencia clave entre estos patrones de tráfico es la forma predominante de MP: paralelismo tensorial (TP) o paralelismo de etapas (PP).

$C1$ asume que se utiliza ampliamente el TP, es decir, que la mayoría de las comunicaciones en bloque siguen un patrón TP (por ejemplo, operaciones *All-*

Reduce y *AllGather*). Este escenario genera un tráfico significativo debido a la necesidad de actualizar los gradientes durante la propagación hacia adelante y hacia atrás entre dispositivos (véase la Sección II-D). Como resultado, este patrón de tráfico genera el 20 % de la comunicación a la red inter-nodo y el 80 % a la red intra-nodo. A continuación, $C2$ y $C3$ también emplean MP; sin embargo, el tamaño de los datos transferidos entre dispositivos es menor, ya que estas configuraciones se basan más en PP que en TP. Esto reduce el tráfico inter-nodo, ya que PP utiliza principalmente operaciones punto a punto. Concretamente, estos patrones de tráfico generan el 15 % y el 10 % del tráfico a la red inter-nodo, mientras que el 85 % y el 90 % permanecen dentro de la red intra-nodo, respectivamente. $C4$ utiliza MP sin TP, basándose exclusivamente en PP. En este caso, la comunicación inter-nodo se minimiza (por ejemplo, 5 %), ya que se requieren menos actualizaciones entre etapas. En consecuencia, la comunicación de la red intra-nodo es del 95 %.

Por el contrario, $C5$ no implementa MP. Suponemos que el LLM no es lo suficientemente grande como para requerir múltiples aceleradores. Dado que cabe en un solo acelerador, solo se utiliza DP para acelerar el entrenamiento y no se necesita comunicación inter-nodo. Como resultado, el 100 % de la comunicación se produce dentro del nodo.

IV. Experimentos y resultados

En esta sección se describen los experimentos realizados para validar nuestro modelo de simulación en comparación con un entorno de clúster real, y para evaluar los resultados de las simulaciones de escalabilidad horizontal y vertical utilizando configuraciones de red y patrones de comunicación realistas.

A. Validación del modelo de simulación

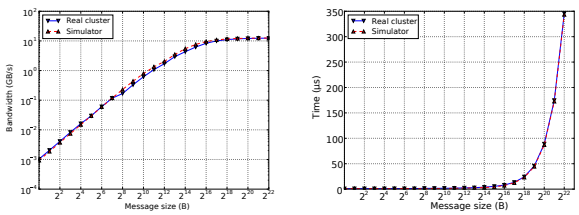
Configuramos el modelo para reproducir la red intra-nodo PCIe según la arquitectura descrita en la Sección III-A, desplegada en el clúster CELLIA. Utilizamos las pruebas InfiniBand [16], que acceden al hardware mediante *verbs*, midiendo ancho de banda y latencia con `ib_write` para mensajes entre 128 B y 4 MiB generados en espacio de usuario. Los mensajes se fragmentan en paquetes intra-nodo de 128 B (MPS PCIe). La Tabla I muestra los resultados experimentales obtenidos en dos nodos del clúster.

El ancho de banda máximo observado es 12.1 GiB/s, cercano al límite teórico de 12.12 GB/s[2], alcanzado a partir de 128 KiB. El incremento de latencia entre 128 B y 128 KiB se debe al overhead de cabeceras (60 B) al fragmentar en TLP. Para tamaños mayores, la latencia crece significativamente al saturarse el enlace inter-nodo (12.1 GB/s), generando acumulación de mensajes en buffers. Esto evidencia que la red inter-nodo puede convertirse en cuello de botella frente a la capacidad intra-nodo (16 GB/s PCIe).

[2]Enlaces InfiniBand EDR con codificación 64b/66b.

Tabla I: Resultados de ancho de banda y lantencia de las pruebas de rendimiento en el clúster CELLIA.

	Ancho de banda (GiB/s)	Latencia (µs)
Mensaje	ib_write	ib_write
128 B	0.44	1.12
256 B	0.87	1.56
512 B	1.75	1.58
1 KiB	3.30	1.70
2 KiB	7.35	1.95
4 KiB	11.02	2.46
8 KiB	11.58	2.84
16 KiB	11.53	3.88
32 KiB	11.60	5.41
64 KiB	11.62	8.06
128 KiB	11.90	13.39
256 KiB	11.92	24.27
512 KiB	11.93	45.73
1 MiB	11.93	88.95
2 MiB	11.93	174.65
4 MiB	11.86	345.97



(a) `ib_write` bandwidth. (b) `ib_write` latency.

Fig. 3: Resultados de validación del modelo de red intra-nodo.

Para validar el modelo, replicamos `ib_write` en el simulador con los mismos parámetros PCIe. La Figura 3 compara ancho de banda y latencia simulados y reales, mostrando resultados prácticamente idénticos y misma tendencia. El modelo fragmenta transacciones en TLP, las transmite vía PCIe a la NIC, que agrega en MTU de 4 KiB para InfiniBand; en destino, los paquetes se descomponen nuevamente en TLP y se entregan a la aplicación.

El modelo intra-nodo es configurable para emular PCIe u otras tecnologías. En las siguientes secciones ampliamos la evaluación a escenarios de escalabilidad vertical y horizontal con configuraciones intra- e inter-nodo genéricas.

B. Configuración de experimentos

En el simulador, configuramos la arquitectura de los nodos y las redes intra e inter-nodo de forma similar a los sistemas para HPC e IA modernos con nodos heterogéneos. En cada nodo suponemos hasta 8 aceleradores (por ejemplo, GPU) interconectados mediante una red intra-nodo de alta velocidad. Cada acelerador dispone de una única NIC para generar tráfico hacia la red intra-nodo. Además, una NIC conecta la red intra-nodo con la red inter-nodo. Consideramos tres configuraciones de red intra-nodo con diferentes velocidades en las NIC de los aceleradores:

- Configuración #1 - 128 Gbps/acelerador. Cada

NIC puede generar hasta 128 Gbps, con un ancho de banda agregado intra-nodo entre 128 (1 acelerador) y 1 024 Gbps (8 aceleradores).
- Configuración #2 - 256 Gbps/acelerador. Cada NIC puede generar hasta 256 Gbps, con un agregado entre 256 (1 acelerador) y 2 048 Gbps (8 aceleradores).
- Configuración #3 - 512 Gbps/acelerador. Cada NIC puede generar hasta 512 Gbps, con un agregado entre 512 (1 acelerador) y 4 096 Gbps (8 aceleradores).

Estas velocidades son compatibles con tecnologías intra-nodo como PCIe o Infinity Fabric [17]. El ancho de banda agregado intra-nodo puede superar ampliamente el de la NIC inter-nodo, que asumimos de 400 Gbps (50 GB/s) según tecnologías comerciales como InfiniBand o Gigabit Ethernet. Esto evidencia que la NIC del nodo puede convertirse en cuello de botella si todos los dispositivos intentan comunicarse simultáneamente con otros nodos.

Para evaluar el impacto del número de aceleradores, consideramos 1, 2, 4 u 8 aceleradores por nodo conectados a un único switch intra-nodo, también enlazado con la NIC del nodo. El switch dispone de búferes de entrada de 128 KiB (arquitectura IQ), conmutación virtual cut-through (VCT) y control de flujo por créditos. El arbitraje se realiza mediante iSlip [18], permitiendo seleccionar solicitudes según Round-Robin (RR) o antigüedad. El número de puertos depende del número de aceleradores, y el ancho de banda de cada puerto coincide con el de la NIC del acelerador.

La red inter-nodo se configura de forma análoga, con switches IQ de 128 KiB, VCT, control por créditos y planificación iSlip (RR o basada en edad). Seleccionamos una topología Real-Life Fat-Tree (RLFT) [19], común en sistemas para HPC e IA. Utilizamos el enrutamiento determinista D-mod-K [20], que equilibra eficientemente las rutas y ofrece rendimiento comparable al adaptativo [21]. La latencia del primer flit en la red inter-nodo se fija en 25 ns/metro.

Como métricas de rendimiento modelamos el *throughput intra-nodo*, que mide la tasa media de datos intercambiados entre aceleradores dentro del nodo, y el *throughput inter-nodo*, correspondiente al tráfico en la red inter-nodo. En latencia, descomponemos el retardo total del mensaje (antes de paquetización) en varios componentes. La Figura 4 muestra las siete métricas consideradas.

La figura ilustra un flujo inter-nodo (rojo) desde Acc_0 de $Server_1$ hasta Acc_N de $Server_M$, y un flujo intra-nodo (verde) dentro de $Server_1$. Se miden la *Source Accelerator Latency*, desde la creación hasta la transmisión del paquete; la *Source Intra-node Latency*, hasta la NIC; la *Source NIC Latency*, que incluye la agregación en MTU inter-nodo; sus equivalentes en destino; y la *Inter-node Latency*, correspondiente al tránsito por la red inter-nodo.

Respecto a los patrones de tráfico, evaluamos el impacto de la comunicación intra e inter-nodo utilizando el modelo descrito en la Sección III-C, basado en entrenamiento de LLM y combinaciones de DP, TP y PP en cinco patrones: $C1$–$C5$. Estos difieren en la proporción de tráfico dirigido a la red intra e inter-nodo. No pretendemos modelar un LLM específico ni determinar configuraciones óptimas, lo cual queda fuera del alcance de este trabajo; herramientas como Calculon [22] pueden abordar ese análisis.

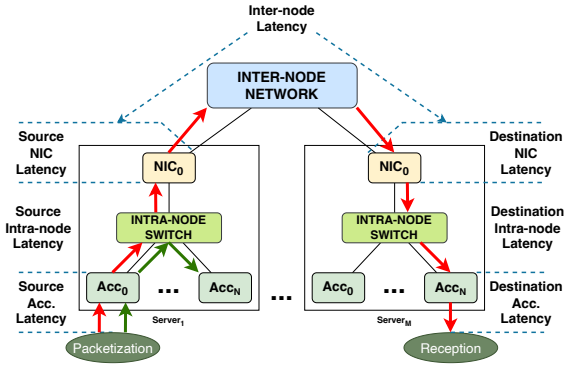


Fig. 4: Métricas de latencia experimentadas por un flujo que atraviesa las redes intra-nodo (rojo) e inter-nodo (verde).

Las NIC de los aceleradores generan mensajes de 4KiB a cargas entre 0 y 100 % del ancho de banda máximo (128, 256 y 512 Gbps según la configuración). Simulamos 10 niveles de carga por patrón durante 2,5 ms de tiempo simulado. En tráfico intra-nodo, el destino se selecciona aleatoriamente entre aceleradores del mismo nodo; en inter-nodo, entre nodos distintos. Los mensajes se fragmentan en paquetes de 148 B en la red intra-nodo. Para destinos remotos, se agregan en MTUs de 4KiB en la NIC origen, se envían por la red inter-nodo y se vuelven a fragmentar en 148 B en el nodo destino. Esta paquetización introduce una sobrecarga significativa que limita el rendimiento, como se analiza en la siguiente sección.

C. Simulación de escalabilidad vertical

A continuación, evaluaremos los resultados del experimento a medida que aumenta el número de aceleradores por nodo, manteniendo el número de nodos en el sistema. Hemos configurado un sistema de 32 nodos interconectados mediante una topología RLFT de dos etapas que comprende doce switches de ocho puertos. Aumentamos el número de aceleradores por nodo de servidor (1, 2, 4 y 8), de modo que el número total en el sistema oscila entre 32 y 256. Configuramos el resto de los parámetros de red inter-nodo e intra-nodo como se describe en la Sección IV-B. También hemos configurado diferentes velocidades para las NIC de los aceleradores (por ejemplo, 128, 256 y 512 Gbps). La Tabla II resume las configuraciones intra-nodo.

C.1 Análisis de la velocidad de la red intra-nodo

En esta sección se analiza el rendimiento de la red intra-nodo e inter-nodo cuando la red intra-nodo se amplía en términos del número de aceleradores por nodo y la velocidad del enlace intra-nodo. La Fi-

Tabla II: Configuraciones de escalabilidad vertical.

#	Velocidad intra-nodo	Nº de aceleradores
1	128 Gbps/acelerador	32, 64, 128, 256
2	256 Gbps/acelerador	32, 64, 128, 256
3	512 Gbps/acelerador	32, 64, 128, 256

gura 5 muestra el rendimiento de la red intra-nodo e inter-nodo de una topología RLFT de 32 nodos, cuando se generan patrones de tráfico C1 a C5. Suponemos que el número de aceleradores por nodo es 1, 2, 4 u 8, de modo que el número total de aceleradores es 32, 64, 128 y 256 para todo el sistema. También suponemos que cada acelerador puede generar hasta 128 Gbps de tráfico a la red intra-nodo.

La Figura 5a muestra los resultados de rendimiento (es decir, la cantidad de información recibida por acelerador) con un solo acelerador por nodo (es decir, 32 aceleradores en total y 128 Gbps por acelerador). En este caso, no es posible la comunicación intra-nodo entre aceleradores dentro de un solo nodo. Por ejemplo, para C1 hay un 20 % de tráfico inter-nodo, es decir, cada acelerador genera $128 \times 0{,}2 = 25{,}6$ Gbps, lo que significa que la tasa de generación total en la red puede ser de hasta $25{,}6 \times 32$ aceleradores= 819,2 Gbps. Por lo tanto, el rendimiento suponiendo que no hay saturación a la carga máxima (en este caso, $25,6$ Gbps por acelerador) es de $819,2$ Gbps. Tenga en cuenta que los nodos de un solo acelerador ya no son prácticos y que los nodos de múltiples aceleradores se han vuelto esenciales. Por ejemplo, una de las razones es el uso cada vez más popular de los LLM [23] y sus crecientes requisitos de memoria, donde el paralelismo tensorial (TP) es crucial para dividir los modelos entre múltiples aceleradores. El TP requiere una comunicación intensiva, lo que exige intercambios frecuentes de mensajes para las operaciones *All-Reduce*.

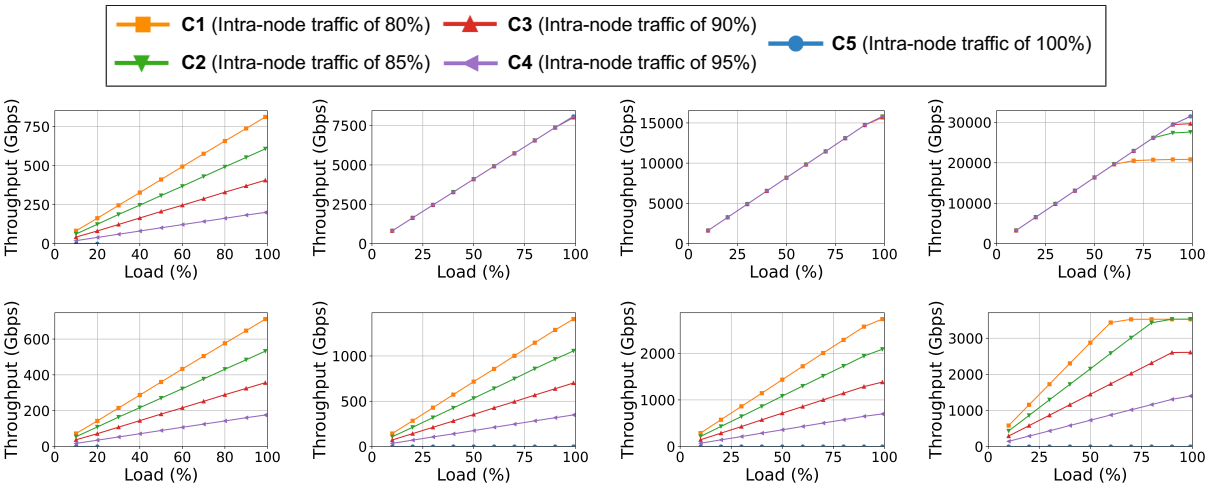
Aumentar el número de aceleradores por nodo a 2 o 4 también incrementa el rendimiento de la red. Las Figuras 5b y 5c muestran que no hay saturación para los nodos con 2 y 4 aceleradores, independientemente del patrón de tráfico, alcanzando el ancho de banda máximo de 8 192 Gbps y 16 384 Gbps, respectivamente. Este rendimiento se consigue con 64 y 128 aceleradores que generan tráfico a toda velocidad (es decir, 128 Gbps) sin ningún cuello de botella en la red. El enlace entre la red intra-nodo y la NIC (véase la Figura 4) permanece sin saturarse. Por lo tanto, las redes intra e inter-nodo alcanzan el máximo rendimiento para todas las configuraciones de patrones de tráfico (C1-C4), incluso para C5, que no implica tráfico inter-nodo.

La Figura 5d muestra los resultados con 8 aceleradores por nodo. Para el patrón de tráfico C5 (es decir, 100 % de tráfico intra-nodo), la red inter-nodo no se utiliza, por lo que el sistema alcanza su máximo rendimiento posible, que es 128 Gbps/acelerador × 8 aceleradores/nodo × 32 nodos = 32 768 Gbps. En las configuraciones de tráfico C4 y C3 (es decir, 95 % y 90 % de tráfico intra-nodo, respectivamente) no se satura el enlace NIC. Teniendo en cuenta que el tráfico máximo que un nodo puede inyectar en la red inter-nodo es de 128 Gbps/acelerador × 8 aceleradores = 1 024 Gbps, cada NIC solo recibe paquetes a una velocidad de 51,2 Gbps para el patrón de tráfico C4 (es decir, el 5 % del tráfico inter-nodo) y 102,4 Gbps para C3 (es decir, el 10 % del tráfico inter-nodo), respectivamente. Por lo tanto, el enlace que conecta el switch intra-nodo a la NIC del nodo, cuyo ancho de banda máximo es de 128 Gbps, puede asumir la carga intra-nodo que los 8 aceleradores envían a la red inter-nodo sin convertirse en un cuello de botella.

Por el contrario, los patrones de tráfico C2 y C1 (85 % y 80 % del tráfico intra-nodo, respectivamente) superan la capacidad máxima del enlace de red intra-nodo (es decir, 128 Gbps), ya que 8 aceleradores por nodo generan una carga de tráfico de 153,6 Gbps para C2 (es decir, el 15 % del tráfico inter-nodo) y 204,8 Gbps para C1 (es decir, el 20 % del tráfico inter-nodo), respectivamente. En estos escenarios, el enlace intra-nodo se convierte en un cuello de botella. Al aumentar el número de aceleradores por nodo, es esencial tener en cuenta el ancho de banda del enlace intra-nodo y la cantidad de tráfico que los aceleradores pueden generar en la red inter-nodo, ya que un tráfico excesivo enrutado a través de la NIC (ya sea entrante o saliente) puede degradar el rendimiento general del sistema. Ten en cuenta que, debido al cuello de botella mencionado, la red inter-nodo no puede alcanzar su rendimiento máximo teórico de 400 Gbps × 32 nodos = 12 800 Gbps, ya que la tasa de inyección de tráfico de la red intra-nodo está limitada por su ancho de banda máximo de enlace, es decir, 128 Gbps. Por lo tanto, para enlaces intra-nodo de 128 Gbps, el número adecuado de aceleradores por nodo es 1, 2 o 4, mientras que 8 aceleradores saturan la red intra-nodo cuando hay un pequeño porcentaje de tráfico generado hacia la red inter-nodo.

Además del rendimiento, hemos analizado la latencia de la red de los paquetes inter-nodo y los componentes que conforman las mediciones de latencia. La Figura 6 muestra los resultados de latencia para una red RLFT de 32 nodos y 8 aceleradores por nodo y patrones de tráfico C4 y C5, con la latencia de los paquetes dividida en sus distintos componentes (explicados posteriormente en la Sección IV-B).

Para C5, podemos observar que, las latencias son despreciables en comparación con las de C4, ya que no existe contención cuando únicamente se genera comunicación intra-nodo. En cuanto a C4, podemos ver que la latencia aumenta significativamente cuando la carga de tráfico alcanza el 70 %. Tras la saturación, varios componentes de latencia consumen la mayor parte del tiempo (por ejemplo, la red intra-nodo de origen, la NIC de origen, la red inter-nodo y la NIC de destino). Sin embargo, la causa principal de la saturación se produce en la NIC de destino, donde los paquetes inter-nodo se fragmentan en paquetes intra-nodo. Este proceso requiere tiempo y acumula tráfico entrante en las NICs de destino, lo



(a) 1 Acelerador/Nodo. (b) 2 Aceleradores/Nodo. (c) 4 Aceleradores/Nodo. (d) 8 Aceleradores/Nodo.

Fig. 5: Rendimiento de la red en función de la carga de tráfico (%) en una topología RLFT de 32 nodos (configuración de red #1 en la Tabla II). El rendimiento total de la red se muestra en las figuras superiores, mientras que el rendimiento inter-nodo se muestra en las inferiores.

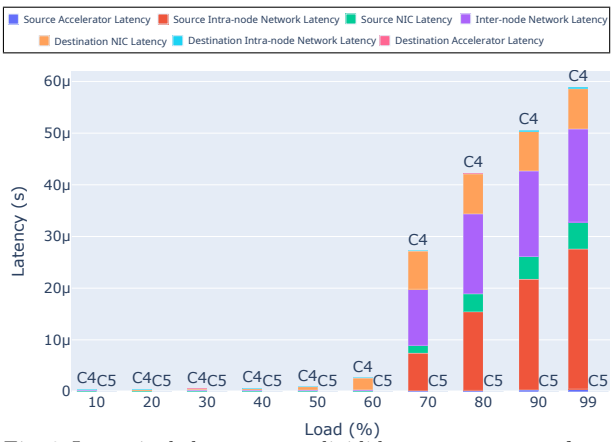


Fig. 6: Latencia de los paquetes dividida en componentes frente a la carga de tráfico para los patrones C4 y C5.

que genera *backpressure* hacia la red inter-nodo que se propaga hasta las NICs de los nodos origen y su red intra-nodo.

La Figura 7 muestra los resultados de rendimiento para la configuración de red #2 (véase la Tabla II), cuando la velocidad del enlace intra-nodo es de 256 Gbps. En este escenario, el punto de saturación se alcanza con solo 4 aceleradores y el patrón de tráfico C2 (es decir, el 15 % del tráfico inter-nodo). Teniendo en cuenta el ancho de banda máximo que pueden generar todos los aceleradores de un nodo, es decir, 256 Gbps/acelerador × 8 aceleradores =2 048 Gbps. Tenga en cuenta que el cuello de botella entre la red intra-nodo y la NIC sigue presente. De hecho, el rendimiento del sistema se degrada más rápidamente en comparación con la configuración de red #1.

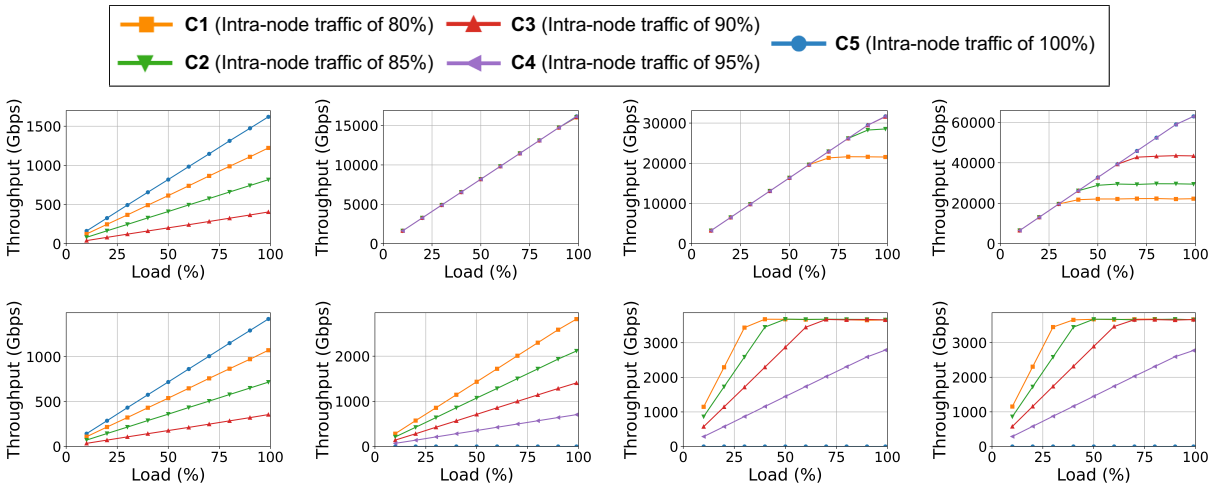
La Figura 8 muestra los resultados de rendimiento para la configuración de red #3 (véase la Tabla II). En este escenario, el enlace entre la red intra-nodo y la NIC puede funcionar a 512 Gbps, y los enlaces de red inter-nodo funcionan a 400 Gbps, lo que no es suficiente para aliviar el cuello de botella entre el switch inter-nodo y la NIC del nodo cuando los aceleradores intercambian una pequeña cantidad de tráfico. Ten en cuenta que la red se satura con solo 2 aceleradores por nodo que generan el 60 % de la carga de tráfico y el patrón de tráfico C1.

Estos resultados experimentales sugieren que el simple hecho de aumentar el ancho de banda de la red intra-nodo no garantiza una mejora del rendimiento. Si bien la presencia de múltiples aceleradores por nodo es beneficiosa para minimizar la latencia durante el paralelismo tensorial (TP), también aumenta el volumen de tráfico inter-nodo. Si este tráfico supera la capacidad de la red inter-nodo o de la NIC, puede producirse una degradación del rendimiento incluso antes, cuando el ancho de banda intra-nodo es alto.

C.2 Conclusiones sobre la escalabilidad vertical

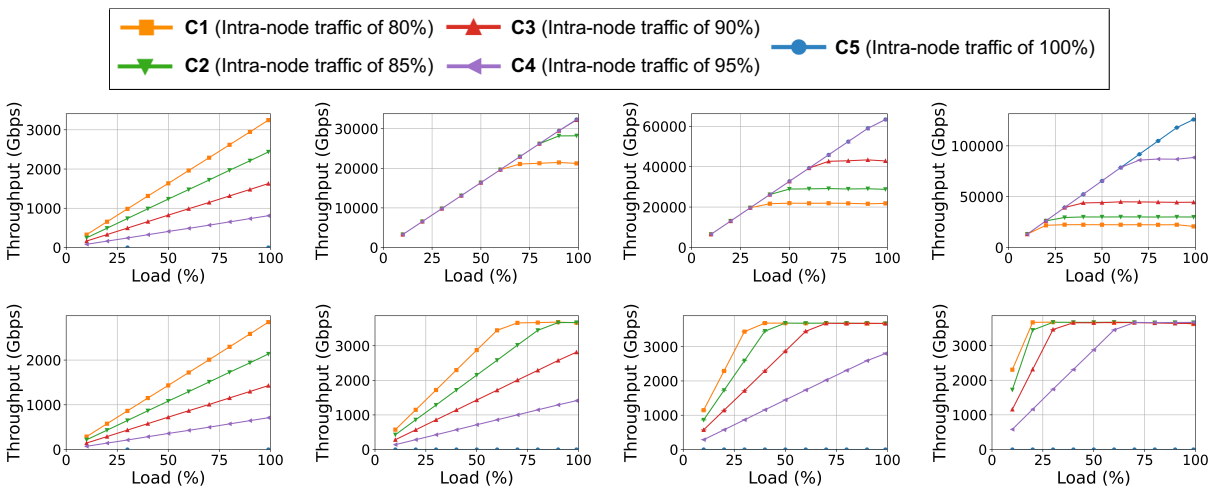
A medida que los LLM siguen creciendo, cada vez es más necesario distribuir los parámetros del modelo entre varios aceleradores, lo que aumenta el número de aceleradores por nodo. Hemos analizado las implicaciones de ampliar el número de aceleradores dentro de un nodo. Hemos observado que aumentar el número de aceleradores o la velocidad de la red intra-nodo puede provocar cuellos de botella en la NIC de destino, ya que la red intra-nodo suele utilizar tamaños de carga útil más pequeños que la red inter-nodo, lo que da lugar a una mayor sobrecarga debido a la conversión.

Por lo tanto, la escalabilidad vertical de los aceleradores dentro de los nodos es eficaz cuando la mayor parte de la comunicación se produce dentro del mismo nodo, como en configuraciones con paralelismo tensorial (TP) intensivo dentro de los nodos y solo paralelismo de etapa (PP) o paralelismo de datos (DP) inter-nodo. Sin embargo, en escenarios en los que la memoria disponible en un solo nodo es insuficiente para almacenar un modelo completo y se debe aplicar el TP en varios nodos, la comunicación inter-nodo se vuelve más frecuente y crítica. En tales casos, el rendimiento del sistema se degrada significativamente y las ventajas de la escalabilidad vertical disminuyen.



(a) 1 Acelerador/Nodo. (b) 2 Aceleradores/Nodo. (c) 4 Aceleradores/Nodo. (d) 8 Aceleradores/Nodo.

Fig. 7: Rendimiento de la red en función de la carga de tráfico (%) en una topología RLFT de 32 nodos (configuración de red #2 en la Tabla II). El rendimiento total de la red se muestra en las figuras superiores, mientras que el rendimiento inter-nodo se muestra en las inferiores.



(a) 1 Acelerador/Nodo. (b) 2 Aceleradores/Nodo. (c) 4 Aceleradores/Nodo. (d) 8 Aceleradores/Nodo.

Fig. 8: Rendimiento de la red en función de la carga de tráfico (%) en una topología RLFT de 32 nodos (configuración de red #3 de la Tabla II). El rendimiento total de la red se muestra en las figuras superiores, mientras que el rendimiento inter-nodo se muestra en las inferiores.

D. Simulación de escalabilidad horizontal

En esta sección, evaluamos el rendimiento del sistema a medida que aumenta el número de nodos, lo que conlleva un aumento del número total de aceleradores en el sistema. La Tabla III resume las configuraciones de red utilizadas en esta evaluación de escalabilidad horizontal. Cada nodo está equipado con 8 aceleradores. Evaluamos dos configuraciones de red intra-nodo de alta velocidad para valorar su impacto en el rendimiento general. El resto de los parámetros de red son los mismos que en los experimentos anteriores.

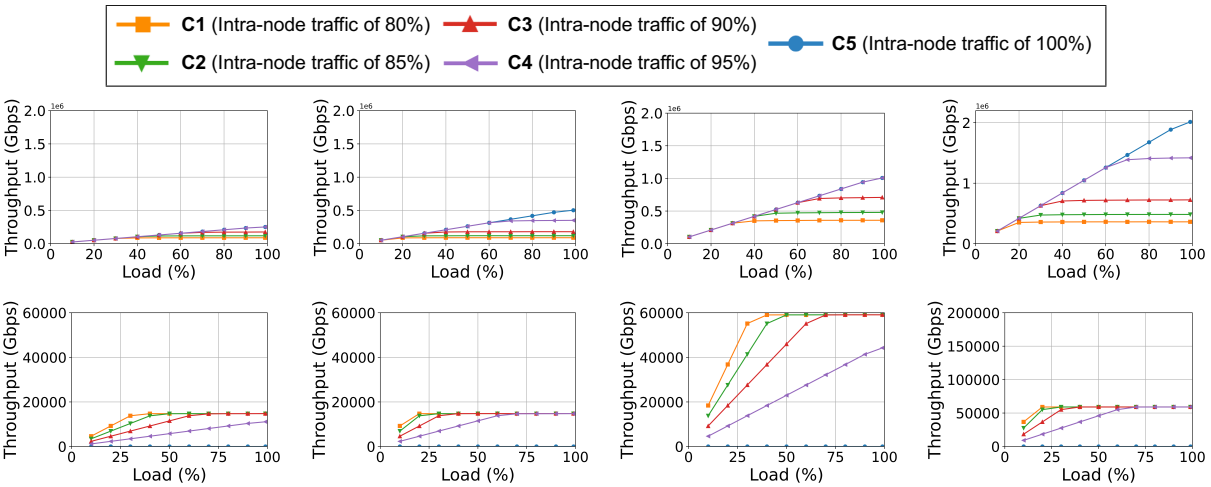
Tabla III: Configuraciones de red de escalabilidad horizontal.

#	Nodos	Sw. inter-nodo	Aceleradores	Vel. intra-nodo
1	128	24	1 204	256 Gbps
2				512 Gbps
3	512	48	4 096	256 Gbps
4				512 Gbps

La Figura 9 muestra el rendimiento de la red en las diferentes configuraciones descritas en la Tabla III. En comparación con los resultados de la Sección IV-C, observamos que el comportamiento de la red sigue tendencias similares, con una degradación del rendimiento que se produce en niveles comparables de intensidad de tráfico.

Por ejemplo, en la configuración de red #2, que admite un rendimiento máximo teórico de 524 288 Gbps, el sistema solo alcanza: Rendimiento (Gbps) $= \frac{16\,384}{1{,}14\times 20} \times 128 = 91\,980$ Gbps con el patrón de tráfico C1. Esto corresponde solo al 17,54 % de la capacidad máxima de la red. De manera similar, en la configuración #4, que tiene un rendimiento máximo teórico de 2 097 152 Gbps, el rendimiento observado con tráfico C1 es de 367 921 Gbps, lo que supone, de nuevo, aproximadamente el 17,54 % de la capacidad máxima.

Estos resultados refuerzan la observación de que la escalabilidad horizontal del sistema puede verse críticamente limitado por los patrones de tráfico inter-nodo. Cuando una parte significativa del tráfico se dirige entre diferentes nodos, el enlace entre la red intra-nodo y la NIC puede convertirse en un grave



(a) Configuration #1. (b) Configuration #2. (c) Configuration #3. (d) Configuration #4.

Fig. 9: Rendimiento de la red en función de la carga de tráfico (%) (configuraciones de red de la Tabla II). El rendimiento intranodo se muestra en la sección superior, mientras que el rendimiento inter-nodo se muestra en la sección inferior.

cuello de botella. Además, la sobrecarga de paquetización causada por los diferentes tamaños de carga útil entre las comunicaciones intra-nodo e inter-nodo puede degradar aún más el rendimiento. A medida que aumenta el número de nodos, estas limitaciones persisten, por lo que es esencial tener en cuenta los patrones de tráfico y los límites del hardware al escalar sistemas distribuidos para cargas de trabajo a gran escala, como el entrenamiento y la inferencia de LLM.

V. Conclusiones y trabajo futuro

Este artículo ha analizado el impacto de la interferencia entre las operaciones de comunicación intra e inter-nodo generadas por aplicaciones emergentes (por ejemplo, IA generativa o LLM) en sistemas para HPC e IA. Para ayudar en esta investigación, hemos desarrollado un modelo detallado de simulación de red intra e inter-nodo a nivel de paquetes, validado con una infraestructura de clúster real utilizando microbenchmarks estándar. Hemos analizado exhaustivamente los efectos de la latencia y la sobrecarga de la conversión de paquetes inter- e intra-nodo cuando se utilizan diferentes tamaños de carga útil. Además, hemos realizado experimentos de simulación de escalabilidad horizontal modelando redes intra e inter-nodo de hasta 4 096 aceleradores. También hemos modelado patrones de comunicación realistas basados en el modelo de programación LLM, utilizando diferentes tipos de paralelismo (paralelismo de datos, de etapa y tensorial), que se han utilizado para alimentar nuestra herramienta de simulación. Nuestro análisis ha identificado un cuello de botella crítico en la interfaz entre la red intra-nodo y el enlace NIC, que afecta significativamente al rendimiento de las redes intra- e inter-nodo y reduce el rendimiento general del sistema. También hemos observado que aumentar o disminuir el número de nodos o aumentar la velocidad de los enlaces intra-nodo no altera el impacto de este cuello de botella, ya que las limitaciones de la red intra-nodo siguen siendo el factor dominante. Nuestra simulación y los hallazgos relacionados con la dinámica de la comunicación intra- e inter-nodo proporcionan información valiosa para mejorar el diseño de redes de interconexión intra- e inter-nodo a gran escala.

Agradecimientos

El presente trabajo ha sido financiado mediante el proyecto del Ministerio de Ciencia y Universidades de España MCIN/AEI/10.13039/501100011033, la Unión Europea (NextGenertionEU/PRTR) en el marco del proyecto TED2021-130233B-C31, la Junta de Comunidades de Castilla-La Mancha y los fondos FEDER en el marco del proyecto SBPLY/21/180501/000248, las subvenciones PERTE-Chip (Cátedra UCLM, TSI-069100-2023-0014) financiadas por el Ministerio de Transformación Digital y Servicio Público de España, y la Universidad de Castilla-La Mancha en el marco del proyecto 2025-GRIN-38312.

Referencias

[1] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Anand Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia, "Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM," Aug. 2021, arXiv:2104.04473 [cs].

[2] Daniele De Sensi, Lorenzo Pichetti, Flavio Vella, Tiziano De Matteis, Zebin Ren, Luigi Fusco, Matteo Turisini, Daniele Cesarini, Kurt Lust, Animesh Trivedi, Duncan Roweth, Filippo Spiga, Salvatore Di Girolamo, and Torsten Hoefler, "Exploring gpu-to-gpu communication: Insights into supercomputer interconnects," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2024, pp. 1–15.

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, aug 2011.

[4] Saptarshi Bhowmik, Nikhil Jain, Xin Yuan, and Abhinav Bhatele, "A simulation study of hardware parameters for future gpu-based hpc platforms," in *2021 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 2021, pp. 1–10.

[5] Nikhil Jain, Abhinav Bhatele, Sam White, Todd Gamblin, and Laxmikant V. Kale, "Evaluating hpc networks

via simulation of parallel workloads," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2016, SC '16, IEEE Press.

[6] "NVIDIA H100 Tensor Core GPU Architecture Overview," Dec. 2024, [Online; accessed 18. Dec. 2024].

[7] "NVIDIA Grace Hopper Superchip Architecture Whitepaper," Dec. 2024, [Online; accessed 18. Dec. 2024].

[8] "Intel® Gaudi® 3 AI Accelerator White Paper," Dec. 2024, [Online; accessed 18. Dec. 2024].

[9] Torsten Hoefler, Ariel Hendel, and Duncan Roweth, "The convergence of hyperscale data center and high-performance computing networks," *Computer*, vol. 55, no. 7, pp. 29–37, 2022.

[10] Andrea Biagioni and et al., "Red-sea: Network solution for exascale architectures," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, 2022, pp. 712–719.

[11] Shekhar Borkar and Andrew A. Chien, "The future of microprocessors," *Communications of the ACM*, vol. 62, no. 5, pp. 44–52, 2019.

[12] Christopher J Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl, "Measuring the effects of data parallelism on neural network training," *arXiv preprint arXiv:1811.03600*, 2018.

[13] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint:1909.08053*, 2019.

[14] Yanping Huang, Orhan Firat, HyoukJoong Lee, Yonghui Wu, Youlong Cheng, Mia XuChen, Jiquan Ngiam, Ankur Bapna, Dehao Chen, Quoc V Le, and Zhifeng Chen, "Gpipe: Easy scaling with micro-batch pipeline parallelism," *arXiv preprint arXiv:1811.06965*, 2019.

[15] Pedro Yébenes, Jesus Escudero-Sahuquillo, Pedro J. García, Francisco J. Alfaro, and Francisco J. Quiles, "Providing differentiated services, congestion management, and deadlock freedom in dragonfly networks," in *2016 2nd IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, 2016, pp. 33–40.

[16] NVIDIA, "Perftest Package," Mar. 2023, [Online; accessed 11. Mar. 2024].

[17] Gabin Schieffer, Ruimin Shi, Stefano Markidis, Andreas Herten, Jennifer Faj, and Ivy Peng, "Understanding data movement in amd multi-gpu systems with infinity fabric," 2024.

[18] N. McKeown, "The islip scheduling algorithm for input-queued switches," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201, 1999.

[19] Hridoy Jyoti Mahanta, Abhijit Biswas, and Anwar Hussain, "An architecture based routing for heterogeneous fat tree network on chip," in *2015 International Symposium on Advanced Computing and Communication (ISACC)*, 2015, pp. 341–345.

[20] Eitan Zahavi, "Fat-tree routing and node ordering providing contention free traffic for mpi global collectives," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1423–1432, 2012, Communication Architectures for Scalable Systems.

[21] C. Gomez, F. Gilabert, M.E. Gomez, P. Lopez, and J. Duato, "Deterministic versus adaptive routing in fat-trees," in *2007 IEEE International Parallel and Distributed Processing Symposium*, 2007, pp. 1–8.

[22] Mikhail Isaev, Nic Mcdonald, Larry Dennison, and Richard Vuduc, "Calculon: a methodology and tool for high-level co-design of systems and large language models," in *ACM Conferences*, pp. 1–14. Association for Computing Machinery, New York, NY, USA, Nov. 2023.

[23] Pablo Villalobos, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, Anson Ho, and Marius Hobbhahn, "Machine learning model sizes and the parameter gap," 2022.

Metodología para la visualización en tiempo real de simulaciones de SimGrid

Manuel Gómez-Plana Rodríguez[1], Félix García Carballeira[2]

Resumen— **Desde comienzos del siglo XX existe un interés por el desarrollo y el diseño de simulaciones, entornos artificiales que permiten explorar casos hipotéticos reduciendo el coste, el tiempo o el riesgo asociado a experimentos reales. A raíz de esto, se han desarrollado una gran cantidad de simuladores digitales que permiten a los investigadores estudiar una diversa y amplia variedad de escenarios. Entre los múltiples simuladores de sistemas informáticos creados a lo largo de los años, SimGrid destaca por su accesibilidad, precisión y versatilidad, permitiendo simular sistemas compuestos por múltiples clúster en un único proceso de un ordenador. A pesar de su popularidad y de todos los atributos que hacen de SimGrid un buen simulador, este no cuenta con funciones relacionadas con la visualización de los datos, limitando así su impacto potencial en el paradigma de la simulación ya que no permite visibilizar el estado de una simulación en tiempo real de forma nativa. Por ello, este trabajo presenta una metodología mediante la cuál se pueden visualizar los datos de una simulación de SimGrid en tiempo real e, incluso, cambiar parámetros de la simulación en caliente.**

Palabras clave— **Simulaciones, SimGrid, Visualización, Tiempo real.**

I. Introducción

La simulación se ha convertido en una herramienta crucial en la ciencia y la ingeniería, pero esto no es algo novedoso. La primera simulación se podría atribuir a William Sealy Gosset, que, bajo el alias "Student", publicó un artículo en 1908 en el que describía la famosa distribución de t de Student [1]. Para poder contar con resultados analíticos completos, Gosset usó una forma primitiva de simulación manual que le permitió explorar la función de densidad de dicha distribución [2].

El desarrollo del primer ordenador eléctrico de uso general, el ENIAC, en 1940, continuó demostrando la utilidad de las simulaciones. En este ordenador se aplicó el famoso método de Monte Carlo para resolver algunos problemas de diseño de la bomba de hidrógeno [2]. A partir de 1950 y con el desarrollo industrial, los ordenadores se fueron convirtiendo progresivamente en herramientas disponibles para todos y, consecuentemente, las técnicas de simulación se fueron desarrollando. Así, en los siguientes años, se publicaron proyectos importantes que avanzaron en el paradigma de la simulación, como:

- **Sistema de Simulación de Propósito General (GPSS)**: desarrollado en la década de 1960, este programa facilitaba el modelado de simulaciones de sistemas de telecomunicación complejos [3].
- **SIMSCRIPT**: creado en 1963 en *FORTRAN*, este programa fue diseñado para ser usado por personas sin conocimiento sobre el uso de ordenadores[4].
- **SIMULA**: SIMULA I extendió ALGOL 60, convirtiéndose en uno de los lenguajes de programación más importantes en el mundo de la simulación [5].

A partir de entonces, han surgido innumerables simuladores que cubren distintos aspectos científicos, como el campo militar, el campo empresarial o el campo social [6]. La informática es otro de estos campos y, en este, existen varios simuladores que permiten a los investigadores explorar diversos escenarios, como el comportamiento de un algoritmo, la manera en que funciona un sistema de ficheros distribuido o incluso la simulación de un sistema distribuido a gran escala. Entre las herramientas de simulación que han surgido en este campo en los últimos años, destaca SimGrid, un *framework* que permite definir simulaciones versátiles [7].

Este *framework* destaca por la precisión con la permite simular sistemas complejos, sumada a la flexibilidad para definir cualquier sistema distribuido que un investigador tenga en mente. Así, este *framework* es usado y citado cada año en múltiples investigaciones, artículos científicos publicados en conferencias y revistas, e incluso trabajos de fin de grado.

Sin embargo, a pesar de todas las características que hacen de este simulador una muy buena herramienta, presenta un fallo crucial: SimGrid no ofrece una manera nativa de visualizar los datos de sus simulaciones en tiempo real. Esto provoca que el análisis se deba hacer tras la simulación, recogiendo datos y luego visualizándolos, algo que a menudo es suficiente pero que limita el uso del simulador a simulaciones estáticas. Así, este trabajo pretende ofrecer una metodología de base que permita visualizar los datos de una simulación de SimGrid en tiempo real, así como interactuar con los parámetros de la misma mientras esta se ejecuta.

II. Estado de la cuestión

Antes de proceder a explicar la metodología, es necesario analizar el estado en el que se encuentra la visualización en SimGrid. Un vistazo a la literatura existente muestra que la mayoría de trabajos emplean un análisis de la simulación *off-line*, es decir, una vez que se acaba la simulación. La propia documentación de SimGrid explica una metodología para recoger trazas de la simulación y visualizar estos datos usando R y ggplot2 [8]. Mientras que el acercamiento es válido, este se aleja de la visualización

[1]Estudiante, Universidad Carlos III de Madrid, e-mail: 100472310@alumnos.uc3m.es.

[2]Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: fgcarbal@inf.uc3m.es.

en tiempo real que pretende proponer este artículo.

Del Pozo Puñal et al. [9] proponen un enfoque parecido al de SimGrid, donde los datos son recogidos durante la simulación para ser analizados tras esta. A diferencia de la primera metodología, este trabajo representa los datos en un mapa, lo que demuestra que la representación de la información no tiene por qué reducirse a gráficos y que puede adaptarse a cada caso de uso.

Por último, Sri Vatsavai et al. [10] presentan en su artículo `CGSim`, un *framework* de simulación construido sobre SimGrid y que cuenta con un módulo de visualización de la simulación en tiempo real. Esta visualización se realiza mediante paneles de control, que permiten monitorizar los recursos y el estado de la simulación. Sin embargo, este enfoque difiere de la metodología propuesta en dos aspectos clave: la visualización está inherentemente ligada al *framework* y, por lo tanto, es difícilmente modificable e imposible de separar de la herramienta; y no permite modificar parámetros de la simulación en caliente.

Este problema de visualización y control de la simulación no es algo exclusivo a SimGrid, sino que tiene su propio paradigma, conocido como *computational steering*. Mulder y Van Wijk describen este concepto como el objetivo de las simulaciones interactivas, donde el investigador es capaz de modificar parámetros de la simulación mientras recibe retroalimentación del efecto producido [11]. Este concepto, aunque desarrollado originalmente en el contexto de la dinámica de fluidos computacional y otras disciplinas de alto rendimiento, resulta directamente aplicable a la simulación de sistemas distribuidos.

Más recientemente, la comunidad de HPC ha desarrollado la visualización in situ. La visualización post hoc ha sido el paradigma dominante durante décadas, en la que los códigos de simulación iteran y guardan ficheros en disco para que se analicen posteriormente. Sin embargo, este paradigma se ha visto sometido a una creciente presión debido a la divergencia entre las velocidades de crecimiento de las operaciones de E/S y de cómputo. [12] Herramientas como ParaView, Catalyst y Ascent representan el estado del ámbito: el procesamiento in situ ayuda a mitigar estos cuellos de botella de E/S, permitiendo que los cálculos de simulación y visualización se ejecuten en memoria, evitando la transferencia de datos en bruto a disco.

Así, el enfoque que se propone en los siguientes apartados define una metodología genérica que permite una interacción en tiempo real y bidireccional, es decir, de la simulación hacia el investigador y viceversa.

III. Metodología propuesta

Tras haber definido el objetivo a alcanzar de la metodología, se explica el enfoque desarrollado.

A. Arquitectura

Primero, es necesario definir los componentes que conforman esta metodología. Existen dos componentes principales:

- **Visualizador**: este componente es el encargado de recibir los datos y mostrarlos al usuario.
- **Simulador**: este componente es el encargado de ejecutar la simulación, recoger los datos y enviarlos al visualizador.

Como es natural, existen distintas herramientas que podrían actuar como el visualizador. Se podría crear un servidor en Python, una página web con un *framework* de desarrollo, una aplicación de ordenador... A pesar de esta amplia gama de opciones, el visualizador debe contar con una característica crucial, la capacidad de exponer un *API*. Esta *API* debe definir *endpoints* para que el simulador publique los datos.

Por otro lado, SimGrid actúa como herramienta para el simulador. Para poder realizar la comunicación entre estos dos componentes, se propone una extensión de la *API* de SimGrid con métodos genéricos encargados tanto de enviar datos hacia el visualizador, como de recibir comandos desde el mismo. Las Figuras 1 y 2 muestran visualmente la arquitectura de la metodología.

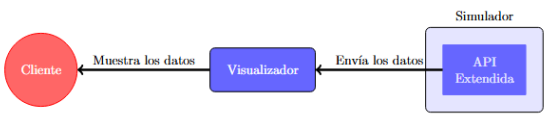


Fig. 1: Envío de datos del simulador.

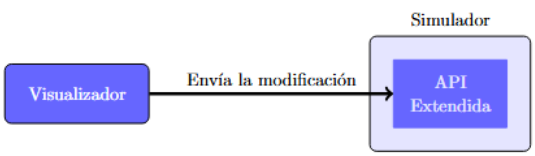


Fig. 2: Envío de órdenes del visualizador.

B. Extensión de la API de SimGrid

Como ya se ha mencionado, la comunicación se realiza de una forma lo suficientemente genérica como para extender la *API* de SimGrid con métodos que se encarguen de enviar los datos hacia el visualizador y recibir órdenes del mismo. Los métodos propuestos para la extensión son los siguientes:

- **Envío de datos hacia el visualizador**: para que el simulador envíe los datos hacia el visualizador, se propone el método `sendSimulationState`. Este método recibe como parámetros:
  - La IP de la máquina donde se encuentra alojado el visualizador.
  - El puerto donde se está ejecutando el visualizador.
  - Los datos a enviar, como un string formateado como un *JSON*.

  Con esto, el método crea un cliente *HTTP* para enviar los datos de la simulación hacia el visua-

lizador. Esto se realiza mediante una petición a la *API* del visualizador, concretamente, al path "/api/updateSimulationState".

- **Recepción de comandos del visualizador**: si se quiere que la comunicación sea bidireccional y que el visualizador envíe comandos para cambiar aspectos o parámetros de la simulación en tiempo real, se propone el método `launchAPI`. Este método recibe como parámetros:
  - El puerto donde se lanzará el servidor que escucha peticiones del visualizador.
  - Una lista de métodos que definen los *endpoints* del simulador.

  Los *endpoints* de la *API* del simulador se definen mediante una lista de objetos de la estructura `APIMethod`, que tiene los siguientes atributos:
  - **path**: define el nombre de la ruta del método de la *API* del simulador.
  - **method**: define el tipo de método del *endpoint* de la *API* del simulador.
  - **handler**: define una función de *callback* que recibe como parámetro una petición y devuelve una respuesta.

Estos dos métodos funcionan de manera genérica para proporcionar una forma de comunicación entre el visualizador y el simulador de manera agnóstica a la simulación. Es decir, estos métodos podrían ser integrados en la *API* de SimGrid para enviar datos de una simulación y recibir comandos exponiendo una *API*.

C. Ejemplo de uso genérico

A continuación, se muestra un ejemplo genérico de una simulación, y cómo podría adaptarse para incluir los métodos explicados en el punto anterior. La función `main` de una simulación podría incluir la función `launchAPI` de la siguiente forma:

Listing 1: Pseudocódigo de la función `main`

```
Creación del motor de la simulación
Registro de las funciones de los actores
Carga de la plataforma
Carga del despliegue
<- Creación de la lista de métodos de la API
<- Llamada a launchAPI
Ejecución de la simulación
```

Este pseudocódigo muestra la ejecución normal de una función `main` de SimGrid, pero incluye dos pasos nuevos que permiten definir una *API* para recibir los comandos mediante peticiones *HTTP*. De la misma forma, un actor de la simulación podría enviar sus datos hacia el visualizador siguiendo el siguiente comportamiento:

Listing 2: Pseudocódigo del envío de datos al visualizador

```
Obtención de datos
<- Formato de los datos a un JSON
<- Llamada a sendSimulationState
```

D. Implementación

Por último, es necesario explicar la implementación de estos métodos y de su integración en el código de la simulación. Es necesario mencionar que toda la comunicación entre el visualizador y el simulador se realiza mediante varias *API REST*.

D.1 Implementación de `sendSimulationState`

Este método es realmente simple, ya que emplea la librería de C++, *httplib*, para crear un cliente *HTTP* que realiza una petición hacia la *API* del visualizador. Esto se realiza mediante el siguiente código de C++:

Listing 3: Implementación del envío de datos

```
void sendSimulationState(
    const std::string& ip,
    uint16_t port,
    const std::string& data) {

    httplib::Client cli(ip, port);

    auto res = cli.Post(
        "/api/updateSimulationState",
        data,
        "application/json"
    );
}
```

D.2 Implementación de `launchAPI`

Este método es algo más complicado que el anterior. Para poder recibir peticiones *HTTP*, la simulación debe exponer un puerto donde se aloje un servidor que esté escuchando las peticiones. Para ello, se usa la librería CROW, que permite definir varios *endpoints* de una *API* y lanzar un servidor que escuche las peticiones que reciba. Este servidor se ejecuta en un hilo para evitar bloquear el hilo principal del programa y, así, permitir a SimGrid ejecutar la simulación. La función se define de la siguiente manera:

Listing 4: Implementación de la recepción de comandos

```
std::thread launchAPI(
    uint16_t port,
    const std::vector<APIMethod>& routes) {

    return std::thread([port, routes]() {

        auto& cors = app.get_middleware<crow::
            CORSHandler>();
        cors.global()
            .origin("*")
            .methods("GET"_method,
                     "POST"_method,
                     "PUT"_method,
                     "PATCH"_method,
                     "DELETE"_method);

        for (const auto& route : routes) {
            std::string fullPath = "/api/" +
                route.path;
            crow::HTTPMethod crowMethod =
                toCrowMethod(route.method);

            app.route_dynamic(fullPath).methods(
                crowMethod)(route.handler);
        }

        app.port(port).multithreaded().run();
    });
}
```

Esta función define, para cada entrada del listado de rutas pasado como parámetro, un *endpoint* de la *API* que llama al método de *callback* asociado. Así, la simulación puede ser adaptada fácilmente a esta metodología, ya que es el programador el encargado de definir los comandos que se envían desde el visualizador, así como sus efectos en la simulación.

Adicionalmente, el método mostrado usa una función adicional, `toCrowMethod`. Esta función simplemente transforma el tipo de método que recibe como parámetro al formato que CROW necesita.

D.3 Actor especial en SimGrid

Para poder implementar correctamente el envío de datos hacia el visualizador, es recomendable crear un actor especial que se encargue de llamar a la función *sendSimulationState*. Principalmente, esto se debe a que la comunicación por *API* bloquea el programa. Por lo tanto, es mejor tener esta comunicación en un actor especial, que no interfiera con el funcionamiento normal de la simulación.

Adicionalmente, desde un punto de vista del diseño de la simulación, es mejor separar las responsabilidades del actor que se comunica con el visualizador del resto de tareas. Esto es algo necesario para simplificar la sincronización con los datos que recibe el servidor del simulador. Por ejemplo, es mucho más limpio y sencillo que el servidor que recibe las peticiones se sincronice con un único actor cuyo único cometido es enviar mensajes al visualizador.

Desde cierto punto de vista, se podría pensar que este actor especial de SimGrid es fácilmente reemplazable con un hilo, igual que el servidor que expone la *API*. Ese es otro enfoque válido, pero si el actor especial forma parte de la simulación, es mejor integrar este actor como un actor de SimGrid desde el punto de vista del diseño.

Por ejemplo, si en el sistema simulado existe un cliente que recibe los datos y los visualiza, como en una aplicación de odometría, es más preciso y sencillo implementar al cliente como un actor de SimGrid que como un programa corriendo en un hilo. Al formar parte de la simulación, los tiempos de ejecución se adecúan a la plataforma de uso, logrando así una simulación mucho más controlable.

A esto se suma que los mecanismos de comunicación que proporciona SimGrid, como los mailboxes, funcionan excepcionalmente bien entre actores de SimGrid, pero pueden ser más difíciles de integrar con hilos externos a la simulación. El modelo de comunicación adaptado a la existencia de este actor especial se muestra en la Figura 3.

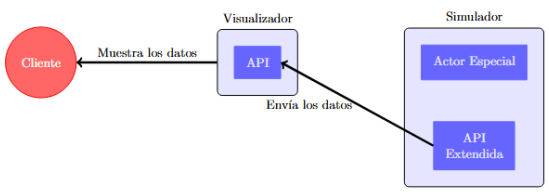


Fig. 3: Envío de datos del simulador con un actor especial.

IV. Validación y Caso de Uso

Para poder validar la metodología creada, se ha ideado un caso de uso donde aplicarla: la monitorización de la posición de los trenes del metro de Madrid. Esta sección describe las herramientas usadas, la descripción de la simulación, la descripción del visualizador y los resultados obtenidos.

A. Herramientas usadas

Como ya se ha mencionado, la simulación está implementada empleando el *framework* SimGrid. Concretamente, se usa la versión 4.1 de la interfaz S4U en C++ [13]. La herramienta con la que se construye el visualizador es Svelte [14], un *framework* de desarrollo web elegido por su facilidad de uso, la familiaridad con la herramienta y la integración del mismo con Leaflet, una biblioteca de JavaScript que proporciona una interfaz para trabajar con mapas.

B. Descripción de la simulación

En la simulación ideada para el experimento, existen los siguientes actores:

- **Trenes**: estos actores se encargan de simular los trenes que recorren las líneas de metro. Estos se comunican con los servidores mediante una simulación del protocolo UDP para enviar su información de odometría, como la velocidad o la posición.
- **Servidores**: estos actores se encargan de recibir los mensajes de los trenes y enviarlos a los *brokers* de *MQTT*, a los cuales se suscriben al comienzo de la simulación.
- ***Brokers* de *MQTT***: estos actores se distribuyen por el mapa para aportar disponibilidad y replicación a la simulación. Estos reciben los datos publicados por los servidores y los reenvían a los suscriptores del *topic* concreto, es decir, a otros *brokers* de *MQTT* y a los clientes.
- **Clientes**: estos actores son los actores especiales mencionados en el apartado anterior. Estos actores se suscriben a los *brokers* de *MQTT* para recibir los datos de odometría de los trenes y los envían al visualizador haciendo una petición a la *API* del mismo.
- **Pacer**: este es un actor único especial usado para sincronizar la simulación. La simulación está discretizada en pasos de 0.25 segundos, es decir, los trenes envían su información cada paso y su localización y velocidad varían según este lapso de tiempo.

Como se ha mencionado, estos actores se comunican para pasar la información desde los trenes hasta los clientes. El modelo jerárquico de la comunicación de los actores se muestra en la Figura 4

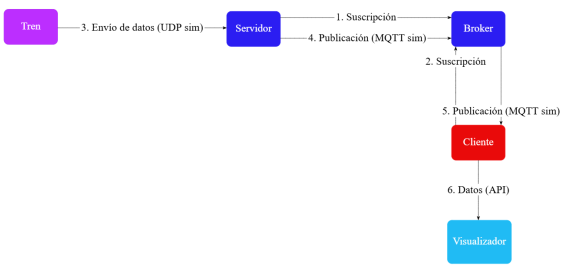


Fig. 4: Modelo de comunicación de los actores.

Por otro lado, antes de iniciar la simulación se ha creado un método para lanzar en la *API* del simulador. Esto está pensado para poder desactivar o activar componentes de la simulación, haciendo como si los actores hubiesen perdido la conexión en la vida real, como si un tren no tuviese acceso a Internet durante un tramo o como si un servidor se cayera. Para poder hacer esto, se define el siguiente método en la *API* del simulador:

- **/api/change_status**: un método de tipo *POST* que cambia el estado del actor especificado como parámetro de la llamada.

Para adaptar la simulación a la desactivación o activación de los actores, los trenes, servidores y clientes están programados para adaptarse a la disponibilidad de los actores. Esto se realiza para simular la tolerancia a fallos del sistema distribuido representado en la simulación. Si los actores se ven desactivados, continuarán con su funcionamiento normal pero no participarán en ningún tipo de comunicación en el sistema.

C. Descripción del visualizador

En este caso de uso, el visualizador debe mostrar un mapa con las estaciones y los tramos de las líneas del metro de Madrid. Con el objetivo de proporcionar un ejemplo sencillo, se ha decidido incluir solamente la información relacionada con la línea 12 del metro de Madrid. Por lo tanto, el mapa incluye las estaciones y los tramos de dicha línea. Así, obteniendo la localización de las estaciones y los tramos del *Portal de Datos Abiertos del Consorcio Regional de Transportes de Madrid*, se crea un mapa de Leaflet, visible en la Figura 5.

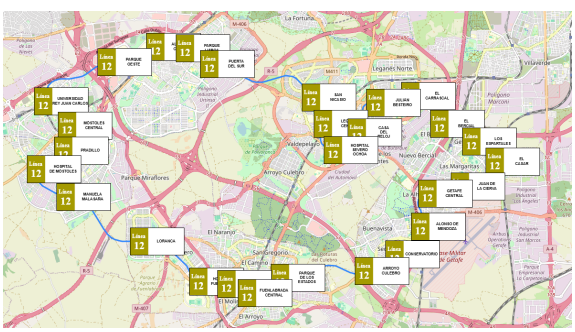


Fig. 5: Mapa del visualizador.

Este mapa representa además los servidores y *brokers* que estén definidos en la simulación. Estas entidades son inicializadas en los ficheros de despliegue y de plataforma de SimGrid con las coordenadas de cada una como parámetro. Adicionalmente, los servidores y los *brokers* han de presentar alguna manera de cambiar su disponibilidad. Así, el resultado de representar estas entidades en el visualizador se muestra en las Figuras 6.

A continuación, es necesario representar los trenes que se mueven por los tramos. Los datos de odometría de los trenes deben ser representados también en el visualizador. Los trenes deben presentar una manera de ser activados o desactivados, como se muestra en la Figura 7.

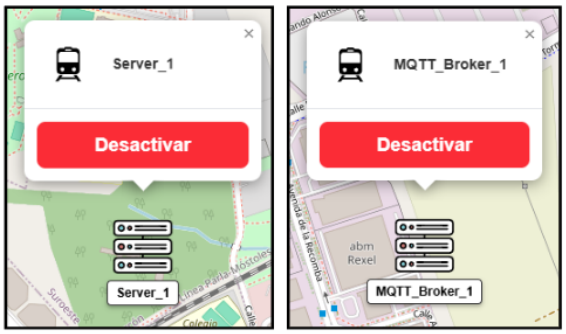


Fig. 6: Ejemplo de un servidor y un broker en el mapa.



Fig. 7: Ejemplo de un tren en el mapa.

Por último, es necesario mencionar que la información de los trenes se envían en un formato *JSON*, como se explicita en la descripción del método `sendSimulationState`. El *JSON* de este caso de uso contiene los siguientes campos:

- **ID**: define el nombre del tren.
- **coords**: define las coordenadas (latitud y longitud) del tren.
- **speed**: define la velocidad en metros por segundo del tren.
- **seconds_to_arrive**: define los segundos que el tren tardará en llegar a la próxima estación.
- **going_east**: define si el tren se dirige hacia el este o hacia el oeste, lo cual se usa para girar o no la imagen del tren.

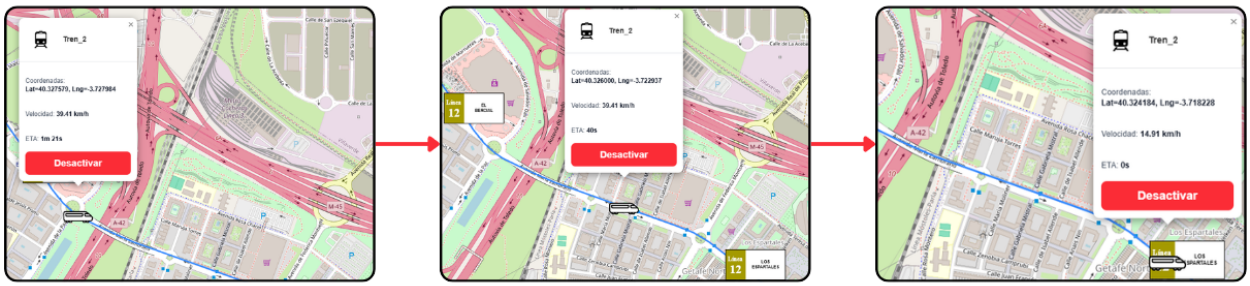


Fig. 8: Desplazamiento de un tren por el mapa.

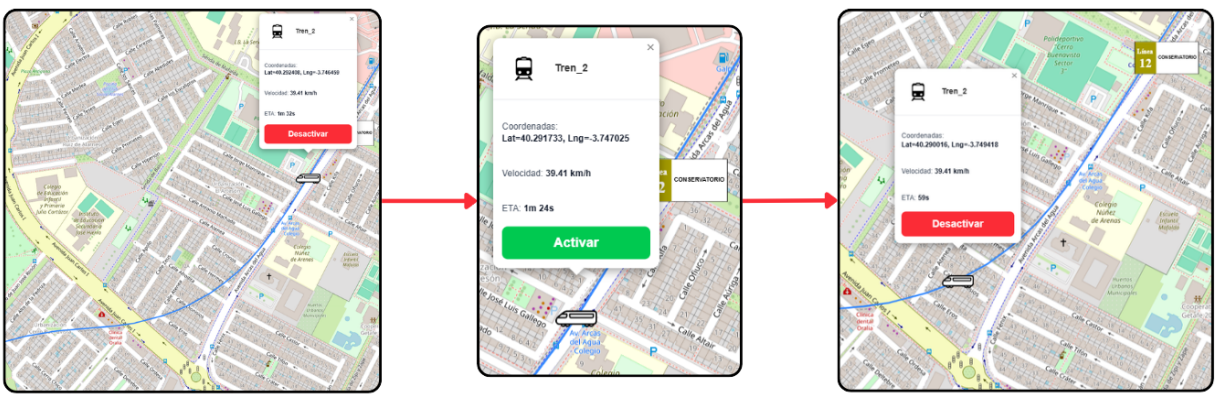


Fig. 9: Desactivación y reactivación de un tren.



Fig. 10: Desactivación y reactivación de un servidor y un *broker*.

D. Resultados del caso de uso

Para mostrar los resultados de este experimento, se emplea un fichero de plataforma pequeño que consta de:

- Dos servidores
- Dos *brokers*
- Cuatro trenes
- Un cliente

Si bien esta configuración describe una plataforma pequeña, esta se podría ampliar fácilmente para incluir un sistema mucho más complejo compuesto por un mayor número de entidades. Para poder verificar que la simulación funciona de manera correcta, se presentan las figuras 8, 9 y 10.

La figura 8 muestra a un tren desplazándose por uno de los tramos entre dos estaciones. A lo largo de las imágenes, se puede ver cómo el tren comienza cerca de la primera estación, se desplaza hasta la mitad del tramo y finaliza cuando llega a la segunda estación. Adicionalmente, se puede observar cómo el tren, a medida que se va acercando hacia la segunda estación, comienza a reducir su velocidad.

La figura 9 muestra qué ocurre cuando se desactiva y reactiva un tren, algo que se realiza mediante la *API* del simulador descrita en el apartado IV-B. En la primera figura se puede ver un tren desplazándose por un tramo de manera normal. En la segunda, es cuando se desactiva el tren, pulsando el botón de desactivación que, tras esto, cambia para poder volver a reactivar el tren. Por último, una vez que se pulsa el botón de activación del tren, este vuelve a comunicar sus datos, pero se encuentra en otro punto del tramo, ya que el tren ha estado avanzando mientras estaba desconectado del sistema.

Por último, la figura 10 muestra qué ocurre cuando se desactiva un servidor o uno de los *brokers MQTT*. Para poder visualizarlo mejor, se ha incluido una funcionalidad en el visualizador que permite ver los servidores y *brokers* implicados en una comunicación. En la primera imagen, se observa cómo un tren envía los datos al servidor 2 y este los reenvía al *broker* 1. Como se puede observar en la segunda imagen, si se desactiva el servidor 2, el tren comenzará a enviar los datos al segundo servidor más cercano, en este caso, el servidor 1. Finalmente, como se puede ver

en la tercera imagen, si se desactiva el *broker* 2, el servidor 1 comenzará a enviar los datos al segundo *broker* más cercano, en este caso, el *broker* 1.

V. Conclusiones y Trabajo Futuro

La metodología propuesta resuelve el problema de la visualización en SimGrid en tiempo real de forma efectiva. La generalidad de dicha metodología demuestra una flexibilidad para adaptarse a las necesidades, así como a la forma de cualquier simulación desarrollada en este *framework*. De la misma manera, la simplicidad de implementación hace que sea adoptable en cualquier proyecto y, de nuevo, atractiva en aquellas simulaciones en las que la visualización aporte valor al resultado final.

Si bien esta metodología resulta efectiva para resolver el problema mencionado, está lejos de ser una solución perfecta. Por ello, se definen los siguientes campos de mejora y trabajo futuro:

- **Integración de los métodos genéricos en SimGrid**: dados los resultados favorables obtenidos a partir del uso de los métodos genéricos desarrollados, el siguiente paso es la propuesta de integrar estas funciones como parte de la *API* oficial de SimGrid.
- **Integración del actor especial en SimGrid**: la creación de un actor especial cuyo único propósito sea exportar los datos hacia un visualizador es una solución temporal que puede entrar en conflicto con la ejecución de la simulación de SimGrid. Así, el objetivo final es el desarrollo de este actor integrado directamente en la *API* de SimGrid. Esto podría evitar los conflictos del actor con el funcionamiento normal de la simulación.
- **Pruebas de rendimiento**: un punto de estudio interesante es el análisis del impacto en el rendimiento de la simulación de la metodología. Este análisis debería observar la velocidad de la simulación, así como la velocidad de transferencia de los datos enviados hacia el visualizador.

Referencias

[1] El Mundo de los Datos, "Distribución t de student: Funcionamiento, propiedades y aplicaciones en estadística," 2023, Accedido el 27 de marzo de 2026.

[2] David Goldsman, Richard Nance, and James Wilson, "A brief history of simulation," 01 2010, pp. 310 – 313.

[3] Thomas J. Schriber, "Introduction to gpss," in *Proceedings of the 16th Conference on Winter Simulation*. 1984, WSC '84, p. 12–15, IEEE Press.

[4] Stephen V. Rice, Harry M. Markowitz, Ana Marjanski, and Stephen M. Bailey, "The simscript iii programming language for modular object-oriented simulation," in *Proceedings of the 37th Conference on Winter Simulation*. 2005, WSC '05, p. 621–630, Winter Simulation Conference.

[5] "Simula ibm system 360/370 and historical documentation," .

[6] Tao Luan, "A comprehensive review of simulation technology: Development, methods, applications, challenges and future trends," *International Journal of Emerging Technologies and Advanced Applications*, vol. 1, pp. 9–14, 06 2024.

[7] SimGrid, "Simulation of distributed computer systems," n.d., Accedido el 27 de marzo de 2026.

[8] "Visualizing simgrid traces with r," Accedido el 27 de marzo de 2026.

[9] Elías Del-Pozo-Puñal, Felix Garcia-Carballeira, and Diego Camarmas-Alonso, "Una propuesta para el uso de dispositivos móviles en simgrid: Aplicación a las simulaciones de iot," July 2022.

[10] Sairam Sri Vatsavai, Raees Khan, Kuan-Chieh Hsu, Ozgur O. Kilic, Paul Nilsson, Tatiana Korchuganova, David K. Park, Sankha Dutta, Yihui Ren, Joseph Boudreau, Tasnuva Chowdhury, Shengyu Feng, Jaehyung Kim, Scott Klasky, Tadashi Maeno, Verena Ingrid Martinez, Norbert Podhorszki, Frédéric Suter, Wei Yang, Yiming Yang, Shinjae Yoo, Alexei Klimentov, and Adolfy Hoisie, "Cgsim: A simulation framework for large scale distributed computing environment," 2025.

[11] Robert van Liere, Jurriaan D. Mulder, and Jarke J. van Wijk, "Computational steering," in *High-Performance Computing and Networking*, Heather Liddell, Adrian Colbrook, Bob Hertzberger, and Peter Sloot, Eds., Berlin, Heidelberg, 1996, pp. 696–702, Springer Berlin Heidelberg.

[12] "Course: In situ analysis and visualization with paraview catalyst and ascent — cscs," .

[13] "The modern age of computer systems simulation — simgrid documentation," .

[14] "Svelte • web development for the rest of us," .

[15] SimGrid, "Projects that use simgrid," 2025, Accedido el 27 de marzo de 2026.

Infraestructura P2P Agnóstica para Streaming Multimedia Seguro en Entornos Clínicos: Soberanía del Dato mediante Virtualización IA On-Premise

Miguel Ángel Sánchez-Olmedo[1], Savíns Puertas-Martín[1], Alejandra de Torres Sanchez[2], Pedro Mezquita Raya[2], Juana L. Redondo[1]

Resumen— **La investigación en salud digital y la monitorización de pacientes crónicos dependen cada vez más del intercambio de contenido multimedia de alta frecuencia: audio, vídeo e imágenes. Sin embargo, la mayoría de las plataformas existentes utilizan servicios en la nube de terceros (SaaS), lo que compromete la soberanía del dato y genera una brecha de seguridad-usabilidad en entornos clínicos. Este trabajo propone una infraestructura middleware robusta y agnóstica a la aplicación que proporciona una capa de comunicación segura para cualquier interacción médico-paciente, con independencia de la especialidad médica. La arquitectura combina una red mallada P2P basada en el protocolo WireGuard y travesía NAT asistida por STUN, que habilita túneles cifrados directos entre los dispositivos del paciente y los servidores hospitalarios. El backend se orquesta mediante un entorno de virtualización Proxmox con contenedores LXC para aislar las tareas de IA intensivas, como la transcripción con Whisper y los módulos de visión artificial, garantizando que los datos sensibles nunca abandonen el control jurisdiccional de la institución. La infraestructura fue validada en un caso de uso de endocrinología, demostrando alto rendimiento y cumplimiento normativo estricto.**

Palabras clave— **VPN P2P, WireGuard, STUN, Soberanía del dato, Telemedicina, LXC, Proxmox, IA on-premise.**

I. Introducción

La rápida digitalización sanitaria ha impulsado el uso de plataformas en la nube (SaaS) para teleconsultas y el intercambio de datos clínicos [1–3]. Aunque esta migración ofrece escalabilidad y ubicuidad, introduce importantes riesgos de privacidad, seguridad y soberanía del dato, al centralizar información sensible en servidores de terceros [4, 5]. La posible exposición de esta información por parte de los proveedores, junto a las estrictas normativas vigentes (HIPAA, RGPD o la Ley CLOUD), hacen imprescindible buscar enfoques técnicos y legales que garanticen la soberanía de los datos de los pacientes [1, 6].

Como alternativa a la nube, el establecimiento de flujos de datos directos (P2P) entre puntos clínicos choca frecuentemente con redes corporativas, cortafuegos y sistemas NAT que impiden la conectividad de extremo a extremo [7, 8]. La solución estándar basada en el protocolo ICE (con STUN/TURN) presenta limitaciones en redes clínicas restrictivas y puede sufrir penalizaciones de latencia inaceptables o exponer información sensible de la topología interna [9, 10]. Incluso protocolos de VPN avanzados y eficientes como WireGuard adolecen de un plano de control integrado y a menudo requieren mecanismos adicionales de travesía o servidores de retransmisión para funcionar en estos entornos [11, 12].

Simultáneamente, la integración de modelos de inteligencia artificial (IA) y reconocimiento de voz (ASR) en procesos médicos, si se realiza en la nube, incrementa gravemente el riesgo de fuga de información de salud protegida (PHI) [13, 14]. Para mitigar este riesgo, las investigaciones apuntan a la necesidad de desplegar estas cargas computacionales en infraestructuras locales (on-premise) o en el borde del sistema sanitario [15, 16]. En este contexto, las tecnologías de contenerización como LXC y Docker ofrecen un despliegue ágil, un mayor rendimiento y menor latencia en recintos clínicos en comparación con las máquinas virtuales tradicionales [17, 18].

Sin embargo, a pesar del estudio individual de estos problemas (los riesgos de la nube, la problemática de la conectividad y las ventajas de la IA local), existe una falta de integración profunda en la literatura [4]. Actualmente carecemos de arquitecturas unificadas que combinen redes P2P robustas con entornos de virtualización seguros y escalables para el procesamiento de IA in situ, garantizando en todo momento la soberanía de los datos hospitalarios.

Este artículo aborda dicha brecha presentando una arquitectura de doble capa, independiente de la aplicación concreta, diseñada para el procesamiento seguro de multimedia e IA clínica. A nivel de *red*, nuestra propuesta integra WireGuard con travesía NAT basada en STUN para forjar túneles P2P directos. A nivel de *infraestructura*, emplea el entorno de virtualización Proxmox junto a contenedores LXC para alojar las aplicaciones de IA sin abandonar el perímetro institucional seguro. Esta aproximación brinda una base de conectividad y computación confiable sobre la cual es posible asentar sistemas de telemedicina avanzados evitando los riesgos de privacidad asociados a arquitecturas SaaS.

[1]Dpto. de Informática, Universidad de Almería. e-mails: mso328@ual.es, savinspm@ual.es, jlredondo@ual.es
[2]Hospital Universitario Torrecárdenas, Almería, Spain. e-mails: ats.csp@gmail.com, pedro.mezquita.sspa@juntadeandalucia.es

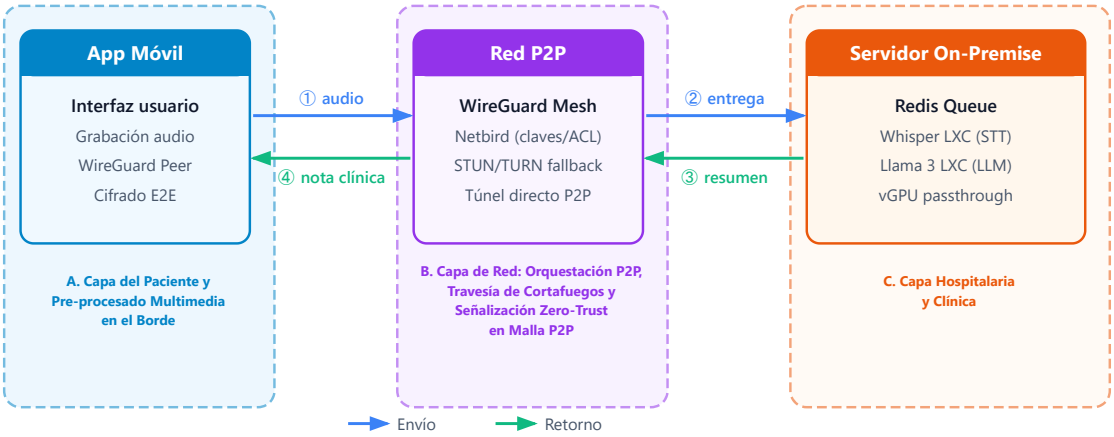


Fig. 1: Arquitectura global del sistema y ciclo de vida del dato para el intercambio mHealth, integrando el preprocesado en borde, la red P2P y el procesamiento local on-premise mediante contenedores de IA.

II. Arquitectura del Sistema y Diseño del Middleware

PARA superar las limitaciones organizativas y de los cortafuegos hospitalarios, y con la finalidad de garantizar una soberanía del dato de extremo a extremo, se ha diseñado un middleware robusto articulado en tres capas interdependientes (Figura 1).

El ciclo se inicia en la **Capa del Paciente**, donde la aplicación cliente optimiza y prepara el contenido multimedia de forma adaptativa. Continúa en la **Capa de Red**, que orquesta en tiempo real un túnel cifrado P2P en esquema de malla (*mesh network*), eludiendo infraestructuras de terceros. La información fluye hacia la **Capa Hospitalaria**, alojada en el perímetro seguro del servidor institucional, encargada de la recepción y encolado asíncrono de peticiones masivas para garantizar la escalabilidad. Dentro de esta misma capa se integra el **módulo clínico**, que mediante el aislamiento en contenedores on-premise despliega una inteligencia artificial capaz de inferir transcripciones e informes médicos, garantizando la privacidad absoluta de los datos.

A. Capa del Paciente y Pre-procesado Multimedia en el Borde

Para mitigar problemas de latencia o pérdida de paquetes en redes móviles deficientes, el middleware implementa técnicas de compresión en el propio dispositivo (*Edge Computing*). Las imágenes capturadas se reescalan de forma inteligente sin comprometer los marcadores diagnósticos fundamentales. Paralelamente, el flujo de audio se procesa utilizando el formato OGG codificado en Opus, con una tasa de bits variable (VBR). La alta resiliencia de este estándar frente a desconexiones temporales garantiza transferencias veloces que, sin esta contundente optimización, ralentizarían inevitablemente el flujo de la plataforma.

B. Capa de Red: Orquestación P2P y Travesía de Cortafuegos

Entregar los datos clínicos salvando NATs asimétricos y estrictos *firewalls* requiere el uso de herramientas especializadas en enrutamiento. La arquitectura se asienta sobre la plataforma *NetBird* [19] en combinación con el firme protocolo *WireGuard* [20] para hilar túneles P2P seguros.

B.1 Descubrimiento mediante STUN

Debido a que los *routers* tradicionales ocultan las redes mediante direcciones privadas, el sistema recurre a las utilidades cruzadas de STUN [21]. Como muestra la Figura 2, cliente y servidor emiten un *Binding Request* a un nodo público para exponer sus credenciales de red públicas. Obtenidas, ambos ejecutan una técnica bidireccional conocida como *UDP Hole Punching* [22], la cual perfora el cortafuegos y facilita un acoplamiento directo. En contextos de extrema restricción (NAT Simétrico), la red intercede redirigiendo el tráfico sin fisuras por relés cifrados (TURN).

B.2 Señalización Zero-Trust en Malla P2P

Respetando un paradigma de ciberseguridad *Zero-Trust*, los servidores controladores intermedios solo poseen acceso a los metadatos de señalización, quedando completamente ciegos frente a la carga clínica subyacente. Un Servidor de Gestión coordina la autorización, prestando paso al Servidor de Señal, el cual transfiere y actualiza las claves públicas. Validado este apretón de manos, el middleware conforma un túnel *WireGuard* encriptado nativamente (*ChaCha20-Poly1305*) que enlaza paciente e institución de par a par, eximiendo a cualquier actor central de visibilidad. El ciclo queda sintetizado organizativamente en la Tabla I.

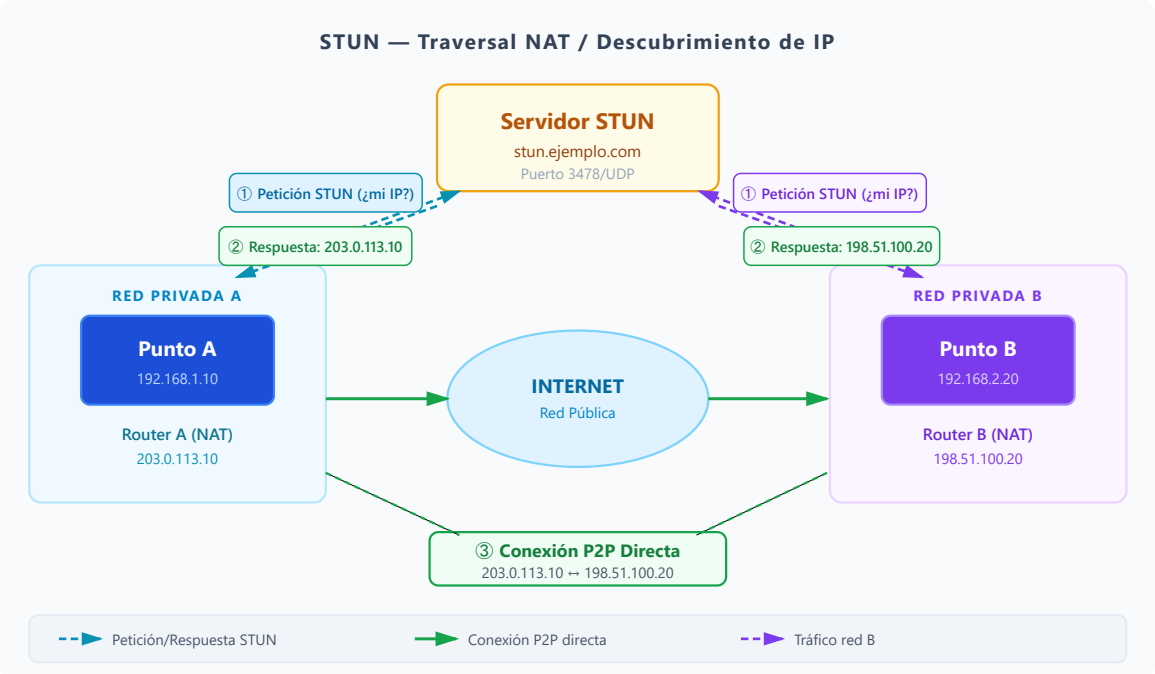


Fig. 2: Flujo técnico del descubrimiento coordinado por STUN. Abarca la solicitud a un servicio lógico externo inicial (pasos 1 y 2) y culmina habilitando y facilitando el definitivo cierre de la conexión E2E directa (paso 3).

Tabla I: Secuencia estandarizada del enlace de VPN P2P extremo a extremo.

Fase	Mecanismo Subyacente	Finalidad Transversal
1. Control	Servidor Ppal. de Gestión	Autenticación y registro de sesión (Validación ACL).
2. Estado	Servidor Emisor de Señal	Distribución cruzada de claves públicas e IPs.
3. Descubrir	Instancia Abierta STUN	Localización estricta de NAT a `<IP:Puerto>`.
4. Travesía	*UDP Hole Punching*	Acoplamiento y perforación del Firewall bidireccional.
5. Consumo	Túnel Cifrado WireGuard	Conexión nativa hermética final en bucle soberano E2E.

C. Capa Hospitalaria y Clínica: Orquestación LXC y Colas Asíncronas

Consolidada la entrega a través del blindaje P2P de red, el caudal de archivos clínicos y peticiones API arriban a la capa de hipervisor del bloque central **Proxmox VE** (Servidor Hospitalario). En los cimientos de la arquitectura interna de esta institución impera y domina un modelo de diseño estrictamente enfocado en la absorción asíncrona, indispensable para sortear eventuales colapsos o picos desbordantes por concurrencia (Figura 3).

C.1 Orquestación Asíncrona mediante Redis y FastAPI

El primer nivel de contención institucional es una pasarela analítica orquestada con FastAPI, la cual recibe los datos entrantes de tipo multipart. Sin embargo, para evitar los históricos colapsos presentes en sistemas monolíticos puramente síncronos, FastAPI no bloquea las rutinas en su recepción. En su lugar, el sistema deposita referencias ultrarrápidas de cada evento entrante en un clúster **Redis**, una memoria caché de enorme resiliencia que oficia como *message broker* (cola activa transitoria).

Desde dicho búfer de transferencia, el sistema se estabiliza. Los verdaderos operarios de *backend* o lógicas especializadas (trabajadores de red) proceden sólo a extraer y efectuar resoluciones de nuevas peticiones médicas del bloque intermedio cuando confirman asertivamente que cuentan con capacidad latente, hilos suficientes o memoria (RAM/VRAM) sobrante para hacerlo. Esto aplaca de manera rotunda las denegaciones o desbordamientos del centro intrahospitalario durante jornadas laborales o guardias con volúmenes álgidos de trabajo.

C.2 Procesamiento Aislado por LXC (Contenedores) en Modelos IA

Para conformar estricto apego con el Reglamento General de Protección de Datos (RGPD) en materia de información sensible o biométrica (PHI), la aproximación local evita frontalmente subcontratar o enviar el pesado esquema computacional de la analítica IA de grabaciones crudas hacia nubes foráneas de terceros. Como mitigación de peso, este servidor maestro subyacente despliega complejas rutinas algorítmicas aislando los ecosistemas computacionales con perfiles delimitados o enjaulados, empleando la fiabilidad de contención por contenedores virtuales de Linux (**LXC**).

En estos recintos paralelos, apoyándose por tecnología de virtualización interna *GPU Passthrough*

(**vGPU**), los contenedores acceden directamente de forma localística a los clústeres nativos de procesamiento de aceleración de matriz, sin merma representativa de rendimiento o cuellos asimétricos entre ellas.

Esta interrelación unificada activa dos líneas de respuesta en forma escalonada. La primera alberga una instancia analítica procesando audios sobre una ingesta rápida de *Whisper Large-v3* agilizando la fónica conversión cruda de dictados complejos multidimensionales y términos del argot médico hacia las precisas formaciones de transcripciones (Voz-a-Texto). Acto seguido, superpuesto sobre esa lógica fónica, se desencadena el hilo analítico de IA secundario, fundamentado interdependientemente en un núcleo multimodal con motor *Llama 3*. Este último bloque recibe los literales generados por la conversión, procesando cognitivamente los diagnósticos con fluidez y derivando informes depurados ejecutables. Transcurrida toda esta inferencia y sin cruzar datos reales extramuros, las fichas sintetizan una base estructurada o deducciones del caso a la disposición informática de la red e intranet hospitalaria, agilizando firmemente el control directo a los profesionales durante su validación ocular.

III. Framework de Trabajo

Para validar el rendimiento y la viabilidad del middleware agnóstico propuesto, se ha diseñado un entorno experimental que evalúa la robustez, eficiencia y resiliencia del canal de comunicación P2P. El escenario de pruebas compara dos tipos de conectividad que representan los entornos sanitarios actuales: una **red celular 5G** (para escenarios de movilidad y emergencias) y una **conexión de Fibra Óptica combinada con WiFi 5 GHz** (típica de redes intrahospitalarias).

Las pruebas desde el lado del cliente se ejecutaron en un dispositivo **OPPO A54 5G** (SoC Snapdragon 480 5G, 4 GB RAM, Android 11), empleando el cliente WireGuard de NetBird. Las especificaciones detalladas del hardware empleado se resumen en la **Tabla II**. El despliegue establece un túnel P2P cifrado mediante *ChaCha20-Poly1305* hacia un servidor local, utilizando HTTP multipart para la transferencia. La metodología se centró en tres niveles: la caracterización del enlace monousuario, pruebas de estrés de concurrencia masiva en el backbone hospitalario (hasta 768 flujos paralelos), y la medición del impacto computacional del cifrado y transmisión en el dispositivo móvil con distintos volúmenes de datos.

Además, aunque la infraestructura es agnóstica a la aplicación, su viabilidad clínica fue validada en el marco de un caso de uso real en investigación en endocrinología. En este contexto, los pacientes compartieron registros nutricionales y clips de audio explicativos. El sistema permitió automatizar con éxito la transcripción y resumir los resultados, entregando datos estructurados directamente al puesto de trabajo del endocrinólogo y garantizando en todo momento que la información (gráfica y sonora) se mantuviera dentro del perímetro seguro del hospital.

Tabla II: Especificaciones del dispositivo cliente — OPPO A54 5G (CPH2195).

Especificación	**Valor**
Modelo	OPPO A54 5G (CPH2195)
SoC	Qualcomm Snapdragon 480 5G
CPU	2×2,0 GHz Kryo 460 + 6×1,8 GHz
GPU	Adreno 619 @ 650 MHz
RAM	4 GB LPDDR4X
Almacenamiento	64 GB UFS 2.1
SO	Android 11 (ColorOS 11.1)
Conectividad	Wi-Fi 5, Bluetooth 5.1, 5G/4G LTE

IV. Resultados

A. Rendimiento del Enlace y Calidad de Servicio

El análisis de Calidad de Servicio (QoS) demuestra que ambas tecnologías de red mantienen una estabilidad excepcional, fundamental para aplicaciones clínicas de IA en tiempo real. La latencia (jitter UDP) se mantuvo en niveles sumamente bajos, con una pérdida de paquetes nula en ambos escenarios. La **Tabla III** presenta estos valores en detalle.

Tabla III: Métricas de Calidad de Servicio (QoS) – comparativa 5G vs. Fibra + WiFi 5 GHz.

Métrica	**5G**	**Fibra**	**Umbral**	**Eval.**
Jitter UDP	0,604 ms	0,070 ms	<30 ms	Excelente
Pérdida paquetes	0 %	0 %	<1 %	Excelente
Retrans. TCP	48	281	<100	5G Superior
Subida/Bajada	0,42	1,53	–	Asimétrico

Para observar el comportamiento del sistema bajo carga sostenida, la **Figura 4** ilustra la evolución temporal del bitrate TCP durante una ventana de 10 segundos. Se confirma que, aunque la Fibra ofrece mayor volumen, ambas tecnologías mantienen una consistencia temporal notable.

B. Resiliencia bajo Concurrencia Masiva

Para evaluar la infraestructura bajo condiciones de gran escala y validar su resiliencia estructural, se realizó una severa prueba de estrés concurrente exclusivamente sobre el enlace Ethernet-sobre-Fibra de 1 Gbps, inyectando 768 flujos TCP. La **Tabla IV** detalla los resultados de esta ventana de 30 segundos, alcanzando un throughput muy cercado a la capacidad máxima asintótica.

Tabla IV: Desglose estadístico de la prueba de estrés TCP multi-flujo concurrente sobre Gigabit Fibra ($128 \times 3 \times 2 = 768$ flujos concurrentes).

Instancia	**Datos**	**Thr. Rx**	**Reintentos**	**Eval.**
Puerto 5201	941 MB	261 Mb/s	26.351	Estable
Puerto 5203	930 MB	258 Mb/s	26.971	Estable
Puerto 5202	929 MB	258 Mb/s	26.690	Estable
Total	**2,80 GB**	**777 Mb/s**	**80.012**	**Saturación**

El sistema mantuvo la estabilidad estructural sin registrar caídas y aseguró una distribución equitativa del ancho de banda, con un promedio de 2,1 Mbps

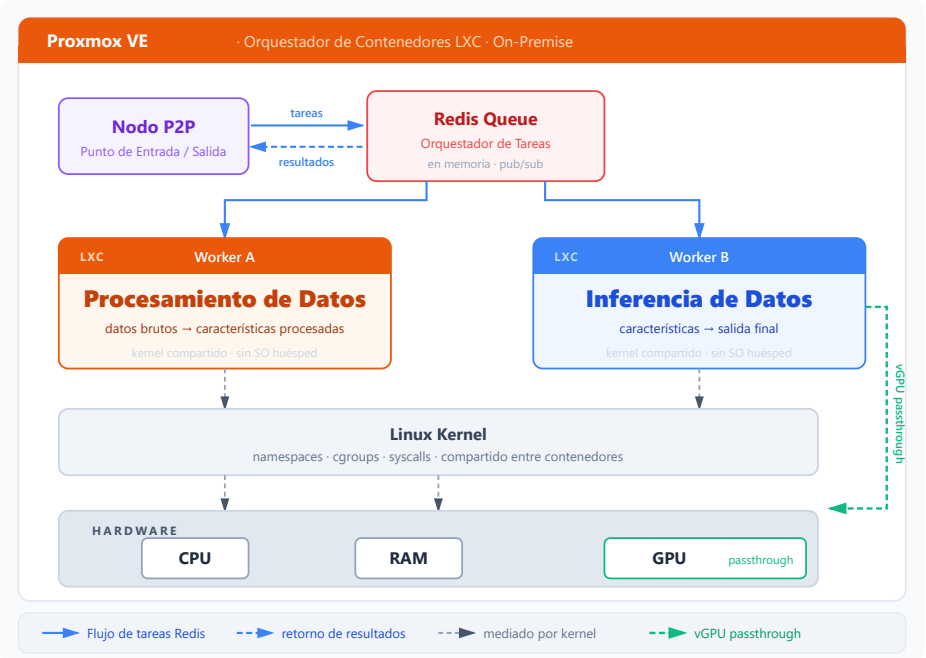


Fig. 3: Diagrama estandarizado del encolado escalable y orquestación interna procesado en Redis (modelo *message broker*). Muestra al detalle el puente que canaliza peticiones intrahospitalarias asíncronas desde internet mediante LXC passthrough, hasta lograr la estricta, enjaulada y secuencial inferencia (Voz/Llama 3) sin depender en absoluto de servidores ajenos.

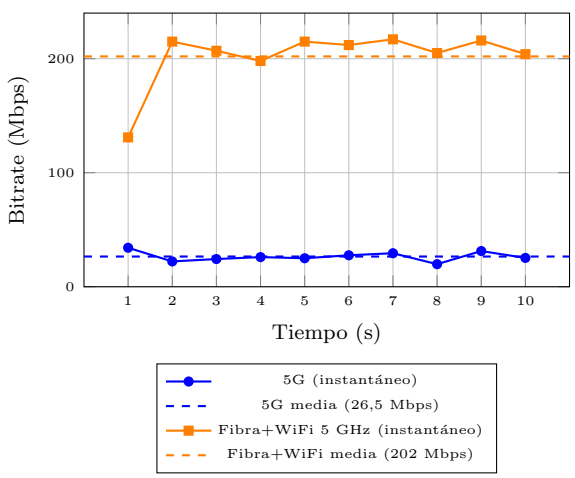


Fig. 4: Evolución temporal del ancho de banda en el test TCP de flujo único.

por flujo, de modo que ningún dispositivo médico quedara sin recursos.

C. Impacto Computacional en el Cliente

La evaluación del procesamiento local en el dispositivo móvil reveló que la carga criptográfica escala de manera lineal con el tamaño de los datos trasferidos. La **Tabla V** muestra el tiempo medio de CPU por fase, evidenciando que el cifrado y la transmisión dominan el uso.

La comparativa entre el uso de una conexión directa frente al túnel VPN activo muestra que el rendimiento máximo se reduce un 41 % (11,3 vs. 19,1 Mbps) con VPN debido a la sobrecarga criptográfica. Sin embargo, este margen excede los requisitos para transmisiones multimedia clínicas en tiempo real. En ensayos clínicos reales, este grado de rendimiento permitió transferir contenido médico en un entorno endocrinológico garantizando plena privacidad y soberanía de los datos, sin merma de la fluidez en la plataforma.

Tabla V: Tiempo medio de CPU (ms) por fase en estado estacionario.

Fase	128 MB	256 MB	512 MB	1024 MB
Cifrado	3,795	7,717	15,359	27,837
Transmisión	7,719	15,314	30,412	54,610
Descifrado	650	758	798	1,013
Total	**12,164**	**23,789**	**46,569**	**83,460**

V. Discusión

Los resultados experimentales obtenidos constatan que es viable abandonar la *"trampa de dependencia"* de los ecosistemas en la nube pública (*SaaS*) al implementar IA generativa clínica. Al mantener la orquestación del backend en la red privada del entorno hospitalario y utilizar un túnel P2P directo (STUN), se garantiza una soberanía de datos absoluta, cumpliendo con las exigentes directrices del RGPD sobre datos biométricos en la nube pública.

El determinismo técnico de la solución ha quedado demostrado alcanzando una latencia (jitter) de apenas 0,60 ms en redes móviles 5G y de 0,07 ms en Fibra, evitando el convencional e impredecible enrutamiento de *"doble salto"* que imponen las APIs externas. Al mismo tiempo, la prueba de estrés de concurrencia masiva (soportando 768 flujos intrahospitalarios) y el análisis del impacto computacional

refrendan su robustez operativa. A pesar de que el túnel VPN P2P incurre en una penalización del 41 % del ancho de banda y eleva el uso de la CPU por el cifrado (ChaCha20), se sostiene un rendimiento (*throughput*) medio de 11,3 Mbps. Este margen holgado se sitúa muy por encima de los umbrales exigibles para herramientas de transcripción multimedia clínica de gran volumen, garantizando viabilidad técnica sin sacrificar la seguridad.

VI. Conclusiones

ESTE estudio evidencia que el requisito estricto de proteger los datos sanitarios sensibles en infraestructuras locales no supone una barrera para la inteligencia artificial de alto impacto, sino un espacio para la innovación segura. La arquitectura P2P agnóstica probada, exenta de dependencias de terceros y dotada de un cifrado sólido mediante tecnologías como WireGuard, soporta un despliegue masivo y picos de tráfico con escaso impacto para el usuario final.

De cara a desarrollos futuros, la consecución de herramientas de IA totalmente desconectadas conforma la base ideal para habilitar entornos de Aprendizaje Federado (*Federated Learning*) en hospitales conectados. Bajo dicho paradigma colaborativo, los modelos podrán entrenarse globalmente trasladándose directamente al servidor del propio centro sin el riesgo de intercambiar la información primaria del paciente, sentando el camino para investigaciones multicéntricas robustas, deterministas y privativas.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación. Programa Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica (Comp4Health-II, PID2024-155521OB-I00) y supervisado por el Proyecto GluVarHUT del Hospital Universitario Torrecárdenas. Savíns Puertas-Martín es beneficiario del Programa de Formación, Captación, Incorporación y Movilidad de los Recursos Humanos de I+D+I de la Estrategia EIDIA, Horizonte 2027, financiado por la Consejería de Universidad, Investigación e Innovación de la Junta de Andalucía y cofinanciado por el Programa FSE+ de Andalucía 2021-2027.

Referencias

[1] Z. Lakhani, "The role of cloud integration in healthcare sector: A comprehensive review," *International Journal for Research in Applied Science and Engineering Technology*, vol. 12, no. 4, pp. 4280–4285, 2024.

[2] A. Arafa, H. Sheerah, and S. Alsalamah, "Emerging digital technologies in healthcare with a spotlight on cybersecurity: A narrative review," *Information*, vol. 14, no. 12, pp. 640, 2023.

[3] S. Sachdeva, S. Bhatia, A. Al-Harrasi, Y. Shah, K. Anwer, A. Philip, S. Shah, A. Khan, and S. Halim, "Unraveling the role of cloud computing in health care system and biomedical sciences," *Heliyon*, vol. 10, no. 7, pp. e29044, 2024.

[4] Y. Al-Issa, M. Ottom, and A. Tamrawi, "eHealth cloud security challenges: A survey," *Journal of Healthcare Engineering*, vol. 2019, pp. 1–15, 2019.

[5] L. Griebel, H. Prokosch, F. Köpcke, D. Toddenroth, J. Christoph, I. Leb, I. Engel, and M. Sedlmayr, "A scoping review of cloud computing in healthcare," *BMC Medical Informatics and Decision Making*, vol. 15, no. 1, 2015.

[6] F. Blancato, "The cloud sovereignty nexus: How the European Union seeks to reverse strategic dependencies in its digital ecosystem," *Policy & Internet*, vol. 16, no. 1, pp. 12–32, 2023.

[7] J. Paillissé, A. Barcia, A. López, A. Rodríguez-Natal, F. Maino, and A. Cabellos-Aparicio, "A control plane for WireGuard," in *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*, 2021.

[8] V. Kjorveziroski, C. Bernad, K. Gilly, and S. Filiposka, "Full-mesh VPN performance evaluation for a secure edge–cloud continuum," *Software: Practice and Experience*, vol. 54, no. 8, pp. 1543–1564, 2024.

[9] S. Tatschner, F. Jarisch, A. Giehl, S. Plaga, and T. Newe, "The stream exchange protocol: A secure and lightweight tool for decentralized connection establishment," *Sensors*, vol. 21, no. 15, pp. 4969, 2021.

[10] WebRTC privacy analysis, "WebRTC privacy analysis," *Sensors*, vol. 17, no. 11, pp. 2650, 2017.

[11] X. Zheng and V. Oleshchuk, "The design of secure and efficient P2PSIP communication systems," in *Lecture Notes in Computer Science*, pp. 253–260. 2010.

[12] A. Müller, A. Klenk, and G. Carle, "On the applicability of knowledge based NAT-traversal for home networks," in *Lecture Notes in Computer Science*, pp. 264–275. 2008.

[13] N. Tasmurzayev, B. Amangeldy, B. Imanbek, Z. Baigarayeva, T. Imankulov, G. Dikhanbayeva, I. Amangeldi, and S. Sharipova, "Digital cardiovascular twins, AI agents, and sensor data: A narrative review from system architecture to proactive heart health," *Sensors*, vol. 25, no. 17, pp. 5272, 2025.

[14] J. Wu, "Edge speech-enabled LLM pipelines for healthcare," 2025.

[15] S. Kaiser, M. Haq, A. Tosun, and T. Korkmaz, "Container technologies for ARM architecture: A comprehensive survey of the state-of-the-art," *IEEE Access*, vol. 10, pp. 84853–84881, 2022.

[16] R. Pope, A. Zenonos, W. Bryant, A. Spiridou, D. Key, S. Patel, J. Robinson, A. Styles, C. Rockenbach, G. Bicknell, P. Rajendran, A. Taylor, and N. Sebire, "Real-world learnings for digital health industry–NHS collaboration: Life sciences vision in action," *Future Healthcare Journal*, vol. 11, no. 3, pp. 100168, 2024.

[17] M. Guri, "Pico-cloud: Cloud infrastructure for tiny edge devices," 2025.
[18] O. Freyer, I. Wiest, and S. Gilbert, "Policing the boundary between responsible and irresponsible placing on the market of large language model health applications," *Mayo Clinic Proceedings: Digital Health*, vol. 3, no. 1, pp. 100196, 2025.
[19] NetBird Authors, "NetBird: Open-source wireguard-based overlay network," `https://netbird.io`, 2024.
[20] J. A. Donenfeld, "WireGuard: Next generation kernel network tunnel," Proc. NDSS 2017, 2017.
[21] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session traversal utilities for nat (stun)," Tech. Rep., 2008.
[22] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-peer communication across network address translators.," in *USENIX ATC, General Track*, 2005, pp. 179–192.

Modelado y simulación de la red de adquisición de datos de ATLAS para el HL-LHC

Carlos Medrano Navalón[1] [2], Mikel Eukeni Pozo Astigarraga[1], Gabriel Gómez-López[2], Jesús Escudero-Sahuquillo[2], Pedro Javier García[2], Francisco J. Quiles[2]
{carlos.medrano,eukeni.pozo}@cern.ch,
{gabriel.gomez, jesus.escudero, pedrojavier.garcia, francisco.quiles}@uclm.es

Resumen— **La próxima actualización de Alta Luminosidad del Gran Colisionador de Hadrones (HL-LHC) plantea un desafío para el Sistema de Adquisición de datos (DAQ) del detector ATLAS, que deberá adquirir y procesar datos a tasas de hasta 4,6 TB/s. El efecto *incast* es habitualmente un cuello de botella crítico en este tipo de entornos de alto rendimiento, donde las ráfagas de tráfico altamente sincronizadas pueden provocar la saturación de los búferes y la pérdida de paquetes, lo que deriva en una degradación significativa del rendimiento. Si bien las herramientas de simulación existentes proporcionan un modelado de alta fidelidad de los componentes de red, a menudo carecen de la escalabilidad necesaria para simular sistemas a gran escala, con miles de nodos, como es el caso del futuro DAQ de ATLAS. En este artículo, presentamos un modelo de simulación de red de eventos discretos escalable, basado en OMNeT++, capaz de representar con precisión la dinámica de la red de adquisición de datos de ATLAS. El modelo ha sido validado frente a un banco de pruebas experimental representativo del sistema real, demostrando una alta precisión en la reproducción de las latencias de recolección de datos y de los patrones de ocupación de los búferes. Este marco de trabajo proporciona una herramienta robusta para la toma de decisiones informadas sobre la adquisición de hardware para la próxima generación de experimentos de física de partículas.**

Palabras clave— **ATLAS, DAQ, shared-buffer, simulación por eventos discretos, topología, OMNeT++, incast.**

I. Introducción

DURANTE el periodo denominado «*Long Shutdown 3*» (2026-2030), el detector ATLAS [1] del CERN se someterá a una serie de mejoras para operar bajo las nuevas condiciones de Alta Luminosidad (HL) [2] a partir de la cuarta fase (*Run4*, 2030-2033) de toma de datos en el Gran Colisionador de Hadrones (LHC). El conjunto de datos que describe los resultados de las interacciones de protones producidas por el HL-LHC dentro del detector para un cruce determinado de paquetes de protones se conoce como «evento». Con respecto a las condiciones operativas del *Run3* (2022-2026), se espera que el tamaño medio de los eventos se duplique respecto a su valor actual, hasta alcanzar los 4,6 MB, mientras que la tasa de lectura de eventos del detector se multiplicará por diez hasta alcanzar 1 MHz (es decir, 1.000.000 de eventos por segundo). Esto representa un aumento de 20 veces en el rendimiento (4,6 TB/s) respecto al sistema actual.

Bajo estas nuevas condiciones, se está diseñando

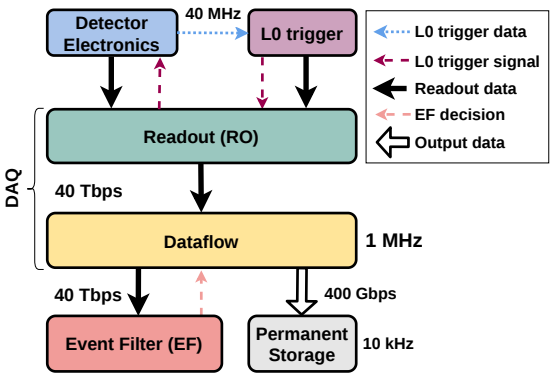


Fig. 1: ATLAS «*Trigger*» y DAQ para el HL-LHC

actualmente un nuevo DAQ. Como se muestra en la Figura 1, tras la señal del «*Trigger*» de nivel 0 (que funciona a 1 MHz), los servidores Readout (RO) leen los datos proporcionados por los componentes electrónicos del detector y el sistema denominado «*Dataflow*» envía los fragmentos de eventos correspondientes a la granja de servidores de Filtrado de Eventos (EF), encargada de filtrar y cribar los eventos que no resultan interesantes para estudios de física. Cada evento individual se asigna a un nodo distinto de la granja y los fragmentos de datos correspondientes a las distintas partes del detector se ensamblan mediante un proceso conocido como «construcción de eventos» (*Event Building*). Una vez reconstruidos los eventos completos, el EF reduce aún más la tasa de eventos en un factor de 100, lo que da como resultado una tasa de salida de 10 kHz de eventos que se almacenan de forma permanente para estudios físicos fuera de línea.

Para llevar a cabo el proceso de ensamblado de eventos de manera eficiente, el sistema Dataflow se basa en una red de computadores de alto rendimiento que interconecta los servidores RO con la granja EF. Dados los notables avances logrados por la industria de las redes en los últimos años, nos encontramos ante una oportunidad única para incorporar los últimos avances en velocidad de enlace, número de puertos y capacidad de reenvío de paquetes en el nuevo sistema DAQ para el Run4 de ATLAS. Sin embargo, la integración de estas tecnologías conlleva el riesgo de adquirir hardware que no se ajuste a las necesidades del sistema. Por ejemplo, es difícil estimar con precisión los requisitos del búfer del conmutador debido a las complejas dinámicas del tráfico que implica. En

[1]CERN
[2]Universidad de Castilla-La Mancha

consecuencia, herramientas como la creación de prototipos y la simulación de redes resultan muy útiles para tomar decisiones basadas en datos, lo que permite a los ingenieros del DAQ comprender el comportamiento del sistema futuro antes de su implementación física.

En este trabajo, presentamos un modelo de simulación de la red DAQ de ATLAS para la siguiente fase del LHC, que pretende convertirse en una potente herramienta para ayudar a los ingenieros del experimento en el diseño y la puesta en marcha del nuevo sistema DAQ. Las siguientes secciones están estructuradas de la siguiente manera. En la sección II ofrecemos una visión general del fenómeno «*incast*» y una revisión de los modelos de simulación empleados previamente en diversos DAQ para experimentos de física de altas energías. El modelo de simulación de la red y los patrones de tráfico se presentan en la sección III. En la sección IV explicamos el proceso de validación, incluyendo la configuración de los experimentos y los resultados. Por último, en la sección V se presentan las conclusiones, junto con las tareas futuras relacionadas con el modelo de simulación.

II. Trabajo relacionado

Cuando múltiples flujos de paquetes compiten por los mismos recursos de red, se produce una situación de contención. La contención persistente se convierte en congestión, lo que puede aumentar la latencia de estos mensajes o provocar la pérdida de paquetes en redes con pérdidas. Las ráfagas de paquetes altamente sincronizadas de muchos a uno suelen provocar este tipo de congestión, lo que supone un reto para los ingenieros de redes. Este patrón se conoce comúnmente como tráfico «*incast*». El fenómeno del *incast* ha sido un problema ampliamente estudiado durante décadas en el contexto de las redes de interconexión de altas prestaciones [3], [4], [5], donde las aplicaciones con requisitos de baja latencia pueden sufrir una degradación del rendimiento debido a este fenómeno. Por lo tanto, existe una amplia gama de propuestas para mitigar los efectos del *incast*. Las soluciones más conocidas van desde protocolos a nivel de transporte, como DCTCP [6] para TCP/IP, hasta mecanismos especializados de control de congestión, como DCQCN [7] para redes RDMA, así como soluciones específicas de la capa de aplicación o el uso de búferes de paquetes más grandes en los conmutadores de red.

En lo que respecta a los sistemas de adquisición de datos científicos, el problema del *incast* suele suponer un reto durante el diseño del sistema. En sistemas comparables existen requisitos similares a los de la red del DAQ de ATLAS. En concreto, los experimentos CMS [8], ALICE [9] y LHCb [10] del LHC también utilizan un sistema DAQ para enviar datos a una granja de filtrado. Teniendo en cuenta las elevadas necesidades de rendimiento de estos sistemas, se han realizado múltiples estudios para determinar los requisitos de hardware. En este contexto, la simulación es un enfoque habitual para comprender las capacidades de un sistema complejo antes de que el hardware o el software estén disponibles. Por ejemplo, se han propuesto herramientas de simulación de redes para estudiar los patrones de tráfico de las actualizaciones anteriores de la red DAQ del LHCb [11], [12], que utiliza la tecnología de red InfiniBand. También se propuso un modelo de simulación de Storage Area Network (SAN) para la colaboración ALICE [13], mientras que el experimento CMS caracterizó su sistema DAQ mediante diferentes enfoques [14], [15]. No obstante, las arquitecturas específicas de estos experimentos presentan diferencias notables debido a las restricciones propias de cada detector, lo que requiere un estudio centrado en la red DAQ de ATLAS.

Para la red ATLAS en concreto, se propuso un modelo de simulación en [16] y [17], que modela con precisión las latencias de recopilación de datos para la actualización del Run3. Este modelo también se basa en OMNeT++ y utiliza el modelo INET [18]. Aunque este modelo basado en INET era muy preciso para la red TCP del periodo Run3, no está actualizado a los requisitos actuales del sistema: se prevé una granja de filtrado mucho mayor y las tecnologías emergentes, como RDMA over Converged Ethernet (RoCE), suponen una alternativa razonable a TCP que INET no admite actualmente. Un estudio más reciente [19] utilizó modelos analíticos y de simulación para predecir la ocupación del búfer de los conmutadores de agregación, o conmutadores Top-of-Rack (ToR), del EF. Si bien estos modelos proporcionan información valiosa, también están desactualizados en cuanto a las restricciones de diseño.

A medida que la red del DAQ se adapta a los requisitos del HL-LHC, la complejidad computacional de la simulación de bajo nivel se convierte en un obstáculo. Aunque las herramientas existentes, como INET, proporcionan modelos de TCP/IP detallados, su sobrecarga suele limitar la capacidad para simular los miles de nodos necesarios para estudios de sistemas completos. Además, las tecnologías emergentes de baja latencia, como RoCE, no son actualmente compatibles con OMNeT++ a esta escala. Aunque existen modelos especializados para otras herramientas de simulación, como NS-3 [20] o SST [21], existe una clara necesidad de un modelo validado, de alto rendimiento y flexible que permita evaluar múltiples arquitecturas a gran escala para el nuevo sistema DAQ de ATLAS. Este artículo presenta el desarrollo y la validación de un modelo de este tipo, capaz de representar los patrones de tráfico específicos generados en la red de adquisición de datos de ATLAS a la escala necesaria.

III. Modelo de simulación

El modelo de simulación utilizado en este trabajo es SAURON [22], una herramienta de simulación de redes basada en OMNeT++. OMNeT++ es un *framework* de simulación basado en eventos discretos [23] que proporciona un conjunto de herramientas para facilitar el desarrollo de modelos de simulación, incluyendo un entorno de desarrollo, estadísticas, he-

rramientas de análisis de datos, un lenguaje de descripción de topologías, etc. Estos modelos de simulación específicos pueden construirse sobre OMNeT++ para simular diferentes dominios particulares, normalmente entornos de red, que constituyen su objetivo principal. Un modelo de OMNeT++ consiste en una colección de módulos, simples o compuestos. Estos módulos se comunican entre sí mediante paso de mensajes. En cuanto a SAURON, se trata de un modelo centrado en redes de interconexión de alto rendimiento, aplicable a escenarios como supercomputadores, centros de procesamiento de datos y redes de adquisición de datos. El modelo pretende ser altamente escalable, de modo que se puedan simular topologías de gran tamaño, pero también preciso al modelar componentes específicos o tecnologías de red, como arquitecturas de conmutadores o técnicas de control de flujo. Además, SAURON ha participado recientemente en el diseño y la evaluación de nuevas tecnologías, como BXI [24], [25]. Este modelo de simulación ofrece, entre otras, las siguientes características principales:

- Compatibilidad con diferentes tecnologías de red: los modelos de conmutadores Input-Queued (IQ) y Combined Input-Output Queued (CIOQ) pueden configurarse para simular varias generaciones de conmutadores InfiniBand (DDR, QDR, EDR) o RoCE.
- Amplia gama de topologías, incluidas múltiples variantes de Fat-Tree [26], [27], variantes de Dragonfly [28], Dragonfly+ [29], hipercubos, mallas, toros y KNS [30]. Se pueden utilizar algoritmos de encaminamiento determinista, «*packet spraying*» y ciertos algoritmos de encaminamiento adaptativo en función de la topología.
- Cargas de tráfico: generación de acuerdo con múltiples distribuciones de destinos y tamaños de mensajes, o mediante la reproducción de trazas de aplicaciones Message Passing Interface (MPI) reales.
- Control de flujo Priority-based Flow Control (PFC) o basado en créditos.
- Múltiples esquemas de colas estáticas, como DBBM [31], Flow2SL [32], vFtree [33], BBQ [34], etc.
- Control de congestión mediante limitación de inyección en redes RDMA mediante DCQCN [24].

Dado que muchos de los requisitos para nuestro objetivo ya están implementados en este modelo de simulación, y dada su alta escalabilidad y flexibilidad a la hora de modelar arquitecturas y protocolos de red con precisión, SAURON ha sido elegido como base para el desarrollo del modelo de red del DAQ de ATLAS. Sin embargo, hemos identificado varios requisitos clave para la red que no estaban disponibles en SAURON, entre ellos el patrón de tráfico, una arquitectura de conmutador con búfer compartido, funciones básicas de control de congestión y las topologías específicas tanto para la configuración de prueba DAQ como para el sistema final. En consecuencia, el simulador se ha ampliado con estas funcionalidades, que se describen con más detalle en esta sección.

A. Generación de tráfico

El patrón de tráfico generado por el proceso de construcción de eventos (Event Building) en la red DAQ de ATLAS es sumamente específico. Mientras que otros escenarios de red suelen diseñarse para patrones de tráfico diversos y difíciles de predecir, la red DAQ de ATLAS está adaptada a un patrón bien definido, lo que permite un modelado detallado de la carga de trabajo de tráfico esperada en el simulador.

Se espera que un evento tenga un tamaño de 4,6 MB en el HL-LHC. Los eventos se asignan a los nodos del EF a una tasa determinada según la disponibilidad de estos para procesarlos. La granja EF es responsable de seleccionar los eventos que se almacenarán permanentemente y los que se descartarán. En consecuencia, para cada evento, el nodo del EF debe construir el evento recuperando sus datos asociados. Estos datos están fragmentados en todos los nodos del sistema RO. Para reunir todos los fragmentos asociados a un evento determinado, la aplicación de ensamblado, el Event Builder (EB), que se ejecuta en el nodo del EF, envía peticiones a todas las aplicaciones Data Handler (DH) que operan en los nodos RO. Hay dos aplicaciones DH ejecutándose en cada servidor RO (una por CPU) y un EB por cada servidor EF. Esto supone un total de unas 600 aplicaciones DH y hasta 4000 EB en el Run4. Las peticiones tienen una carga útil (*payload*) insignificante, normalmente inferior a la de las cabeceras de red.

Al recibir una petición, una aplicación DH la procesa y responde con el fragmento de datos correspondiente. Dado que las 600 aplicaciones DH responden casi simultáneamente, las ráfagas de tráfico sincronizado provocan un fuerte patrón de congestión *incast* de muchos a uno que puede saturar los búferes de los conmutadores.

Además, los conmutadores de agregación ToR del EF entregan los datos a los nodos del EF a una velocidad sustancialmente menor que la de recepción, ya que los enlaces que interconectan los servidores RO y los conmutadores centrales (core) son 4 veces más rápidos que los enlaces de acceso a la granja de filtrado. Esta disparidad en la velocidad de los enlaces hace que el almacenamiento en el búfer de los conmutadores del EF sea inevitable. Cuando se produce la saturación de los búferes de los conmutadores EF por agregación de datos, se producen miles de pérdidas de paquetes por segundo.

Las pérdidas de paquetes introducen una penalización significativa en la latencia, especialmente al utilizar protocolos TCP/IP, donde se desencadenarán «*timeouts*» y retransmisiones. Por lo tanto, el EB utiliza un mecanismo de créditos a nivel de aplicación para evitar que los nodos del EF soliciten todos los fragmentos a la vez, reduciendo así el número de fuentes que contribuyen al *incast*. Finalmente, la aplicación del EB implementa el procesamiento por lotes

(*batching*), permitiendo la recolección de fragmentos asociados a múltiples eventos simultáneamente. El propósito del *batching* es reducir la tasa efectiva a la que las aplicaciones DH reciben peticiones, mejorando así el rendimiento del sistema RO, aunque también aumenta la magnitud de la ráfaga.

Este patrón de tráfico se ha modelado con precisión en el simulador. Los tamaños de los mensajes generados por cada tipo de detector se obtienen a partir de una distribución normal basada en las estimaciones actuales del tamaño de los fragmentos, lo que proporciona una aproximación precisa del tamaño total del evento. El procesamiento por lotes (*batching*) puede configurarse para aumentar este tamaño de mensaje mediante un factor proporcional al tamaño del lote especificado.

El tiempo de respuesta de una aplicación DH tras recibir una petición se modela mediante una distribución gamma con una media de 80 µs y una desviación estándar de 25 µs. En el sistema real, la aplicación DH es multihilo y puede gestionar múltiples peticiones de forma concurrente; por lo tanto, las peticiones entrantes también se gestionan de manera concurrente en las aplicaciones DH simuladas. El orden en que se asignan los eventos a los nodos EF se aleatoriza para evitar desequilibrios de carga.

Una vez finalizada la recolección de todos los fragmentos asociados a un evento determinado, se mide la latencia de recolección del evento completo. Esta latencia se denomina tiempo de recolección de datos (*data collection time*). El tiempo de recolección de datos aumenta en un valor constante de 0,75 ms, que representa la sobrecarga de procesamiento introducida por la aplicación de EB.

Dado que las versiones finales del software descritas aún están en desarrollo, se ha implementado en el modelo de simulación una versión alternativa del patrón de tráfico para imitar el comportamiento de los emuladores de software disponibles actualmente en el entorno de pruebas. En cuanto a la generación de tráfico, las principales diferencias entre el software del emulador del laboratorio y la versión final incluyen:

- Soporte para *batching*, que aún no está disponible en la versión del laboratorio. Esto se gestiona en el modelo de simulación, configurando el tamaño del lote en 1 durante la fase de validación.
- Generación de fragmentos en el sistema RO. Mientras que la aplicación final recuperará los datos de los detectores, la configuración del entorno de pruebas generará fragmentos de forma sintética. El tamaño de los fragmentos sigue una distribución normal y es uniforme en todas las aplicaciones de DH. Los parámetros que definen esta distribución son configurables tanto en el simulador como en la aplicación del laboratorio.

B. Características de control de congestión en TCP

El diseño base de la red DAQ de ATLAS asume el uso de los protocolos TCP/IP. Dada la dificultad de modelar la pila completa y los objetivos específicos del proyecto, solo se han implementado ciertas características clave de TCP/IP en el simulador. El objetivo es modelar los mecanismos que puedan tener un impacto directo y significativo en el patrón *incast*. Los mecanismos de detección de pérdidas de paquetes y de retransmisión no se han modelado en detalle, ya que el objetivo es operar en un régimen sin pérdidas o con una cantidad insignificante de ellas.

En consecuencia, las características implementadas incluyen: un protocolo de ventana deslizante, el algoritmo CUBIC [35] para aumentar el tamaño de la ventana y los paquetes de confirmación (ACKs). Las simulaciones comenzarán asumiendo un estado estable de la ventana, en la fase de *Congestion Avoidance*, con un tamaño de ventana inicial basado en el Round-Trip Time (RTT) y el Bandwidth Delay Product (BDP) de la topología. En caso de que un conmutador descarte paquetes, un mecanismo básico de retransmisión los reinyecta en la red, de modo que sigan contribuyendo al *incast* y la aplicación sea capaz de recibir todos los paquetes de cada mensaje. Como este mecanismo de retransmisión no es realista, no se tendrán en cuenta los escenarios que impliquen un uso frecuente del mismo (es decir, escenarios con un número significativo de pérdidas de paquetes).

C. Arquitectura de conmutador de búfer compartido (shared-buffer switch

Los conmutadores utilizados en la red *Dataflow* de DAQ de ATLAS tienen una arquitectura de búfer compartido. Por lo tanto, se ha implementado un modelo de conmutador de búfer compartido en el simulador. Con este propósito, se ha añadido un módulo Memory Management Unit (MMU). Este módulo se encarga de asignar y liberar la memoria necesaria para los paquetes entrantes y salientes. La ocupación del búfer compartido se calcula en unidades de tamaño fijo denominadas *cells*.

La MMU implementa *pipelines* de *ingress* y *egress*, que comprueban si los paquetes caben en sus respectivas particiones de búfer. Estas comprobaciones incluyen principalmente los límites de memoria de cada partición, así como la técnica de *dynamic threshold* [36], que limita el crecimiento de las colas a una fracción del búfer compartido, basándose en el número de colas que compiten por el búfer y en el parámetro configurable α. Este último es proporcional a la fracción del tamaño del búfer compartido que puede asignarse a una cola o a un grupo de prioridad determinado. Una configuración precisa del *dynamic threshold* es clave para este estudio de caso, ya que tiene un impacto significativo en el número de pérdidas de paquetes.

Específicamente, hemos implementado la técnica de *dynamic threshold* tal como se describe en 1, obtenida de [37].

$$\text{MaxEspacio} = \frac{\text{EspacioParteCompartida} \cdot \alpha}{1 + \alpha \cdot \text{ColasEnCompetencia}} \quad (1)$$
$$\alpha \in \{2^{\text{n}}\}_{\text{n}=-7}^{3}$$

Cabe destacar que α puede ser distinta para las reservas de entrada y de salida, lo que da lugar a

comportamientos del conmutador muy distintos en términos de pérdida de paquetes. Con esta implementación de *dynamic threshold*, una única cola congestionada (es decir, presente en el búfer compartido) podría crecer hasta el 50 % del tamaño de su reserva compartida, asumiendo un valor típico de α=1.

D. Topología

La topología final exacta de la red DAQ de ATLAS aún no se ha definido, ya que el número de servidores y *racks* en la granja EF variará según la estrategia elegida para la reconstrucción de trayectorias de partículas (actualmente se están evaluando algoritmos adaptados para CPUs, GPUs y FPGAs). No obstante, su estructura base, representada en la Figura 2, seguirá siendo la misma.

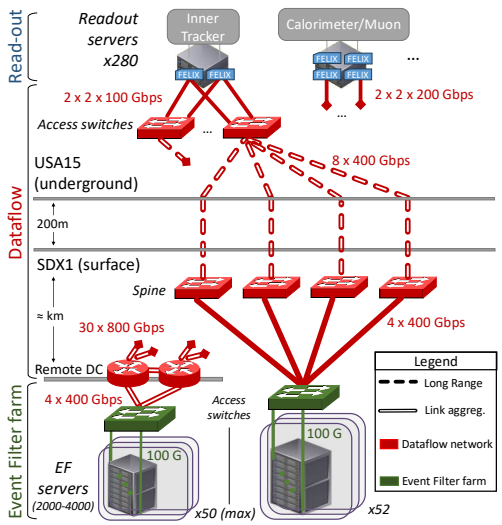


Fig. 2: Topología de red propuesta para la red DAQ de ATLAS en el HL-LHC.

La topología que queremos estudiar consiste en una red de tres etapas que comprende conmutadores RO, conmutadores centrales (*core*) y conmutadores EF. Tanto los conmutadores RO como los EF tendrán un *radix* de 32 y una velocidad de enlace de 400 Gbps, mientras que los conmutadores centrales tendrán un *radix* de 64 y una velocidad de enlace de 800 Gbps (los puertos pueden dividirse en cada conmutador mediante *splitting*). Los enlaces que conectan directamente con los servidores operarán a 100 Gbps. Los servidores RO estarán conectados a dos conmutadores *ToR* para aumentar la redundancia. Los conmutadores *ToR* RO que se conectan a los detectores que generan mensajes más grandes (es decir, conmutadores de *High Throughput*) se conectan a la capa central mediante enlaces agregados.

Mientras que la mitad del tráfico permanecerá en el emplazamiento experimental, la otra mitad se derivará a instalaciones de computación externas, accesibles a través de los conmutadores centrales. Por consiguiente, dado que nuestro principal objeto de estudio es el tamaño del búfer de los conmutadores centrales y *ToR* EF, se simula el tráfico de todo el sistema, pero solo se modela la topología del emplazamiento experimental. La computación externa se representará en el modelo de simulación mediante un enrutador de *deep-buffer* conectado a múltiples nodos de computación que simplemente recibirán cualquier tráfico entrante. Esto permitirá realizar simulaciones más rápidas y eficientes en términos de memoria.

En cuanto al enrutamiento, se utilizará Equal-Cost Multi-Path (ECMP) [38] para equilibrar la carga entre rutas mínimas. El modelo de simulación aún no soporta ECMP, pero en su lugar se configurará *packet spraying*, que proporcionará un efecto comparable en términos de equilibrio de carga, dado el patrón de tráfico y la topología de la red.

IV. Validación del modelo

Para validar el modelo de red DAQ de ATLAS, se construyó una configuración dedicada en el laboratorio. La comparación con el sistema real se realizó midiendo el tiempo de recolección de datos, el indicador más relevante del rendimiento del *Event Builder*.

A. Configuración del testbed

La Figura 3 muestra la configuración del hardware del entorno de pruebas utilizado en los experimentos. La configuración incluye 30 servidores RO (AMD EPYC 9354P, 32 núcleos, 64 hilos, 3,3 GHz) y 30 nodos EF (Xeon E5-2680v3, 12 núcleos, 24 hilos, 2,5 GHz). Los servidores están interconectados mediante una red Ethernet con una topología similar a la del sistema DAQ final previsto. Se utilizan conmutadores Juniper QFX5120 [39] como conmutadores de agregación de servidores RO (versión de 32 puertos a 100GbE) y EF (versión de 48 puertos a 25GbE), mientras que los conmutadores centrales (*core*) son Juniper QFX5130 [40]. El conmutador de agregación EF proporciona un ancho de banda de enlace ascendente (*uplink*) agregado de 800 Gbps y enlaces de acceso de 25 Gbps a cada uno de los 30 nodos EF. Los conmutadores *ToR* de los servidores RO se conectan cada uno al *core* con una capacidad de ancho de banda agregado de 1200 Gbps. Aunque el número de nodos representa una pequeña fracción del sistema final, esta configuración es capaz de imitar el comportamiento del sistema DAQ del *HL-LHC*, previsto en términos de frecuencia de asignación de eventos por bastidor (*rack*) y del uso del búfer del conmutador.

Para la configuración del software, todos los servidores ejecutan Alma Linux 9.6. Las mediciones se realizaron en el *testbed*, ajustando el software del *Event Builder* actual de Run3 para imitar el futuro *Event Builder* de Run4. La diferencia más sustancial entre ambas versiones radica en la forma en que se solicitan los datos. El *Event Builder* de Run3 solicita fragmentos de eventos de forma incremental al sistema RO a medida que los procesos de la EF avanzan a través de la cadena de algoritmos de selección. Por el contrario, el *Event Builder* de Run4 ensambla el evento completo por adelantado y lo sirve a la EF cuando una ranura de procesamiento (*slot*) queda disponible, desacoplando eficazmente la construcción de eventos del filtrado de eventos.

Este mismo comportamiento puede lograrse con el

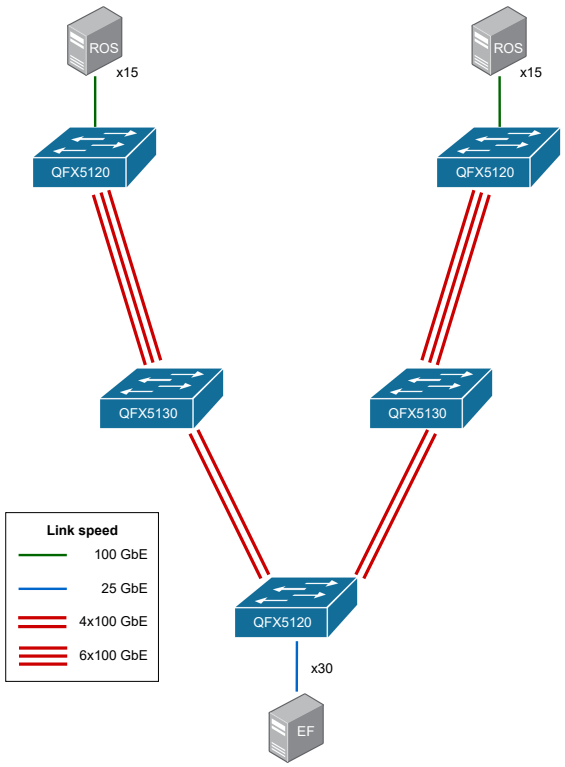


Fig. 3: Topología del demostrador del DAQ configurado en el laboratorio.

Event Builder de Run3, forzando la reconstrucción completa de todos los eventos. Además, el número de *slots* de procesamiento para un nodo determinado se redujo a uno, evitando la construcción de eventos en paralelo en el mismo nodo. Finalmente, como se introdujo en la Sección III-A, los tamaños de los fragmentos de datos sintéticos generados por el sistema RO son proporcionados por un emulador de software, que utiliza una distribución normal estocástica representativa del sistema real, mientras que el tiempo de procesamiento de los nodos EF se modeló con un valor constante de 1 milisegundo. Este valor puede ser significativamente mayor en el sistema real, pero el objetivo de nuestro experimento era emular el *Event Builder* de Run4, que opera cerca de una tasa constante y casi independientemente del estado instantáneo de la EF.

Los experimentos se llevaron a cabo variando tres parámetros relevantes dentro de un amplio rango de valores (véase un subconjunto de las mediciones en la Figura 4): el tamaño del evento, la tasa de eventos por bastidor (*rack*) de servidores EF y los créditos de petición asignados a cada proceso individual del *Event Builder*. Mientras que la modificación del tamaño del evento requirió una reconfiguración manual de todo el sistema, el escaneo de la tasa y del valor de los créditos pudo automatizarse por completo. Para cada medición, consideramos un periodo de calentamiento (*warm-up*) de 30 segundos y medimos el tiempo promedio de recolección de datos y el total de pérdidas de paquetes durante un intervalo de 30 segundos. También realizamos comprobaciones de integridad (*sanity checks*), como mediciones de la tasa real alcanzada, para descartar escenarios críticos en los que las CPUs de los *Event Builders* no pudieran hacer frente a tasas de petición muy elevadas.

Cada prueba se repitió tres veces y se promediaron todos los resultados. Las desviaciones estándar alcanzadas oscilan entre 1 µs y 100 µs. También se realizó una búsqueda de valores atípicos (*outliers*) para evaluar la consistencia de los datos e identificar escenarios problemáticos. Solo un pequeño subconjunto de mediciones presentó un comportamiento inconsistente en escenarios extremos (tasa de eventos excesiva, gran tamaño de evento o un alto número de créditos de petición) y no se consideró para la validación del modelo.

B. Resultados

Para validar el modelo de simulación, comparamos los tiempos de recolección de datos medidos en el *testbed* del laboratorio con los obtenidos en el simulador mediante varios escenarios experimentales. Estos resultados se presentan en la Figura 4. Los escenarios del *testbed* utilizan tasas de eventos por bastidor (*rack*) que oscilan entre 4 y 8 kHz, lo que representa el límite superior de rendimiento de la configuración actual de hardware y software. Para cada tasa, evaluamos el tiempo promedio de recolección de datos para tres tamaños de evento específicos: 2,038 MB, 3,486 MB y 4,946 MB —este último se aproxima estrechamente al tamaño de evento de 4,6 MB estimado para el sistema HL-LHC final—.

Además, medimos el tiempo promedio de recolección de datos para cada par de tasa y tamaño de evento a través de un rango de valores de créditos a nivel de aplicación, de 1 a 30 (correspondientes al número de servidores RO). Los resultados demuestran que el simulador reproduce con precisión los tiempos de recolección de datos medidos en el sistema real en todos los escenarios evaluados. Si bien los valores simulados son generalmente idénticos o ligeramente inferiores a las mediciones de laboratorio, algunas configuraciones que utilizan un solo crédito muestran latencias más altas en la simulación. Esta discrepancia se debe al alto número de ciclos de petición-respuesta (*request-response*) (30) necesarios para recopilar un evento completo. En consecuencia, se produce un aumento del error acumulado debido a la estimación de las latencias de procesamiento en los servidores RO. Sin embargo, una configuración de un solo crédito no representa un escenario realista, ya que el rendimiento se ve seriamente degradado por la naturaleza secuencial de las solicitudes. Por lo tanto, las desviaciones en estos casos específicos se consideran insignificantes, ya que en la práctica siempre se preferirán las configuraciones del sistema que ofrezcan latencias más bajas manteniendo un régimen de bajas pérdidas (*low-drop*).

La figura también muestra cómo el aumento del tamaño del evento se traduce en latencias más altas para un número determinado de créditos. Esta penalización se debe al tiempo adicional necesario para

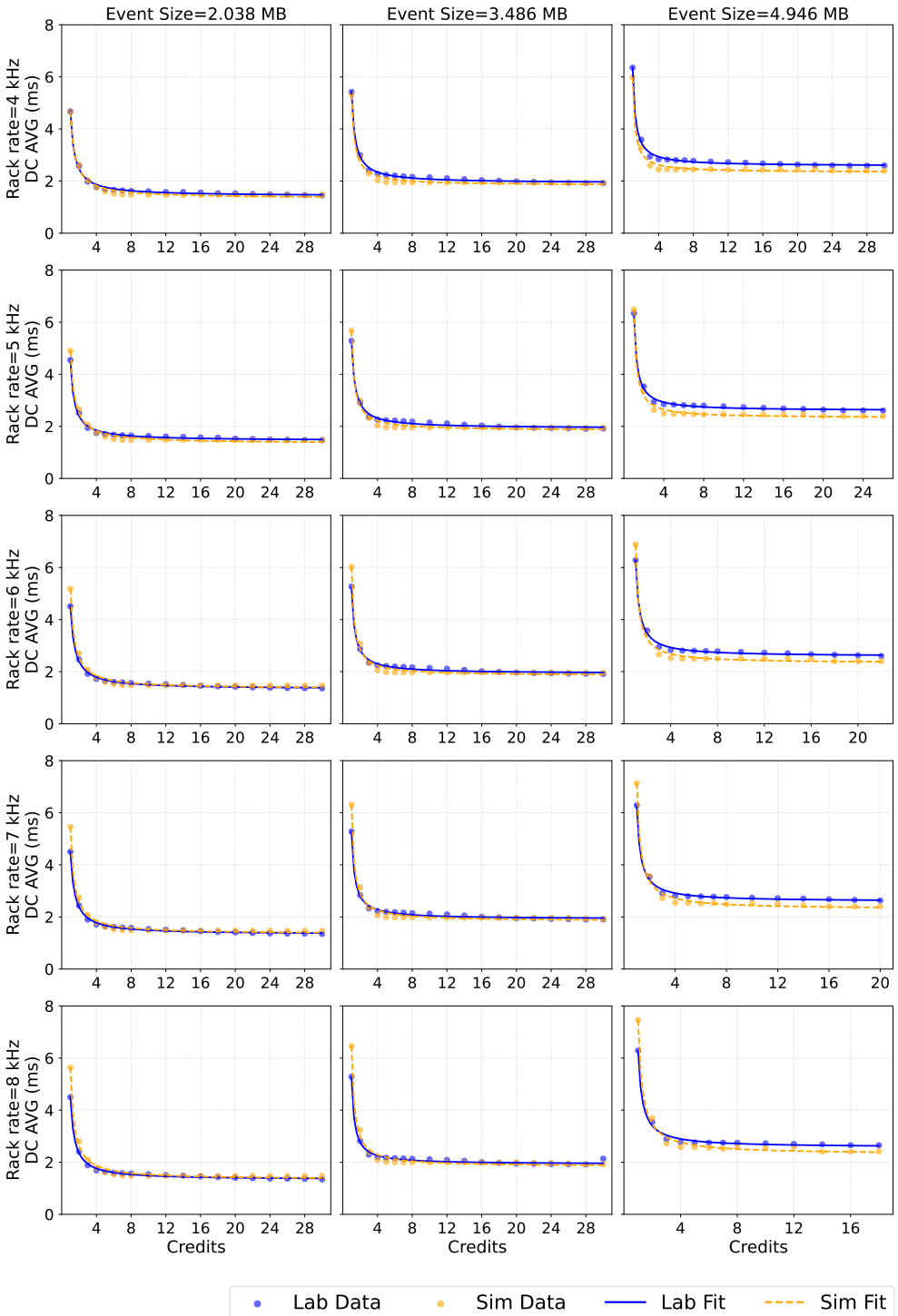


Fig. 4: Tiempo de recolección promedio medido en el laboratorio y en el simulador.

transferir más datos a través de enlaces de la misma velocidad, así como a la contención en el conmutador EF. Por el contrario, aumentar la tasa de la aplicación no tiene un impacto notable en las latencias bajo un régimen de bajas pérdidas de paquetes.

V. Conclusiones

El modelo de simulación presentado en este artículo ha demostrado su capacidad para replicar con alta fidelidad las mediciones de laboratorio, en particular la ocupación del búfer y las latencias de recolección de datos. Además, la escalabilidad del *framework* lo consolida como una herramienta robusta para evaluar las configuraciones de hardware y software esenciales para el diseño del sistema DAQ del *HL-LHC*.

El trabajo futuro se centrará en ampliar las capacidades del entorno de simulación para adaptarlo a los requisitos de red en constante evolución. Específicamente, planeamos extender el modelo con una implementación de TCP de alta fidelidad que incluya una fase de *slow-start*, un protocolo de retransmisión robusto basado en *timeouts* y en la detección de paquetes fuera de orden. Estas mejoras aumentarán significativamente la precisión de la simulación en escenarios congestionados donde se producen pérdidas de paquetes. Además, el mecanismo existente de

Notificación Explícita de Congestión (ECN) [41], actualmente implementado en arquitecturas *CIOQ*, se adaptará a nuestro modelo de conmutador de búfer compartido.

Agradecimientos

Este trabajo ha sido financiado por la Junta de Comunidades de Castilla-La Mancha bajo el proyecto Tetra2 (SBPLY/21/180225/000103), y por la Universidad de Castilla-La Mancha bajo el proyecto 2023-GRIN-34056.

Referencias

[1] ATLAS Collaboration, "The ATLAS Experiment at the CERN Large Hadron Collider," *JINST*, vol. 3, pp. S08003, 2008.

[2] Luis Medina Medrano, Andrea Apollonio, Gianluigi Arduini, Oliver Brüning, Massimo Giovannozzi, Stefania Papadopoulou, Yannis Papaphilippou, Stefano Redaelli, and Rogelio Tomás, "New High Luminosity LHC Baseline and Performance at Ultimate Energy," in *9th International Particle Accelerator Conference*, 6 2018.

[3] Yongmao Ren, Yu Zhao, Pei Liu, Ke Dou, and Jun Li, "A survey on tcp incast in data center networks," *International Journal of Communication Systems*, vol. 27, 08 2014.

[4] Yanpei Chen, Rean Griffith, Junda Liu, Randy H. Katz, and Anthony D. Joseph, "Understanding tcp incast throughput collapse in datacenter networks," in *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, New York, NY, USA, 2009, WREN '09, p. 73–82, Association for Computing Machinery.

[5] Wen Chen, Fengyuan Ren, Jing Xie, Chuang Lin, Kevin Yin, and Fred Baker, "Comprehensive understanding of tcp incast problem," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 1688–1696.

[6] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan, "Data center TCP (DCTCP)," in *Proceedings of the ACM SIGCOMM 2010 conference*, New York, NY, USA, Aug. 2010, SIGCOMM '10, pp. 63–74, Association for Computing Machinery.

[7] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang, "Congestion Control for Large-Scale RDMA Deployments," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 523–536, Aug. 2015.

[8] CMS Collaboration, "The cms experiment at the cern large hadron collider," *JINST*, vol. 3, pp. S08004, 2008.

[9] ALICE Collaboration, "The alice experiment at the cern lhc," *JINST*, vol. 3, pp. S08002, 2008.

[10] LHCb Collaboration, "The lhcb detector at the lhc," *JINST*, vol. 3, pp. S08005, 2008.

[11] Tommaso Colombo, Paolo Durante, Domenico Galli, Matteo Manzali, Umberto Marconi, Niko Neufeld, Flavio Pisani, Rainer Schwemmer, and Sébastien Valat, "Flit-level infiniband network simulations of the daq system of the lhcb experiment for run-3," *IEEE Transactions on Nuclear Science*, vol. 66, no. 7, pp. 1159–1164, 2019.

[12] Flavio Pisani, Tommaso Colombo, Niko Neufeld, Umberto Marconi, Rafal Krawczyk, Domenico Galli, Rainer Schwemmer, and Paolo Durante, "Network simulation of a 40 mhz event building system for the lhcb experiment," *EPJ Web of Conferences*, vol. 245, 11 2020.

[13] Linda Vickovic, Sven Gotovac, and Stipe Celar, "Simulation-based performance analysis of the alice mass storage system," *International journal of simulation modelling*, vol. 15, no. 1, pp. 70–82, 2016.

[14] G Bauer, U Behrens, K Biery, J Branson, E Cano, H Cheung, M Ciganek, S Cittolin, J A Coarasa, C Deldicque, E Dusinberre, S Erhan, F Fortes Rodrigues, D Gigi, F Glege, R Gomez-Reino, J Gutleber, D Hatton, J F Laurens, J A Lopez Perez, F Meijers, E Meschi, A Meyer, R Mommsen, R Moser, V O'Dell, A Oh, L B Orsini, V Patras, C Paus, A Petrucci, M Pieri, A Racz, H Sakulin, M Sani, P Schieferdecker, C Schwick, D Shpakov, S Simon, K Sumorok, and M Zanetti, "The cms data acquisition system software," *Journal of Physics: Conference Series*, vol. 219, no. 2, pp. 022011, apr 2010.

[15] G Bauer, U Behrens, M Bowen, J Branson, S Bukowiec, S Cittolin, J A Coarasa, C Deldicque, M Dobson, A Dupont, S Erhan, A Flossdorf, D Gigi, F Glege, R Gomez-Reino, C Hartl, J Hegeman, A Holzner, Y L Hwong, L Masetti, F Meijers, E Meschi, R K Mommsen, V O'Dell, L Orsini, C Paus, A Petrucci, M Pieri, G Polese, A Racz, O Raginel, H Sakulin, M Sani, C Schwick, D Shpakov, S Simon, A Spataru, and K Sumorok, "Operational experience with the cms data acquisition system," *Journal of Physics: Conference Series*, vol. 396, no. 1, pp. 012007, dec 2012.

[16] Tommaso Colombo, Holger Fröning, Pedro Javier García, and Wainer Vandelli, "Modeling a large data-acquisition network in a simulation framework," in *2015 IEEE International Conference on Cluster Computing*, 2015, pp. 809–816.

[17] Tommaso Colombo, Holger Fröning, Pedro Garcìa, and Wainer Vandelli, "Optimizing the data-collection time of a large-scale data-acquisition system through a simulation framework," *The Journal of Supercomputing*, vol. 72, 12 2016.

[18] Levente Mészáros, Andras Varga, and Michael Kirsche, "INET Framework," in *Recent Advances in Network Simulation: The OMNeT++ Environment and its Ecosystem*, Antonio Virdis and Michael Kirsche, Eds., pp. 55–106. Springer International Publishing, Cham, 2019.

[19] Mikel Pozo Astigarraga, Matias Bonaventura, James Maple, Ezequiel Pecker Marcosig, Giacomo Levrini, and Rodrigo Castro, "Benchmarking data acquisition event building network performance for the atlas hl-lhc upgrade," *EPJ Web of Conferences*, vol. 295, 05 2024.

[20] ns-3 Consortium, "ns-3 Network Simulator," `https://www.nsnam.org`, Accessed: 2026-01-06.

[21] Ming-yu Hsieh, Rolf Riesen, Kevin Thompson, William Song, and Arun Rodrigues, "SST: A Scalable Parallel Framework for Architecture-Level Performance, Power, Area and Thermal Simulation," *The Computer Journal*, vol. 55, pp. 181–191, Feb. 2012.

[22] Pedro Yebenes Segura, Jesus Escudero-Sahuquillo, P.J. Garcia, and Francisco Quiles, "Towards Modeling Interconnection Networks of Exascale Systems with OMNeT++," 02 2013, pp. 203–207.

[23] András Varga and Rudolf Hornig, "An overview of the OMNeT++ simulation environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Brussels, BEL, 2008, Simutools '08, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[24] Gabriel Gomez-Lopez, Miguel Sánchez de la Rosa, Jesús Escudero-Sahuquillo, Pedro Javier García, Francisco J. Quiles, and Pierre-Axel Lagadec, "Hybrid congestion control for bxi-based interconnection networks," in *Euro-Par 2024: Parallel Processing - 30th European Conference on Parallel and Distributed Processing, Madrid, Spain, August 26-30, 2024, Proceedings, Part II*, Jesús Carretero, Sameer Shende, Javier García-Blas, Ivona Brandic, Katzalin Olcoz, and Martin Schreiber, Eds. 2024, vol. 14802 of *Lecture Notes in Computer Science*, pp. 242–256, Springer.

[25] Nikolaos Tampouratzis, Ioannis Papaefstathiou, Gabriel Gomez-Lopez, Miguel Sánchez de la Rosa, Jesús Escudero-Sahuquillo, and Pedro Javier García, "Distributed fast and accurate simulation platform for advanced ARM- and risc-v-based HPC systems," *J. Supercomput.*, vol. 81, no. 16, pp. 1484, 2025.

[26] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, New York, NY, USA, 2008, SIGCOMM '08, p. 63–74, Association for Computing Machinery.

[27] Sabine R. Ohring, Maximilian Ibel, Sajal K. Das, and Mohan J. Kumar, "On generalized fat trees," in *Proceedings of 9th International Parallel Processing Symposium*, Apr. 1995, pp. 37–44.

[28] John Kim, William J. Dally, Steve Scott, and Dennis Abts, "Technology-driven, highly-scalable dragonfly topology," *2008 International Symposium on Computer Architecture*, pp. 77–88, 2008.

[29] Alexander Shpiner, Zachy Haramaty, Saar Eliad, Vladimir

Zdornov, Barak Gafni, and Eitan Zahavi, "Dragonfly+: Low cost topology for scaling datacenters," *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, pp. 1–8, 2017.

[30] Roberto Peñaranda, Crispín Gómez Requena, María Engracia Gómez, Pedro López, and José Duato, "A new family of hybrid topologies for large-scale interconnection networks," in *11th IEEE International Symposium on Network Computing and Applications, NCA 2012, Cambridge, MA, USA, August 23-25, 2012*. 2012, pp. 220–227, IEEE Computer Society.

[31] Teresa Nachiondo, Jose Flich, and Jose Duato, "Buffer Management Strategies to Reduce HoL Blocking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 6, pp. 739–753, 2010.

[32] Jesus Escudero-Sahuquillo, Pedro J. Garcia, Francisco J. Quiles, Sven-Arne Reinemo, Tor Skeie, Olav Lysne, and Jose Duato, "A new proposal to deal with congestion in InfiniBand-based fat-trees," *Journal of Parallel and Distributed Computing*, vol. 74, no. 1, pp. 1802–1819, 2014.

[33] Wei Lin Guay, Bartosz Bogdanski, Sven-Arne Reinemo, Olav Lysne, and Tor Skeie, "vFtree - a Fat-Tree Routing Algorithm Using Virtual Lanes to Alleviate Congestion," in *2011 IEEE International Parallel & Distributed Processing Symposium*, 2011, pp. 197–208.

[34] Pedro Yebenes Segura, Jesus Escudero-Sahuquillo, Crispin Gomez Requena, Pedro Javier Garcia, Francisco J. Quiles, and Jose Duato, "BBQ: A Straightforward Queuing Scheme to Reduce HoL-Blocking in High-Performance Hybrid Networks," in *Euro-Par 2013 Parallel Processing*, Felix Wolf, Bernd Mohr, and Dieter an Mey, Eds., Berlin, Heidelberg, 2013, pp. 699–712, Springer Berlin Heidelberg.

[35] Sangtae Ha, Injong Rhee, and Lisong Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[36] Abhijit K. Choudhury and Ellen L. Hahne, "Dynamic queue length thresholds for shared-memory packet switches," *IEEE/ACM Transactions on Networking*, vol. 6, no. 2, pp. 130–140, 1998.

[37] Parthipan TS, "QFX5K-Series Switches Packet Buffer Architecture," `https://community.juniper.net/blogs/parthipan-ts/2025/04/25/qfx5k-series-switches-packet-buffer-architecture`, 2025, Last accessed: 10-11-2025.

[38] Christian Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," Nov. 2000.

[39] Juniper Networks, "Qfx5120 datasheet," `https://www.hpe.com/psnow/doc/a00150810enw`, 2018, Last accessed: 11-11-2025.

[40] Juniper Networks, "Qfx5130 datasheet," `https://www.hpe.com/psnow/doc/a00150812enw`, 2022, Last accessed: 07-01-2026.

[41] K. K. Ramakrishnan, Sally Floyd, and David L. Black, "The addition of explicit congestion notification (ECN) to IP," 2001.

Tecnologías clúster, plataformas distribuidas, BigData y Deep Learning (1/4)

Modelando la gestión de datos en el cómputo continuo mediante grafos de conocimiento

Dante D. Sánchez-Gallegos [1], Gabriele Morabito[2], Maria Fazio[2] y Jesus Carretero[1]

Resumen— **En los últimos años, el procesamiento y la gestión de datos se han descentralizado. Esto ha impulsado el paradigma del cómputo continuo (*computing continuum*), en el que la información se adquiere en dispositivos de borde y se procesa en la nube o en infraestructuras de alto rendimiento. En este contexto, garantizar la robustez de la transferencia y el almacenamiento de los datos resulta crucial para el correcto funcionamiento de los sistemas. Los sistemas de almacenamiento de área amplia responden a esta necesidad al proporcionar una capa unificada que abarca múltiples ubicaciones físicas. Sin embargo, su administración se vuelve compleja debido a la variabilidad de las cargas de trabajo, la infraestructura y las políticas de datos. Para abordar este desafío, en este artículo se propone un enfoque que modela los sistemas de almacenamiento de área amplia como Grafos de Conocimiento (KG, por sus siglas en inglés). La propuesta representa semánticamente las capas de datos e infraestructura mediante una ontología, lo que facilita su análisis mediante métricas de grafos para el diseño de sistemas dinámicos e impulsa la creación de flujos de trabajo autoadaptativos. Para validar este modelo, se ha integrado en el sistema de almacenamiento DynoStore. Los resultados experimentales demuestran que el flujo de trabajo basado en KG mejora significativamente la eficiencia en la toma de decisiones, optimizando el acceso y la utilización de los recursos de almacenamiento.**

Palabras clave— **Almacenamiento de datos, grafos de conocimiento, cómputo continuo, balanceo de carga**

I. Introducción

En los últimos años han surgido nuevos paradigmas para la gestión de datos en entornos geográficamente distribuidos. Ejemplos de estos paradigmas son el cómputo continuo y el aprendizaje federado, los cuales promueven el procesamiento de la información allí donde se genera [1]. Los sistemas de almacenamiento de área amplia apoyan estas arquitecturas mediante una capa de datos ubicua y transparente que abarca múltiples ubicaciones [2]. De esta forma, ofrecen protocolos para transportar y almacenar información, lo que permite a las aplicaciones acceder de manera transparente a los datos provenientes de fuentes remotas.

Sin embargo, a medida que los sistemas distribuidos se expanden en escala y volumen de datos, su administración se vuelve cada vez más compleja. Esto resulta especialmente problemático en entornos de cómputo continuo, donde las cargas de trabajo y los patrones de acceso son dinámicos [3]. Además, en estos escenarios, nuevos nodos de almacenamiento pueden unirse al sistema, mientras que los existentes pueden abandonarlo debido a fallas; al mismo tiempo, los datos pueden migrar entre nodos o replicarse a través de infraestructuras heterogéneas. En consecuencia, los modelos de gestión estáticos a menudo no logran hacer frente a esta fluidez, lo que da lugar a una utilización ineficiente de los recursos y a mayores costes operativos.

Por consiguiente, es fundamental que los sistemas de almacenamiento subyacentes puedan adaptarse a dicha variabilidad. Esto plantea la necesidad de soluciones dinámicas y elásticas, capaces de ajustar su estructura y comportamiento en respuesta a los cambios del entorno o de la carga de trabajo [4]. Para ello, los Grafos de Conocimiento (KG, por sus siglas en inglés) proporcionan un modelo semántico de entidades y relaciones que permite integrar múltiples fuentes de información heterogéneas y representar la evolución dinámica del estado del sistema. Esta expresividad faculta a los KG para capturar tanto los aspectos estructurales como los contextuales del almacenamiento de área amplia, incluyendo métricas de rendimiento y políticas de calidad de servicio. Mediante el uso de ontologías y mecanismos de razonamiento, los KG facilitan la extracción de nuevo conocimiento a partir de los datos existentes, permitiendo optimizar tareas de administración avanzadas y la aplicación de políticas. Como resultado, los KG constituyen una base sólida para la gestión adaptativa e inteligente de infraestructuras distribuidas a gran escala.

En este trabajo se presenta KAGIO[1], una solución que aborda el problema de la gestión dinámica de recursos en los sistemas de almacenamiento de área amplia mediante el uso de KG para habilitar una administración adaptativa y basada en el conocimiento. Se propone un enfoque semántico que modela tanto las capas de datos como las de infraestructura, lo que facilita un razonamiento continuo sobre el estado del sistema y una adaptación dinámica a cargas de trabajo y entornos cambiantes. Esta propuesta integra el modelado ontológico, el análisis basado en grafos y flujos de trabajo orientados al conocimiento para apoyar la toma de decisiones autónoma y la autooptimización de la infraestructura.

Para validar KAGIO, se ha empleado el sistema de almacenamiento DynoStore [5], el cual permite salvaguardar información en distintos sitios geográficamente distribuidos y proporciona una plataforma flexible para explorar mecanismos adaptativos en entornos heterogéneos. Como parte de esta integración, KAGIO opera como una capa de control transver-

[1]Dpto. de Informática, Universidad Carlos III de Madrid, e-mails: dantsanc@pa.uc3m.es. y jcarrete@inf.uc3m.es.

[2]Universidad de Messina, Italia, e-mail: gabriele.morabito@unime.it.

[1]Knowledge-Aware Graph Interoperability for Orchestrating Continuum Data

sal en DynoStore, lo que viabiliza la representación semántica de las interacciones entre la infraestructura y los datos, así como el razonamiento lógico para la toma de decisiones en su gestión.

Las principales contribuciones de esta investigación son:

1. **Una ontología para sistemas de almacenamiento de área amplia** que captura los aspectos estructurales, operativos y contextuales de estas infraestructuras.
2. **Un modelo de análisis de elasticidad** que cuantifica la adaptabilidad de los sistemas de almacenamiento mediante métricas basadas en la centralidad.
3. **Un flujo de trabajo para la gestión de datos basada en el conocimiento capturado por KAGIO**. Este proceso supervisa las condiciones del sistema, infiere oportunidades de adaptación y activa acciones de reconfiguración para mantener el rendimiento bajo cargas de trabajo dinámicas.
4. **Una estrategia de replicación de datos adaptativa apoyada en el conocimiento** que considera los patrones de acceso a los objetos para equilibrar el uso de los recursos entre los distintos sitios.

El resto del artículo está organizado de la siguiente manera. La Sección II revisa el estado del arte y ofrece los antecedentes teóricos fundamentales. La Sección III detalla la arquitectura de DynoStore, el sistema utilizado para la validación. Por su parte, la Sección IV describe el diseño y funcionamiento de KAGIO. La Sección V expone la evaluación experimental de la propuesta. Finalmente, la Sección VI resume las conclusiones del artículo y plantea posibles líneas de trabajo futuro.

II. Trabajos relacionados y antecedentes

Sistemas de almacenamiento elásticos. En la literatura, el problema de la elasticidad en sistemas de almacenamiento se centra principalmente en la capacidad de reescalar la infraestructura, es decir, añadir o eliminar nodos y recursos de forma dinámica sin interrupciones. Por ejemplo, en [6] se introducen puntos de referencia para medir la escalabilidad de un sistema distribuido, centrándose en el rendimiento de las operaciones a nivel de nodo. De manera similar, en [7] se modela la duración de la operación de desmantelamiento para evaluar la viabilidad de la elasticidad como principio de diseño para los sistemas de archivos distribuidos. Otras investigaciones formalizan el reto de escalar un sistema como un problema tradicional de optimización multiobjetivo, considerando criterios como el balanceo de la carga de trabajo y la duración operativa [8]. Además, los sistemas de archivos ad hoc permiten la gestión elástica de los datos, adaptando el despliegue del almacenamiento a las necesidades de la aplicación en tiempo de ejecución [9], [10].

Otro conjunto de trabajos, independiente pero relacionado, aborda la ubicación y la replicación de datos mediante grafos para representar la topología de la red y las métricas de acceso. En [11] se propone una política para la ubicación de réplicas de archivos, modelando el sistema como una red compleja ponderada en el dominio del *Fog Computing*. En dicho trabajo se utilizan índices de centralidad y particiones de grafos para seleccionar los nodos de almacenamiento donde se replicará la información.

Grafos de conocimiento para modelar sistemas a gran escala. Paralelamente a los modelos de grafos tradicionales, los Grafos de Conocimiento (KG, por sus siglas en inglés) están emergiendo como una herramienta para modelar sistemas complejos y con múltiples relaciones en diversos dominios [12]. Trabajos recientes demuestran que los KG pueden mejorar sustancialmente la orquestación de redes de computadores y servicios al vincular los requisitos empresariales de alto nivel con configuraciones de red de bajo nivel [13], [14]. En el cómputo continuo, los KG actúan como modelos de recursos unificados, enlazando las entidades de computación y de almacenamiento con las aplicaciones a las que brindan soporte. Además, en combinación con enfoques de aprendizaje automático basados en grafos, como GraphSAGE, permiten la supervisión en tiempo real y la detección de anomalías, con el fin de mejorar el uso de la infraestructura [12].

Otra aplicación de los KG se da en los paradigmas de infraestructura como código, donde las representaciones semánticas del conocimiento permiten mejorar la automatización y la escalabilidad en entornos de nube, en el borde y en supercomputadores [15]. Por último, en la computación de alto rendimiento, los KG y el análisis de grafos se han aplicado para modelar interacciones complejas entre usuarios, aplicaciones y recursos, lo que permite analizar su productividad y reproducibilidad [16], [17], [18], [19].

A pesar de su uso generalizado en las redes de computadores, la nube y entornos de cómputo de alto rendimiento, los KG aún no se han explorado como técnica para habilitar o mejorar la elasticidad en los sistemas de almacenamiento distribuidos. Los trabajos existentes se centran en la elasticidad y la reconfiguración mediante enfoques procedimentales o de optimización, dejando abierta la posibilidad de investigar cómo las representaciones semánticas basadas en grafos podrían apoyar la elasticidad contextual basada en el conocimiento.

Definiciones de grafos de conocimiento. Los Grafos de Conocimiento (KG) son estructuras de datos basadas en grafos en las que los nodos y las aristas pueden tener diversas etiquetas y propiedades [20]. Los KG se pueden definir formalmente de la siguiente manera:

Definition 1 (Grafo de conocimiento (KG))
Un grafo de conocimiento es una tupla $KG = (N, A, R, \lambda_N)$, donde:

- N es el conjunto de nodos en el grafo y cada nodo representa una entidad en el sistema;
- R es el conjunto de tipos de relaciones entre en-

tidades;
- $A \subseteq N \times R \times N$ es el conjunto de aristas dirigidas, que representan las relaciones entre los nodos;
- $\lambda_N : N \to 2^K$ es una función que asigna cada nodo a una o más etiquetas de tipo en el conjunto K.

Además, los KG pueden caracterizarse mediante sus unidades atómicas, que constituyen los componentes estructurales más pequeños que definen el grafo.

Definition 2 (Unidad atómica de un KG) La unidad atómica de un KG es una tripleta [21], compuesta por:
- un nodo de origen etiquetado m;
- un nodo de destino etiquetado n;
- una arista dirigida y etiquetada r de m a n.

III. Caso de uso: DynoStore

Como caso de uso para el desarrollo de KAGIO se eligió DynoStore, un sistema de almacenamiento distribuido a gran escala desarrollado por el grupo ARCOS de la Universidad Carlos III de Madrid. Es importante aclarar que KAGIO es independiente de DynoStore y, por lo tanto, podría utilizarse en sistemas de almacenamiento similares.

El componente básico de construcción en DynoStore es una abstracción denominada *contenedor de datos*. Estos se despliegan sobre sistemas de almacenamiento existentes y crean una red distribuida de almacenamiento. Cada contenedor implementa funciones para cargar, descargar, buscar y eliminar objetos del sistema subyacente. Además, DynoStore ofrece servicios de balanceo de carga y de confiabilidad.

El servicio de confiabilidad implementa un esquema de redundancia que divide cada objeto en n fragmentos, los cuales contienen suficiente información para reconstruir el objeto original a partir de cualquier conjunto de k fragmentos ($k < n$) [22]. Por su parte, el servicio de balanceo distribuye estos fragmentos entre los contenedores de datos considerando su utilización actual [5]. En este enfoque, se da preferencia a los contenedores con un factor de utilización bajo frente a los que tienen uno alto. El objetivo es mantener un porcentaje de utilización similar en todos los nodos, independientemente de su capacidad de almacenamiento total.

Los contenedores de datos pueden configurarse para limitar su almacenamiento y uso de memoria, evitando así la sobrecarga de los equipos. Estos límites pueden ajustarse posteriormente por la organización, según la capacidad que deseen asignar al sistema de almacenamiento de área amplia.

Limitaciones de DynoStore. A pesar de su flexibilidad, DynoStore presenta múltiples limitaciones comunes a otros sistemas de la literatura. La primera es que la ubicación física (p. ej., el nodo de almacenamiento) de los objetos es estática y no cambia con el tiempo. En consecuencia, DynoStore carece de la flexibilidad necesaria para adaptarse a las variaciones en las cargas de trabajo o en los patrones de acceso a los datos, ya que el balanceador ubica los objetos basándose únicamente en métricas de utilización, ignorando su popularidad posterior. Como resultado, si un conjunto de objetos almacenados en un mismo contenedor se vuelve popular, dicho contenedor recibirá más solicitudes que otros con una utilización similar pero con información menos demandada. Una posible solución es la reubicación o replicación de los objetos populares para distribuir los accesos entre una mayor cantidad de nodos. Sin embargo, esto requiere identificar la popularidad tanto de los contenedores como de los objetos, lo cual resulta complejo a medida que el sistema crece en volumen.

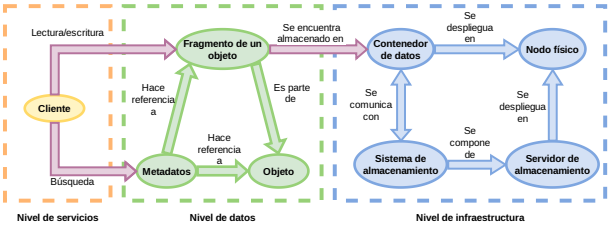


Fig. 1: Modelo de grafo de conocimiento del sistema de almacenamiento DynoStore.

A. Representación semántica de DynoStore como ontología

El objetivo principal de KAGIO es mitigar las limitaciones anteriores integrando DynoStore con un modelo de razonamiento basado en Grafos de Conocimiento y métricas de centralidad. En concreto, el propósito es crear un sistema elástico y distribuido que haga frente a los cambios en la carga de trabajo.

Para ello, se diseñó una ontología que permite visualizar la representación semántica de los componentes de DynoStore. La Figura 1 muestra esta ontología, la cual modela las entidades, las relaciones y la semántica operativa del ecosistema distribuido. Este esquema sirve como base fundamental para los Grafos de Conocimiento en KAGIO, permitiendo la descripción, supervisión y razonamiento sobre el comportamiento de la infraestructura de área amplia. El modelo se ha formalizado mediante una estructura de nodos y relaciones, donde los nodos representan las entidades principales y las aristas codifican las interacciones operativas y estructurales. Tanto los nodos como las aristas incluyen propiedades tipificadas que capturan parámetros cuantitativos y cualitativos, lo que facilita el modelado descriptivo y el análisis del sistema en tiempo de ejecución.

La Tabla I detalla las ocho entidades principales de la ontología, agrupadas en tres capas: (i) componentes a nivel de infraestructura, (ii) a nivel de datos y (iii) a nivel de servicio (representadas respectivamente por los círculos azules, verdes y amarillos en la Figura 1).

La Tabla II muestra las principales interacciones de la ontología. Las relaciones entre entidades de la misma capa se modelan como aristas dirigidas, divididas en dependencias de *nivel de infraestructura* y *nivel de datos* (flechas azules y verdes de la Figura 1, respectivamente). Las dependencias entre diferentes capas se modelan mediante relaciones intercapa

Tabla I: Resumen de las entidades definidas en la ontología.

Capa de entidad	Nombre de la entidad	Descripción	Propiedades principales
Nivel de infraestructura	**Nodo físico**	Representa el hardware o la máquina virtual que aloja los servicios de almacenamiento y de computación.	Núcleos de CPU, utilización de CPU, capacidad de memoria, memoria utilizada, capacidad de almacenamiento, almacenamiento utilizado, rendimiento de escritura/lectura.
	Servidor de almacenamiento	Gestiona las operaciones de almacenamiento local y proporciona interfaces de acceso a las capas superiores.	Memoria utilizada, utilización de CPU, rendimiento de escritura/-lectura.
	Sistema de almacenamiento	Abstracción del mecanismo de persistencia subyacente (por ejemplo, una base de datos o un sistema de archivos).	Tipo, tecnología.
	Contenedor de datos	Unidad de implementación lógica que proporciona una abstracción sobre sistemas de almacenamiento heterogéneos.	Capacidad de memoria, memoria utilizada, utilización de la CPU.
Nivel de datos	**Objeto de datos**	Elemento de datos de alto nivel almacenado en un contenedor.	Tamaño, umbral de reconstrucción (número de fragmentos necesarios para la recuperación), cifrado, comprimido.
	Fragmento de objeto	Fragmento de un objeto de datos generado mediante replicación o codificación de borrado.	Tamaño, cifrado.
	Metadatos	Descriptor semántico que vincula objetos de datos, fragmentos y ubicaciones de almacenamiento.	Tamaño.
Nivel de servicio	**Cliente**	Interfaz a través de la cual los usuarios o las aplicaciones interactúan con el sistema de almacenamiento distribuido.	–

Tabla II: Resumen de los tipos de relación en la ontología.

Tipo de relación	Nombre	Origen → Destino	Descripción / Anotación
Nivel de infraestructura	`se despliega en`	Contenedor de datos / Servidor de almacenamiento → Nodo físico	Asocia los componentes lógicos con sus nodos físicos de almacenamiento.
	`se compone de`	Sistema de almacenamiento → Servidor de almacenamiento	Conecta un sistema de almacenamiento con los servidores que lo integran.
	`comunicación`	Contenedor de datos ↔ Sistema de almacenamiento nativo	Representa el intercambio de datos.
Nivel de datos	`forma parte de`	Fragmento de objeto → Objeto de datos	Vincula fragmentos con su objeto de datos principal.
	`se refiere a`	Metadatos → Objeto de datos / Fragmento de objeto	Conecta los metadatos con los elementos de datos que describen.
Entre capas	`se almacena en`	Fragmento de objeto → Contenedor de datos	Asigna fragmentos a los contenedores que los alojan.
	`lectura/escritura`	Cliente DynoStore → Fragmento de objeto	Captura las interacciones de E/S del cliente, anotadas con el rendimiento.
	`búsqueda`	Cliente DynoStore → Metadatos	Indica operaciones de acceso a metadatos.

(flechas moradas en la Figura 1).

IV. Diseño de KAGIO para manejar cargas de trabajo dinámicas

La ontología definida en la Sección III-A proporciona a KAGIO la base conceptual para representar las entidades y relaciones de DynoStore. Partiendo de este modelo, en esta sección se describe cómo se utiliza el conocimiento semántico para apoyar la adaptación en tiempo de ejecución de un sistema elástico. La Figura 2 ilustra el flujo de trabajo empleado para realizar este proceso, que integra la supervisión continua, la actualización de los Grafos de Conocimiento, el razonamiento y la reconfiguración del sistema.

El flujo propuesto se compone de tres fases principales: (i) Carga de datos, (ii) Razonamiento, y (iii) Reconfiguración del sistema.

A. Carga de datos

En la primera fase, la información del sistema se carga en un KG, que puede implementarse mediante bases de datos NoSQL orientadas a grafos (por ejemplo, Neo4j o ArangoDB). Para este proceso, se emplea un archivo YAML que inicializa el grafo y crea las entidades y relaciones a nivel de infraestructura.

En KAGIO, las entidades y relaciones se crean, actualizan o eliminan de forma dinámica a partir de las acciones y eventos generados en DynoStore. Para ello, se monitorizan continuamente los registros (*logs*) del sistema y se sincronizan los eventos con el KG, garantizando que la representación semántica refleje fielmente el estado real de la infraestructura.

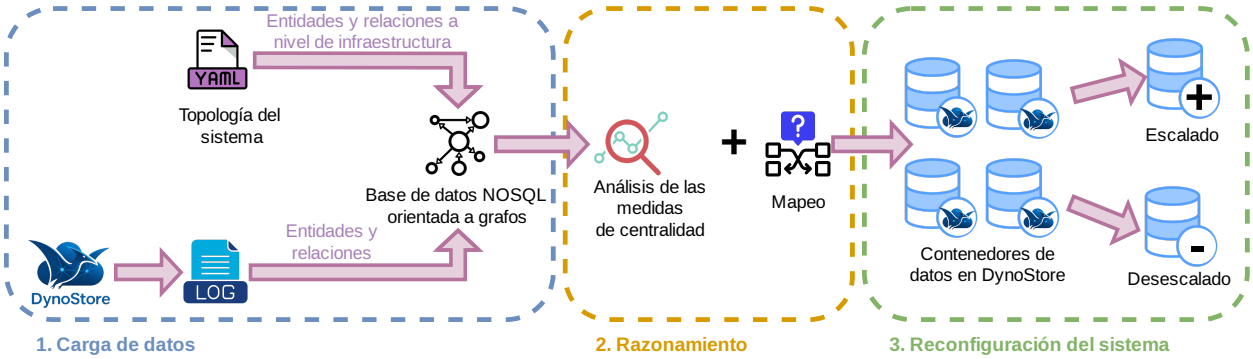


Fig. 2: Flujo de trabajo para crear sistemas elásticos basados en KG.

B. Razonamiento

Una vez actualizado el KG, se ejecutan los módulos de análisis para identificar posibles eventos de elasticidad, tales como la replicación automática de objetos y la redistribución de la carga de trabajo.

La integración del análisis de grafos en la ontología permite razonar sobre el comportamiento del sistema más allá de las relaciones directas. En particular, las medidas de centralidad constituyen una herramienta analítica fundamental para evaluar las fluctuaciones de carga y los patrones de acceso.

Este trabajo se enfoca en dos medidas de centralidad complementarias: el *grado de entrada (in-degree centrality)* y el *PageRank*. Aunque ambas cuantifican la importancia relativa de los nodos dentro del grafo, captan diferentes dimensiones del comportamiento del sistema.

El **grado de entrada** se aplica a los nodos de *metadatos* para identificar los objetos a los que se accede o hace referencia con mayor frecuencia. Se define formalmente de la siguiente manera:

Definition 3 (Grado de entrada) Dado un grafo de conocimiento $KG = (N, A, R, \lambda_N)$, la centralidad de grado de entrada $C_{in}(n)$ de una entidad $n \in N$ se define como:

$$C_{in}(n) = \frac{|\{(u, r, n) \in A \mid r \in R\}|}{|V| - 1}$$

donde el numerador cuenta todas las tripletas entrantes cuyo objeto es n, es decir, todas las entidades u que están conectadas a n a través de alguna relación r. El denominador normaliza la puntuación según el número máximo de aristas entrantes del grafo.

Según la ontología propuesta, cada arista entrante en los nodos de metadatos representa una operación de búsqueda de un cliente. Por lo tanto, los objetos más referenciados pueden identificarse como *datos calientes* (los más accedidos del sistema). En KAGIO, las entidades identificadas en esta categoría se replican o migran a contenedores de datos con menor nivel de uso.

La segunda métrica es el **PageRank** [23], utilizada para clasificar los *contenedores de datos* e identificar los más populares. Esta métrica se define de la siguiente forma:

Definition 4 (PageRank) Dado un grafo de conocimiento $KG = (N, A, R, \lambda_N)$, la puntuación PageRank $PR(v)$ de un nodo $v \in V$ se define recursivamente como:

$$PR(n) = \frac{1 - d}{|N|} + d \sum_{(u,r,n) \in A} \frac{PR(u)}{|\{(u, r', w) \in A\}|}$$

donde:

- $d \in (0, 1)$ es el factor de amortiguación (por defecto $0,85$);
- $(u, r, n) \in E$ representa las relaciones $r \in R$ que conectan el nodo u con el nodo v;
- el denominador $|\{(u, r', w) \in A\}|$ es el número de aristas salientes del nodo u, independientemente del tipo de relación.

A diferencia de las métricas basadas únicamente en el grado de entrada, el PageRank considera la importancia de los nodos vecinos, lo que permite detectar contenedores que alojan objetos altamente populares o centrales. Un contenedor conectado a un gran volumen de objetos de alto rango heredará un valor de PageRank mayor, indicando su relevancia estructural en la distribución de la información.

En conjunto, ambas métricas proporcionan una visión jerárquica de la dinámica del sistema: mientras que la centralidad de grado de entrada identifica *qué* datos generan mayor carga, el PageRank expone *dónde* se concentra dicha carga dentro de la topología de almacenamiento.

C. Reconfiguración del sistema

Las conclusiones obtenidas en la fase de razonamiento se traducen en acciones concretas de adaptación. Cuando el grafo de conocimiento revela una alta concentración de centralidad, el flujo de trabajo activa una de las siguientes operaciones de elasticidad: *(i) Ampliación*, que replica objetos con alta centralidad en diferentes contenedores, distribuyendo los accesos entre una mayor cantidad de recursos virtuales; y *(ii) Reducción*, que identifica los contenedores infrautilizados con un PageRank bajo, libera sus recursos y, de ser necesario, migra los fragmentos de datos almacenados para preservar la redundancia.

El proceso de ampliación consta de tres pasos. Primero, se identifica y etiqueta para replicación el 25 % de los objetos con mayor grado de entrada. Esta replicación consiste en copiar los n fragmentos de dichos objetos a nuevos contenedores, que son seleccionados por un balanceador que combina el valor

de PageRank con su factor de utilización actual. Este mecanismo prioriza contenedores con baja utilización y bajo PageRank, lo que mejora la distribución de recursos y evita la saturación. Para ello, a cada contenedor de la topología se le asigna un valor calculado como:

$$s_j = w_u U_j + w_p P_j + \mathcal{E}_|, \quad (1)$$

donde U_j es el factor de utilización normalizado del contenedor j, P_j es su PageRank normalizado, w_u y w_p son coeficientes que ponderan la importancia de ambas métricas, y $\mathcal{E}_|$ es una pequeña perturbación aleatoria ($Uniform(0, 10^{-6})$) introducida para desempatar contenedores con puntuaciones idénticas.

Una vez obtenido este valor, el balanceador selecciona los n contenedores con las puntuaciones más bajas. Para evitar que varios fragmentos de un mismo objeto se alojen en el mismo lugar, cualquier contenedor que ya posea un fragmento de dicho objeto es excluido de la selección.

Por otro lado, la reducción de escala consiste en disminuir el número de fragmentos asignados a los objetos con bajo grado de entrada. Estos objetos presentan menores dependencias y se acceden con poca frecuencia, por lo que su reconstrucción tiene un impacto mínimo en el rendimiento global. Reducir el número de fragmentos disminuye la cantidad de metadatos por gestionar y la sobrecarga de almacenamiento. Para no comprometer la resiliencia del sistema, la reducción del número de fragmentos n se limita a una unidad por iteración, garantizando que siempre se cumpla la condición $n > k$.

Tras cada reconfiguración, el nuevo estado operativo se registra en el grafo de conocimiento, manteniendo la coherencia absoluta entre la representación semántica y el sistema físico.

V. Resultados experimentales

Esta sección describe el prototipo de KAGIO utilizado para su evaluación y presenta los resultados experimentales obtenidos.

A. Prototipo de experimentación

Para evaluar KAGIO, se implementó un prototipo compuesto por un servidor REST, un agente de monitorización de los registros (*logs*) de DynoStore, y una base de datos orientada a grafos. Todos los experimentos se realizaron utilizando ArangoDB[2], una base de datos NoSQL multimodelo que provee un motor nativo para el manejo de Grafos de Conocimiento (KG). Específicamente, ArangoDB utiliza AQL (*ArangoDB Query Language*) para la manipulación de datos.

El servidor REST se desarrolló utilizando FastAPI[3] y provee operaciones CRUD para todas las entidades y relaciones definidas en la ontología. Además, este servidor expone el modelo de grafo a los clientes, lo que permite agregar, modificar y eliminar dinámicamente relaciones y entidades. Esto preserva tanto la representación semántica del KG como los estados de operación del sistema de almacenamiento distribuido.

El agente de monitorización lee continuamente los registros de DynoStore y los transforma en eventos de gestión, tales como cargas, accesos a datos y reconfiguraciones de la infraestructura. Cada evento se convierte en una llamada a la API del servidor REST, que actualiza el KG. Este mecanismo permite la sincronización en tiempo real entre las actividades físicas del sistema y su representación semántica, asegurando que el grafo refleje correctamente el estado operativo actual.

Para los experimentos y como prueba de concepto, se utilizó un simulador basado en el diseño de DynoStore. Esto permitió realizar evaluaciones bajo cargas de trabajo controladas y agilizar el despliegue de soluciones. El simulador se basa en eventos discretos que reproducen el comportamiento del sistema mediante el análisis de trazas obtenidas en estudios previos [5]. Dicha herramienta modela los componentes principales de la arquitectura: la API Gateway, el Servicio de Metadatos y los Contenedores de Datos, lo que facilita la ejecución de eventos asíncronos de carga, descarga y replicación. La parametrización se realiza mediante un archivo YAML que especifica la latencia entre componentes, el ancho de banda y la capacidad de almacenamiento de cada nodo.

Asimismo, el simulador imita el esquema de redundancia implementado en DynoStore; por lo tanto, cada objeto se divide en fragmentos y un balanceador de carga los asigna a distintos contenedores distribuidos. En estas pruebas, se configuró el entorno para generar 1000 objetos con un tamaño aleatorio distribuido uniformemente entre 128 y 1024 MB. Cada objeto se escribe una vez y se lee T veces. El valor T se obtiene a partir de una variable aleatoria discreta uniforme $D \sim U\{1, 10\}$. Dependiendo de D, se genera el número de operaciones de lectura T en uno de tres rangos: si $D \leq 2$, entonces $T \sim U\{70, 100\}$ (alta frecuencia de acceso); si $3 \leq D \leq 6$, entonces $T \sim U\{10, 70\}$ (frecuencia media); y si $D > 6$, entonces $T \sim U\{0, 10\}$ (baja frecuencia).

Para llevar a cabo esta evaluación, se consideró una topología de 20 nodos simulados. Las características técnicas de dichos contenedores se detallan en la Tabla III.

B. Análisis de los resultados

El análisis experimental se divide en dos fases principales: la primera enfocada en la identificación de objetos populares, y la segunda orientada a detectar posibles desequilibrios en las operaciones atendidas por cada contenedor. El estudio se concentra exclusivamente en las métricas de lectura y acceso, ya que estas suelen constituir el principal cuello de botella en los sistemas distribuidos (al ser solicitadas por una mayor cantidad de clientes que las operaciones de escritura).

El objetivo de esta evaluación es verificar si el sistema es capaz de adaptarse de forma dinámica median-

[2]https://arango.ai/products/arangodb/
[3]https://fastapi.tiangolo.com/

Tabla III: Características de los Contenedores de Datos.

ID	Storage Cap.	Used Mem.	Storage Util.	CPU Util.
dc-0	16GB	0.5GB	40 %	5 %
dc-1	16GB	0.5GB	35 %	10 %
dc-2	16GB	0.5GB	20 %	2 %
dc-3	16GB	0.5GB	30 %	3 %
dc-4	64GB	2GB	30 %	6 %
dc-5	64GB	2GB	40 %	6 %
dc-6	64GB	1GB	15 %	5 %
dc-7	64GB	1GB	30 %	5 %
dc-8	450GB	20GB	44 %	12 %
dc-9	450GB	20GB	45 %	13 %
dc-10	450GB	5GB	11 %	2 %
dc-11	450GB	5GB	10 %	3 %
dc-12	300GB	15GB	33 %	15 %
dc-13	300GB	15GB	33 %	15 %
dc-14	1000GB	25GB	25 %	9 %
dc-15	1000GB	25GB	25 %	9 %
dc-16	1000GB	90GB	75 %	35 %
dc-17	1000GB	90GB	75 %	35 %
dc-18	1000GB	10GB	5 %	5 %
dc-19	1000GB	10GB	5 %	5 %

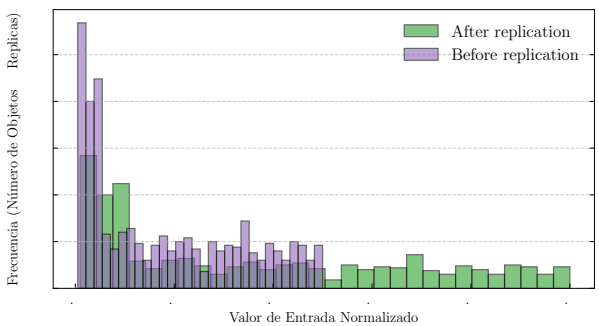


Fig. 3: Grado de entrada de los objetos calientes en el sistema.

te una estrategia de replicación. Dicha técnica proporciona elasticidad al ajustar el número de réplicas en función de las fluctuaciones de la carga de trabajo, los patrones de acceso y los requisitos de rendimiento.

En primer lugar, se calculó el grado de entrada de cada nodo en DynoStore. Posteriormente, se aisló el 25 % de los objetos con el valor más alto. Estos elementos se denominan *calientes*, dado que son los más demandados de la red y, por tanto, los más propensos a generar cuellos de botella por su alto volumen de peticiones. La Figura 3 presenta una comparación del grado de entrada normalizado antes (en verde) y después (en morado) de aplicar la replicación dinámica. En el eje X se representa dicho valor normalizado, mientras que el eje Y muestra la frecuencia de los objetos. Como se puede observar, el rango original se extiende hasta [0, 0.10], mientras que en la versión optimizada con KG se comprime a [0, 0.05]. Físicamente, un valor alto implica que un objeto es accedido en numerosas ocasiones, sometiendo a sus nodos anfitriones a una carga intensiva. Al ejecutar la replicación dinámica, el tráfico se divide entre el objeto original y su copia. En consecuencia, se reduce el número de elementos con un grado extremo, lo que permite balancear las lecturas entre una mayor cantidad de servidores. Todo esto se logra preservando la consistencia, ya que el propio DynoStore se encarga de mantener sincronizada la información [5].

En la segunda etapa, la evaluación se centró en medir el PageRank de cada contenedor antes y después

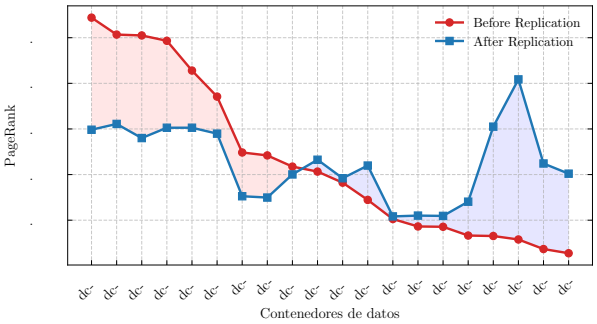


Fig. 4: PageRank de los contenedores de datos antes y después de la replicación de objetos.

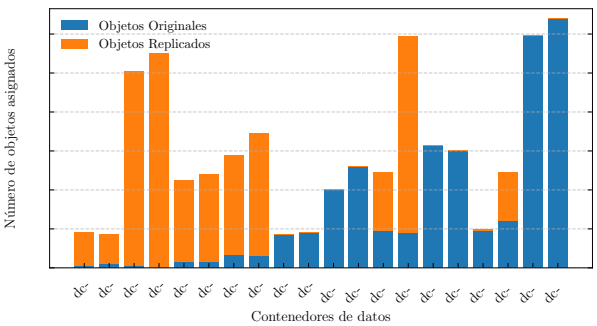


Fig. 5: Migración de objetos *calientes* a nuevos contenedores de datos.

de la adaptación. En la Figura 4, el eje X representa los contenedores evaluados (ordenados descendentemente según su métrica original), y el eje Y indica su puntuación. La línea roja representa el estado inicial previo a la replicación, y la línea azul ilustra la topología resultante. En este contexto, un valor elevado de PageRank indica que un nodo aloja objetos muy demandados; por consiguiente, si la carga recae predominantemente en unos pocos servidores con alta puntuación, el sistema tiende a una centralización extrema. Esta vulnerabilidad es evidente en la configuración inicial (línea roja), donde 6 contenedores superan el umbral de 0.0015, lo que refleja una fuerte concentración de lecturas. Por el contrario, en la versión con replicación dinámica (línea azul), se observa un claro aplanamiento de la curva: el PageRank de los contenedores que originalmente se encontraban por debajo de 0.0010 experimenta un incremento, al tiempo que se reduce drásticamente el de los nodos sobrecargados. Físicamente, esto se traduce en una distribución mucho más equitativa, descentralizando la red y mitigando la dependencia de un grupo reducido de equipos.

Finalmente, en una tercera etapa se analizó la distribución de objetos asignados a cada contenedor de datos. En la Figura 5, el eje X representa los nodos de la infraestructura y el eje Y indica la cantidad total de objetos alojados en cada uno. Las barras en azul cuantifican los elementos originales, y las porciones en naranja ilustran las réplicas generadas tras aplicar la propuesta. Como se observa, el algoritmo ubica de manera efectiva las copias en aquellos contenedores con menor carga inicial. Esta estrategia evita la saturación de los nodos principales, descentraliza el volumen de lecturas y equilibra el uso de los recursos

a través de toda la infraestructura.

VI. Conclusiones

Los sistemas elásticos son capaces de ajustarse dinámicamente a su entorno. En este artículo se exploró el diseño de una solución elástica para el almacenamiento distribuido utilizando un enfoque basado en Grafos de Conocimiento (KG) y análisis de centralidad. El sistema se modeló semánticamente mediante una ontología que captura sus principales características y relaciones operativas. Para evaluar esta propuesta, se utilizó como base DynoStore, que proporciona la flexibilidad necesaria para crear diferentes topologías de almacenamiento distribuido. No obstante, esta ontología puede extenderse para describir los componentes de otros sistemas similares presentes en la literatura, tales como Ceph o DataSpaces. A partir de esta representación, se desarrolló un flujo de trabajo guiado por el conocimiento, capaz de identificar objetos altamente demandados (*calientes*) y contenedores de datos infrautilizados (*fríos*). Posteriormente, los datos con alta demanda se replican en los contenedores con menor carga, creando un entorno que se adapta a las características fluctuantes de la carga de trabajo. Para llevar a cabo este proceso, se diseñó un balanceador que combina el factor de utilización de los contenedores con métricas topológicas, como el PageRank, a fin de determinar la ubicación óptima para almacenar las nuevas réplicas.

La evaluación experimental, llevada a cabo mediante un simulador alimentado con métricas reales de experimentos previos en DynoStore, demostró que el enfoque propuesto incrementa efectivamente el valor de PageRank de los contenedores infrautilizados y disminuye el de aquellos con un volumen de accesos excesivo. Esto reduce significativamente la cantidad de puntos críticos (*hotspots*) en la infraestructura, mitigando la aparición de posibles cuellos de botella. Como trabajo futuro, se planea validar la solución en despliegues físicos a gran escala e incorporar nuevas métricas en el proceso de toma de decisiones, tales como el consumo energético y la localidad de los datos.

Agradecimientos

El presente trabajo ha sido financiado por la Universidad Carlos III de Madrid a través del proyecto "Nuevas técnicas para la gestión de fábricas de datos en el cómputo continuo" (2025/00822/001 - Proyectos JÓVENES PPIT2025) y por la Agencia Española de Investigación a través del proyecto "Nuevas técnicas escalables de E/S para cargas de trabajo híbridas de HPC e intensivas en datos (SCIOT)" con referencia PID2022-138050NB-I00.

Referencias

[1] Valeria Cardellini, Patrizio Dazzi, Gabriele Mencagli, Matteo Nardelli, and Massimo Torquati, "Scalable compute continuum," 2025.

[2] Diana Carrizales-Espinoza, Dante D Sanchez-Gallegos, JL Gonzalez-Compean, and Jesus Carretero, "Structmesh: A storage framework for serverless computing continuum," *Future Generation Computer Systems*, vol. 159, pp. 353–369, 2024.

[3] Marco Danelutto, Patrizio Dazzi, and Massimo Torquati, "Structuring the continuum," in *Advanced Information Networking and Applications*, Leonard Barolli, Ed., Cham, 2024, pp. 212–223, Springer Nature Switzerland.

[4] Marc-André Vef, Alberto Miranda, Ramon Nou, and André Brinkmann, "From static to malleable: Improving flexibility and compatibility in burst buffer file systems," in *High Performance Computing*, Amanda Bienz, Michèle Weiland, Marc Baboulin, and Carola Kruse, Eds., Cham, 2023, pp. 3–15, Springer Nature Switzerland.

[5] Dante D. Sanchez-Gallegos, J. L. Gonzalez-Compean, Maxime Gonthier, Valerie Hayot-Sasson, J. Gregory Pauloski, Haochen Pan, Kyle Chard, Jesus Carretero, and Ian Foster, "Dynostore: A wide-area distribution system for the management of data over heterogeneous storage," in *2025 IEEE 25th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, Tromsø, Norway, 2025, pp. 83–92, IEEE.

[6] Nathanaël Cheriere, Matthieu Dorier, and Gabriel Antoniu, "Pufferbench: Evaluating and optimizing malleability of distributed storage," in *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS)*, Dallas, TX, USA, 2018, pp. 35–44, IEEE.

[7] Nathanaël Cheriere and Gabriel Antoniu, "How fast can one scale down a distributed file system?," in *2017 IEEE International Conference on Big Data (Big Data)*, Boston, MA, USA, 2017, pp. 141–150, IEEE.

[8] Nathanaël Cheriere, Matthieu Dorier, Gabriel Antoniu, Stefan M. Wild, Sven Leyffer, and Robert Ross, "Pufferscale: Rescaling hpc data services for high energy physics applications," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, Melbourne, VIC, Australia, 2020, pp. 182–191, IEEE.

[9] Jesus Carretero, Javier Garcia-Blas, Marco Aldinucci, Jean Baptiste Besnard, Jean-Thomas Acquaviva, André Brinkmann, Marc-André Vef, Emmanuel Jeannot, Alberto Miranda, Ramon Nou, Morris Riedel, Massimo Torquati, and Felix Wolf, "Adaptive multi-tier intelligent data manager for exascale," in *Proceedings of the 20th ACM International Conference on Computing Frontiers*, New York, NY, USA, 2023, CF '23, p. 285–290, Association for Computing Machinery.

[10] Jesus Carretero, Javier Garcia-Blas, André Brinkmann, Marc Vef, Jean-Baptiste Besnard, Massimo Torquati, Yi Ju, and Raffaele Montella, "Adaptive hpc input/output systems," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 14352 LNCS, pp. 199 – 202, 2024.

[11] Carlos Guerrero, Isaac Lera, and Carlos Juiz, "Optimization policy for file replica placement in fog domains," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 21, pp. e5343, 2020.

[12] Katerina Mitropoulou, Panagiotis Kokkinos, Polyzois Soumplis, and Emmanouel Varvarigos, "Anomaly detection in cloud computing using knowledge graph embedding and machine learning mechanisms," *J. Grid Comput.*, vol. 22, no. 1, Dec. 2023.

[13] Kashif Mehmood, Katina Kralevska, and David Palma, "Knowledge graph embedding in intent-based networking," in *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*, Saint Louis, MO, USA, 2024, pp. 13–18, IEEE.

[14] Kashif Mehmood, Katina Kralevska, and David Palma, "Knowledge-driven intent life-cycle management for cellular networks," *IEEE Transactions on Network and Service Management*, vol. 22, no. 5, pp. 4806–4826, 2025.

[15] Zoe Vasileiou, Indika Kumara, Georgios Meditskos, Kamil Tokmakov, Dragan Radolović, Jesús Gorroñogoitia Cruz, Elisabetta Di Nitto, Damian Andrew Tamburri, Willem-Jan Van Den Heuvel, and Stefanos Vrochidis, "A knowledge-based approach for guided development of infrastructure as code," *Software and Systems Modeling*, pp. 1–34, 2025.

[16] Sangkeun Lee, Sudharshan S. Vazhkudai, and Raghul Gunasekaran, "Applying graph analytics to understand compute core usage and publication trends in a petascale supercomputing facility," in *2017 IEEE 24th In-*

ternational Conference on High Performance Computing (HiPC), Jaipur, India, 2017, pp. 294–305, IEEE.

[17] Pouya Kousha, Vivekananda Sathu, Matthew Lieber, Hari Subramoni, and Dhabaleswar K. Panda, "Democratizing hpc access and use with knowledge graphs," in *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, New York, NY, USA, 2023, SC-W '23, p. 243–251, Association for Computing Machinery.

[18] Shuaihao Zhong, Duoqiang Wang, Wei Li, Feng Lu, and Hai Jin, "Burner: Recipe automatic generation for hpc container based on domain knowledge graph," *Wireless Communications and Mobile Computing*, vol. 2022, no. 1, pp. 4592428, 2022.

[19] Yue Zhao, Chunhua Liao, and Xipeng Shen, "An infrastructure for hpc knowledge sharing and reuse," *ACM SIGPLAN Notices*, vol. 52, no. 8, pp. 461 – 462, 2017.

[20] Smita Agrawal and Atul Patel, "A study on graph storage database of nosql," *International Journal on Soft Computing, Artificial Intelligence and Applications*, vol. 5, pp. 33–39, 02 2016.

[21] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann, "Knowledge graphs," *ACM Comput. Surv.*, vol. 54, no. 4, July 2021.

[22] Maxime Gonthier, Dante D. Sanchez-Gallegos, Haochen Pan, Bogdan Nicolae, Sicheng Zhou, Hai Duc Nguyen, Valerie Hayot-Sasson, Greg Pauloski, Jesus Carretero, Kyle Chard, and Ian Foster, "D-rex: Heterogeneity-aware reliability framework and adaptive algorithms for distributed storage," in *Proceedings of the 39th ACM International Conference on Supercomputing*, New York, NY, USA, 2025, ICS '25, p. 853–867, Association for Computing Machinery.

[23] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd, "The pagerank citation ranking : Bringing order to the web," 1999.

[24] T.V. Vignesh, K. Vijaya, and R.I. Minu, "Graph-theoretic approaches in semantic web service orchestration and discovery: A comprehensive literature survey," 2024, p. 1026 – 1029.

[25] Tarek Zaarour, Ahmed Khalid, Preeja Pradeep, and Ahmed Zahran, "Using distributed ledgers to build knowledge graphs for decentralized computing ecosystems," 2024, p. 3083 – 3092.

[26] Ioana Ramona Martin, Tudor Cioara, Gabriel Ioan Arcas, Ionut Anghel, and Massimo Bertoncini, "Knowledge graph model for computing continuum over smart grid," in *2024 15th International Conference on Information, Intelligence, Systems & Applications (IISA)*, Chania Crete, Greece, 2024, pp. 1–8, IEEE.

[27] Nada Elgendy and Ahmed Elragal, "Big data analytics: A literature review paper," *Lecture Notes in Computer Science*, vol. 8557, pp. 214–227, 08 2014.

[28] Yaxin Chang, Jiafei Guo, Hanbo Wang, Dapeng Man, and Jiguang Lv, "An information-centric network caching method based on popularity rating and topology weighting," *Wireless Communications and Mobile Computing*, vol. 2022, no. 1, pp. 4979057, 2022.

METRIC-AI: Entorno de monitorización semi-autónomo para entornos maleables

Alberto Cascajo[1], Javier Fernandez[2], David E. Singh [3], Jesus Carretero [4]

Resumen—
De forma continua, la escala y complejidad de los entornos de Computación de Alto Rendimiento (HPC) y Cloud están aumentando, suponiendo un reto para la gestión de los mismos. Actualmente, uno de los retos principales pasa por maximizar la eficiencia energética y el rendimiento ante cargas de trabajo con comportamientos variables, como workflows y aplicaciones dinámicas. Debido a este nuevo tipo de ejecuciones con recursos variables a lo largo de su ejecución, el modelo actual de reserva y uso de recursos queda obsoleto debido a su ineficiencia. Este trabajo propone un cambio de paradigma gracias la gestión automática de un clúster con soporte para gestión dinámica de recursos mediante agentes de Inteligencia Artificial. `METRIC-AI` facilita la gestión de un clúster HPC, combinado con un algoritmo para asignar recursos de forma dinámica y eficiente a las aplicaciones en ejecución, las cuales son identificadas por el sistema para evaluar su mejor configuración.

Palabras clave— **Maleabilidad, Gestión dinámica de recursos, monitorización, Computación de altas prestaciones.**

I. Introducción

En la actualidad, además de las aplicaciones clásicas de los entornos *High-Performance Computing* (HPC), existen nuevas aplicaciones y workflows que no se ajustan al paradigma de reserva de recursos clásico: reservas estáticas y fijas durante el tiempo total de ejecución de las aplicaciones y workflows. Esta forma de gestión de recursos *estática*, no se realiza de forma eficiente, puesto que no se adapta a las necesidades de todos los tipos de aplicaciones que se ejecutan en los sistemas HPC.

Los distintos tipos de trabajos que se pueden ejecutar en los sistemas HPC se pueden clasificar en: *trabajos estáticos*, *trabajos moldeables*, y *trabajos maleables*. En el caso de los trabajos estáticos, o que requieren siempre los mismos recursos, las ejecuciones se realizan siempre con el mismo número de recursos, tanto en tiempo de despliegue como en tiempo de ejecución. Son aplicaciones que requieren, por defecto, cierta cantidad de recursos y no se pueden modificar. En segundo lugar se sitúan las aplicaciones moldeables, que son aquellas que permiten establecer un número de recursos diferente en tiempo de despliegue. Es decir, se pueden solicitar distintos recursos pero antes del inicio de la ejecución, no siendo posible modificarlos durante la ejecución. Ejemplos de este tipo de trabajos son aquellos basados en MPI, donde se pueden solicitar distintos recursos en función de los procesos que se quieren desplegar. Finalmente, están los trabajos maleables, que son aquellos que permiten solicitar distintos recursos en tiempo de despliegue (similar a los moldeables), pero, además, permiten modificar el uso de recursos en tiempo de ejecución. Esto significa que una aplicación puede solicitar más recursos ante una fase de computación intensiva, y liberarlos si ya no van a ser utilizados, lo que no es factible realizar por defecto en la mayoría de los sistemas actuales.

Los entornos maleables, o sistemas con gestión dinámica de recursos, constituyen una línea de desarrollo en la evolución de las infraestructuras HPC. Como se ha descrito anteriormente, su interés radica en la capacidad de ajustar de forma automática la asignación de recursos de cómputo, memoria, almacenamiento o red en función de la demanda del sistema, lo que permite mejorar la eficiencia, la escalabilidad y, potencialmente, los costes asociados a las ejecuciones de los usuarios. Es una idea que se puede comparar con los modelos de provisión bajo demanda de las plataformas basadas en Cloud Computing.

Para un correcto desempeño de estos sistemas, la incorporación de monitores de recursos resulta especialmente relevante, ya que permite observar en tiempo real el estado, tanto del propio sistema, como de los trabajos en ejecución, lo que permite guiar al mecanismo de decisión para la reasignación dinámica de recursos. La monitorización no solo cumple una función de supervisión, sino que actúa como base para la adaptación automática y la toma de decisiones en estos sistemas que se vuelven cada vez más complejos.

Desde una perspectiva práctica, y considerando la evolución de las aplicaciones hacia trabajos más complejos y workflows con diferentes fases y requerimientos, los sistemas con gestión dinámica de recursos, así como los frameworks y bibliotecas que proporcionan mecanismos de maleabilidad, tienen como objetivo la realización de una gestión de recursos más eficiente, así como evitar consumos innecesarios, y permitir a los usuarios un mecanismo de adaptación de sus trabajos en función de sus requerimientos, sin limitaciones iniciales.

La estructura del artículo es la siguiente: la sección II muestra trabajos relevantes relacionados con nuestra propuesta. La sección III describe la organización de la arquitectura del entorno, así como presenta los benchmarks maleables. La sección IV proporciona una visión práctica de las métricas de rendimiento generadas por las ejecuciones de los micro-

[1]Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: `acascajo@inf.uc3m.es`
[2]Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: `jfmunoz@inf.uc3m.es`
[3]Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: `dexposit@inf.uc3m.es`
[4]Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: `jcarrete@inf.uc3m.es`

benchmarks donde se ve que pueden ampliar o reducir el número de procesos. Finalmente, la Sección V resume las principales conclusiones y el trabajo futuro.

II. Estado de la cuestión

La maleabilidad en HPC se refiere a la capacidad de una aplicación para cambiar su tamaño o número de procesos durante su ejecución, expandiéndose o contrayéndose según un criterio determinado. La gestión dinámica de recursos hace referencia a la capacidad de un sistema de proveer distinto número de recursos a las aplicaciones en tiempo de ejecución. Por lo tanto, ambos conceptos están estrechamente relacionados, ya que las aplicaciones pueden ser maleables, pero si el sistema en el que se ejecutan no dispone de capacidad de gestión dinámica de recursos, no sirve de mucho. Actualmente, hay diversos trabajos como [1], [2] que buscan hacer que los sistemas clásicos de ejecución y despliegue de aplicaciones sean más dinámicos, puesto que normalmente son sistemas diseñados para ejecuciones estáticas, haciendo especial mención Slurm, uno de los gestores de recursos más utilizados en sistemas HPC.

La visión de la mayoría de trabajos para introducir mecanismos dinámicos para la gestión de recursos tiene dos vías: (1) modificando el código de Slurm y generando su propia versión adaptada, (2) en base a plugings, y en menor medida, (3) utilizando mecanismos intermediarios entre el software desarrollado y Slurm. La ventaja principal del trabajo que se presenta en este artículo es que se introduce la figura del Controlador Inteligente, quien hace de intermediario, junto con la toma de decisiones implementada en el entorno METRIC-AI, y permite que se puede utilizar una gestión dinámica de recursos propia, sin necesidad de modificar Slurm o realizar una nueva configuración, por lo que se cataloga en el tercer apartado de las alternativas.

Desde el punto de vista de la maleabilidad en las aplicaciones, es importante destacar que hay varios métodos que se pueden seguir para lograr la idea de "maleabilidad", siendo unas más complejas que otras, o requiriendo más esfuerzo en implementación.

Dado que muchas aplicaciones disponen de mecanismos de checkpoint para salvaguardar el estado de sus ejecuciones en momentos puntuales, dicho punto podría ser utilizado para detener la aplicación y arrancarla de nuevo con un nuevo número de recursos, lo que se puede denominar *stop&restart*. Es el caso del trabajo presentado por Aliaga et al. [3] que proporcionan conclusiones sobre cómo hacer checkpointing y relanzar las aplicaciones con un nuevo número de procesos (tomando como objetivo aquellas aplicaciones que son de tipo *moldeables* y que pueden ser fácilmente adaptables a este contexto). Es un mecanismo sencillo de aplicación sencilla y con tolerancia a fallos implícita. Otra alternativa y de forma similar a la anterior, es PCM (Process Checkpointing and Migration) [4], que es un entorno capaz de implementar características maleables gracias a los momentos en los que se realizan los checkpoints. Finalmente, una alternativa clásica se encuentra en [5], donde los autores describen una herramienta que les permite ejecutar aplicaciones maleables basadas en MPI. Se basa en la implementación de una capa de software *Stop Restart Software* (SRS) junto con *Runtime Support System* (RSS) que gestiona los mecanismos de reconfiguración de las aplicaciones durante su ejecución. Su funcionamiento consiste en la modificación del código fuente de las aplicaciones para insertar funciones específicas (de SRS) donde se realizarán los checkpoints y se gestionará la reconfiguración de la aplicación.

De forma diferente a las alternativas anteriores se presenta *User-Level Failure Mitigation* (ULFM) [1], donde los desarrolladores extienden el estándar MPI con nuevas características para tolerancia a fallos. No es un trabajo relacionado directamente con la maleabilidad en las aplicaciones, sin embargo, las características de poder lanzar nuevos procesos cuando otros fallan, manteniendo el estado de la ejecución y redistribuyendo los datos, nos acerca mucho al significado que tiene la maleabilidad en las aplicaciones. A raíz de trabajar con MPI para maleabilidad, se pueden agregar nuevas y más recientes alternativas. Ejemplos de propuestas que siguen este esquema son FlexMPI [6][7], Elastic MPI [8], ReSHAPE [9] and Dynamic Management of Resources (DMR) [10]. FlexMPI es una solución de maleabilidad muy completa que incluye varios componentes: un monitor, un gestor de recursos, un módulo de análisis del rendimiento, y funciones que permiten hacer determinados modelos de redistribución de datos de forma transparente al usuario. Esta es uno de sus principales puntos fuertes. Sin embargo, es una solución que no está integrada con gestores de recursos como Slurm, lo que dificulta su puesta en funcionamiento en sistemas con recursos compartidos. Elastic-MPI es similar a la propuesta anterior, incluyendo una gran cantidad de componentes, pero en su caso, extienden el estándar MPI con nuevas características para dar soporte a aplicaciones maleables. Su fortaleza principal es la coordinación con el gestor de recursos Slurm. Finalmente, DMR es un entorno de maleabilidad que, como algunas propuestas anteriores, se basa en dos componentes principales: un gestor de recursos (Slurm) y una librería que implementa su solución basándose en MPI.

III. METRIC-AI

En esta sección se va a describir el ecosistema formado por METRIC-AI: Monitorización y Evaluación en Tiempo Real de Infraestructuras Cloud y HPC mediante IA. Para ello, se van a describir los componentes principales que incluye, así como la parte influyente en la toma de decisiones para entornos dinámicos.

[1] https://fault-tolerance.org/

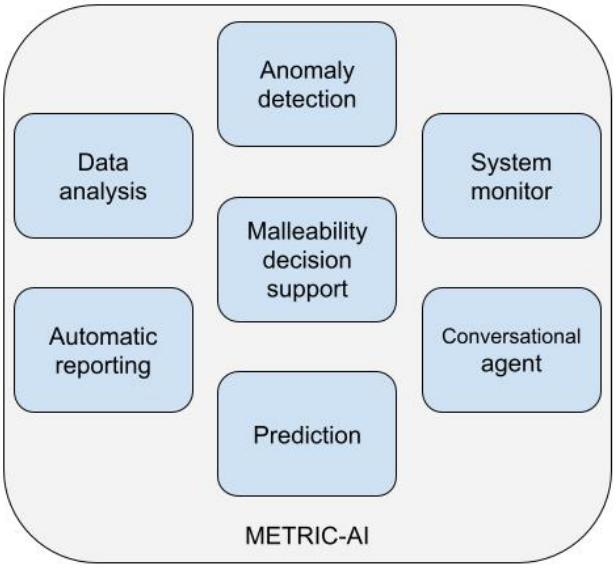


Fig. 1: Componentes principales del framework METRIC-AI.

A. Arquitectura

METRIC-AI es un framework diseñado para ofrecer diversas características de interés, tanto en el ámbito de la monitorización de sistemas, como de proveedor de decisiones en entornos maleables. Para ello, el framework dispone de varios componentes:

1. *Monitor de sistemas:* Para obtener información acerca del estado de los distintos nodos de cómputo que hay en un sistema HPC o Cloud, se hace uso de `LIMITLESS`[11], un monitor libre, escalable y con baja sobrecarga, que se encarga de proporcionar métricas resumidas y generales de cada nodo que lo ejecuta. La selección de este componente se debe a que apenas tiene coste de computación porque es muy ligero, y es fácil de integrar, ya sea a través de sus conexiones con Redis, Influx-DB o a través de ZeroMQ. Para este caso de uso, se ha integrado mediante Redis, ya que otros componentes pueden tener acceso a la misma base de datos de forma sencilla.
2. *Agente de análisis:* La información recopilada por el monitor, y aquella que se obtiene de otros canales (por ejemplo Slurm o las aplicaciones), debe ser procesada y analizada. Para ello, se dispone de un agente que se encarga de obtener datos periódicamente de Redis, procesarlos, y extraer información relevante para el sistema, de forma que otros componentes puedan acceder a ella de forma sencilla, en lugar de que todas las partes deban procesar la misma información.
3. *Agente de detección de anomalías:* Los sistemas HPC son críticos y sensibles en ciertas circunstancias. El componente detector de anomalías se encarga de analizar las series temporales de datos en tiempo real, en busca de datos no aceptados o coherentes. Por ejemplo, temperaturas elevadas pueden provocar fallos graves en el hardware. El agente de detección de anomalías es capaz de detectar variaciones en los datos recolectados si estos no cumplen una serie de requisitos, los cuales pueden estar directamente programados en el agente, o pueden inferirse de su modelo entrenado. En este caso, el agente tiene umbrales para ciertos valores, y un modelo entrenado con valores aceptados, lo que significa que, si los datos están dentro de los umbrales, pero éstos no han sido integrados en el modelo, puede detectarlo como anomalía y reportar al administrador. El ejemplo de las temperaturas es igualmente significativo: si 2 de 10 nodos de cómputo alcanzan una temperatura elevada pero dentro de los umbrales fijados, pero la CPU en uso es baja, el modelo reporta una anomalía, ya que, previsiblemente, si un nodo no está a pleno rendimiento, no debería alcanzar valores altos de temperatura. En este tipo de circunstancias, el agente envía notificaciones al administrador vía Slack.
4. *Agente de reporting:* Desde el punto de vista de los administradores de sistemas, disponer de informes del estado de las plataformas es crucial. En la mayoría de casos se dispone de sistemas de visualización de datos (por ejemplo, paneles de Grafana con gráficos de agujas y líneas para representar el estado general del sistema), pero disponer de resúmenes periódicos autogenerados facilita la tarea de gestión, ya que alivia la carga de tener que revisar, de forma continua, cómo funciona el sistema. Además, esto se complementa con el agente de detección de anomalías, permitiendo observar el estado del sistema antes de la anomalía y una vez que se ha detectado, sirviendo de análisis o búsqueda de causas probables.
5. *Predictor de estados:* Basándose en los datos recopilados durante la ejecución del sistema, este agente se encarga de predecir el estado del sistema en busca de posibles problemas potenciales, fases de poca carga (para apagar nodos de forma automática) o de alta demanda (para arrancar nodos de forma acelerada). Esto facilita una gestión proactiva del sistema.
6. *Agente conversacional:* Dado que el sistema se construye con agentes de IA, se aprovecha su potencia y se integra un agente conversacional conectado a los demás componentes, por lo que es capaz de acceder a toda la información del sistema. Este agente permite que los usuarios puedan hacer consultas en lenguaje natural y obtener respuestas de la misma forma, facilitando así la interacción con los usuarios.
7. *Notificador Maleabilidad:* Para una integración eficiente con mecanismos de maleabilidad, se ha desarrollado un componente que se dedica a informar de las mejores opciones de maleabilidad dentro del sistema. Esto se traduce en que un controlador inteligente, que recibe las peticiones de recursos de las aplicaciones, se retroalimenta de METRIC-AI para obtener cuál es el mejor orden para asignar los recursos, permitiendo que aplicaciones con mejor escalabilidad tengan

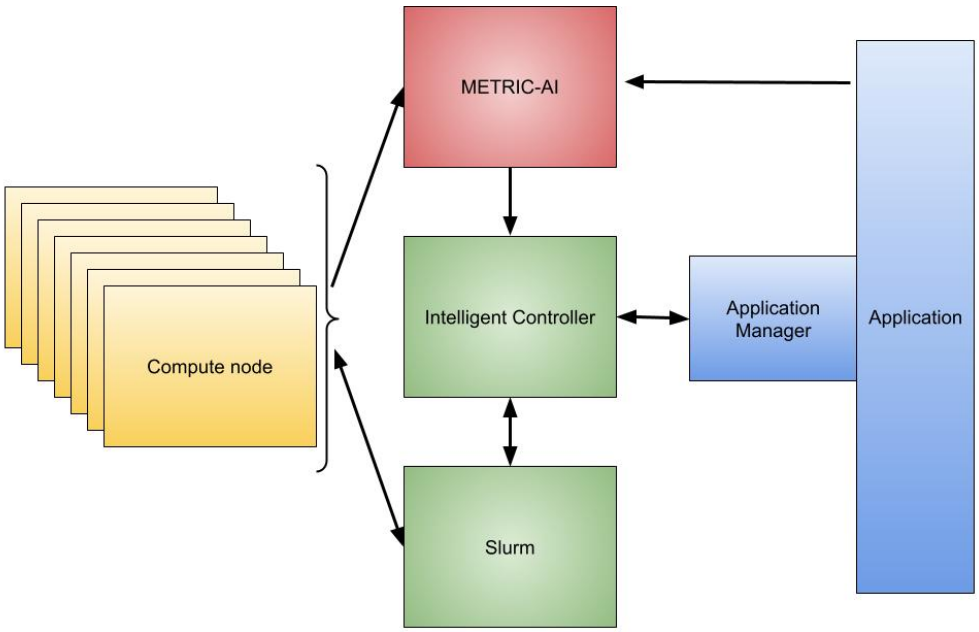


Fig. 2: Arquitectura general de METRIC-AI y componentes principales del sistema dinámico.

más opciones de obtener recursos, derivando en un uso más eficiente de los mismos.

B. Controlador inteligente (IC)

En la Figura 2 se puede ver una vista general de la arquitectura de un sistema con gestión dinámica de recursos. La plataforma consiste en una serie de nodos de cómputo que son administrados por el gestor de recursos Slurm. Dada la complejidad en la gestión de los sistemas dinámicos (en constante cambio o evolución), es necesario tener una visión holística del sistema y de los eventos que surgen.

Con este propósito, el framework integra un ecosistema de monitorización diseñado para suministrar métricas de estado, indicadores de rendimiento, sistemas de alerta y modelos predictivos: METRIC-AI. La arquitectura se fundamenta en el soporte del monitor de sistemas `LIMITLESS`, y esporádicamente *FlexMPI*[12], permitiendo una observabilidad multinivel que abarca desde la infraestructura del nodo hasta la granularidad de la aplicación. El núcleo de la toma de decisiones reside en el Controlador Inteligente (IC), componente centralizado encargado de ejecutar las decisiones estratégicas. Complementariamente, se implementa el Application Manager (AM), el cual actúa como interfaz de enlace con las aplicaciones.

En términos de diseño, el IC se implementa con un modelo cliente-servidor. Bajo este paradigma, el IC actúa como el nodo servidor centralizado, mientras que las aplicaciones asumen el rol de clientes, interactuando a través de las interfaces de comunicación proporcionadas por el módulo AM, y donde el framework METRIC-AI coopera con el IC intercambiando información.

Dado que las tareas de recolección, agregación y ordenación de métricas provenientes del sistema son manejados por el componente METRIC-AI, los requisitos de almacenamiento del IC se centran en la gestión eficiente de identificadores de aplicaciones, listados de nodos y el almacenamiento transitorio de los resultados derivados del proceso de toma de decisiones. Para cubrir estas necesidades, se ha integrado la base de datos en memoria Redis, seleccionada por su capacidad para ofrecer estructuras de datos de baja latencia y alta escalabilidad. Asimismo, su facilidad de integración mediante bibliotecas estándar en C permite realizar consultas sencillas con un rendimiento óptimo para los requisitos del sistema.

C. Integración con Slurm

La interacción entre el IC y el gestor de recursos Slurm se realiza mediante el uso de variables de entorno y la interfaz de programación *libslurm*. Toda la implementación de la integración está embebida en el código del IC, sin embargo, se ofrece como una librería que se ejecuta en el lado del cliente (aplicaciones), concretamente en el módulo AM, basándose en tres pilares fundamentales:

1. **Acceso a información:** Las variables de entorno utilizadas por Slurm, las cuales contienen información sobre la reserva y el entorno de ejecución, son locales al contexto del trabajo del usuario. Por tanto, no son accesibles de forma directa desde un proceso externo (como podría ser el IC).
2. **Seguridad y Privilegios:** Por diseño, Slurm restringe las operaciones sobre los trabajos (jobs) al usuario propietario de los mismos. La ejecución de solicitudes de modificación o asignación de recursos desde el AM garantiza que las peticiones posean los permisos adecuados, evitando conflictos por permisos en las operaciones.
3. **Escalabilidad y Distribución:** Delegar la tarea de solicitar recursos a las aplicaciones mitiga los cuellos de botella en el IC, ya que las comunicaciones quedan distribuidas y ejecutadas por

un hilo desplegado de forma transparente en cada aplicación.

Esta estrategia de diseño asegura la compatibilidad con instalaciones estándar de Slurm en entornos de producción, eliminando la necesidad de parches o modificaciones en el código fuente del gestor de recursos, lo que facilita su despliegue en infraestructuras reales o de desarrollo.

IV. Resultados experimentales

En esta sección se presentan una serie de resultados de ejecuciones de aplicaciones en un sistema con gestión dinámica de recursos. Los resultados principales son ejecuciones en las que se puede apreciar cómo la maleabilidad puede permitir que las plataformas HPC sean utilizadas con mayor eficiencia, reduciendo tiempos de ejecución, tiempos de espera, y potencialmente, costes por reservas.

La plataforma donde se han ejecutado estas pruebas es el Supercomputador C3 de la Universidad Carlos III de Madrid, que consiste en 90 nodos de cómputo, cada uno con dos CPU AMD EPYC 7713 de 64 núcleos cada uno (128 núcleos en total), 1024 GB de memoria RAM y SSD NVMe PCIe Gen4 de 3,5 TB, además de 5 nodos de doble procesador con dos CPU AMD EPYC 7513, cada una con 32 núcleos, 512 GB de memoria RAM, SSD NVMe PCIe de 7 TB para espacio de almacenamiento temporal local, y 42 GPU NVIDIA A40. Todo ello, con sistema de archivos Lustre con 546 TB de capacidad y alta redundancia, NFS con acceso RDMA y 48 TB de capacidad, y Red Infiniband HDR de 100 Gbps para computación y almacenamiento de datos. Para este estudio, se han realizado las pruebas en una partición de 10 nodos.

A. Casos de uso

Para la demostración de funcionamiento del sistema se han utilizado 3 aplicaciones con distintas configuraciones, en función de la escalabilidad alcanzada por cada una. Las tres aplicaciones que se han utilizado son (1) Gradiente Conjugado, (2) Jacobi, y (3) WaComM. El método del Gradiente Conjugado es un algoritmo utilizado comúnmente para resolver sistemas de ecuaciones lineales de la forma $Ax = b$. El método de Jacobi es otro algoritmo iterativo clásico para resolver sistemas de ecuaciones lineales de tipo $Ax = b$. A diferencia del anterior, el Jacobi es un método más sencillo, cuya principal característica es que el cálculo de cada componente de la solución es completamente independiente de los demás dentro de una misma iteración. Por último, WaComM es una evolución del modelo LAMP3D (Lagrangian Assessment for Marine Pollution 3D). WaComM se utiliza operativamente para el transporte y la difusión de contaminantes en el Centro de Monitorización y Modelado de Aplicaciones Marinas y Atmosféricas gestionado por el Departamento de Ciencia y Tecnología (DiST[2]) de la Universidad de Nápoles "Parthenope".

[2]`https://dist.uniparthenope.it`

Para configurar distintos casos de uso, se han elaborado varios ejemplos de las anteriores aplicaciones con distintas configuraciones. El resultado son 4 casos de uso por cada aplicación, que se corresponden con aplicaciones que escalan hasta un máximo de nodos, a saber: 1, 2, 4 y 8 nodos. Los resultados iniciales de los casos de uso se pueden encontrar en la Tabla I, y una representación gráfica de su escalabilidad en la Figura 3.

Caso de uso	**Tiempo total(s)**
Gradiente8	653
Gradiente4	660
Gradiente2	513
Gradiente1	780
Jacobi8	206
Jacobi4	160
Jacobi2	386
Jacobi1	309
Wacomm8	615
Wacomm4	746
Wacomm2	1084
Wacomm1	544

Tabla I: Tabla resumen de casos de uso

Para demostrar el funcionamiento del sistema con gestión dinámica de recursos, se han ejecutado las aplicaciones en tres configuraciones diferentes: (1) ejecuciones estáticas, donde todas las aplicaciones son forzadas a comenzar con su número máximo de nodos; (2) ejecuciones de $[1 - n]$, donde cada aplicación es desplegada indicando que se desea comenzar con un valor entre 1 y n, siendo n el número máximo de nodos, y donde Slurm es quien decide finalmente el valor, y donde luego cada aplicación solicita recursos hasta su máximo; y (3) estableciendo que todas las aplicaciones comiencen con 1 nodo y, posteriormente, soliciten recursos hasta su máximo deseado.

El primer modo de ejecución, *estática*, que se puede ver en la Figura 4, se obtiene el tiempo de ejecución total (*walltime*) equivalente al que se conseguiría en un sistema HPC clásico, donde Slurm lanza las aplicaciones cuando éstas tienen recursos suficientes. En este caso, el planificador de trabajos de Slurm (*backfilling*[13]) juega un papel importante porque es capaz de adelantar trabajos si estos pueden ocupar los espacios que dejan las aplicaciones al terminar, antes de que la siguiente que se ha planificado obtenga todos los recursos necesarios.

Tras la segunda ejecución, $[1 - n]$, que se puede ver en la Figura 5, no es necesario que las aplicaciones obtengan el máximo de recursos para empezar, sino que, con una pequeña porción de ellos pueden arrancar y, gracias a la maleabilidad, ir adquiriendo el resto de recursos en tiempo de ejecución. Con esta política se consigue que las aplicaciones empiecen antes de lo que lo harían de forma estática, aunque también pueden tener ciertos desventajas: (1) es posible que no se puedan obtener más recursos en tiempo de ejecución (depende directamente del planificador de Slurm), con lo cual se realizaría una ejecución con recursos mínimos, pero adelantada; (2) es posible que

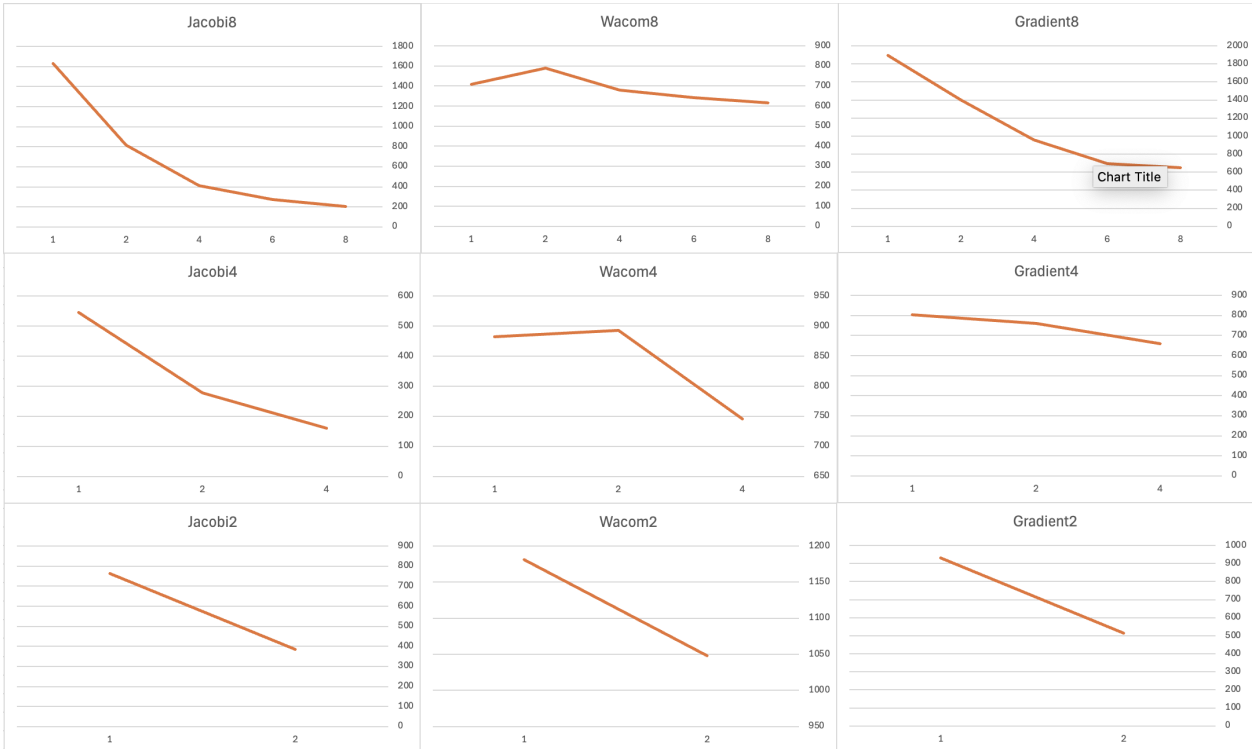


Fig. 3: Aceleración de los distintos casos de uso. El eje X representa el número de nodos utilizado, y el eje Y representa el tiempo de ejecución de la misma aplicación y configuración. No se muestran datos de ejecución de casos donde 1 nodo es la mejor configuración porque representa un único punto, por lo que se referencia dicho dato en la Tabla I.

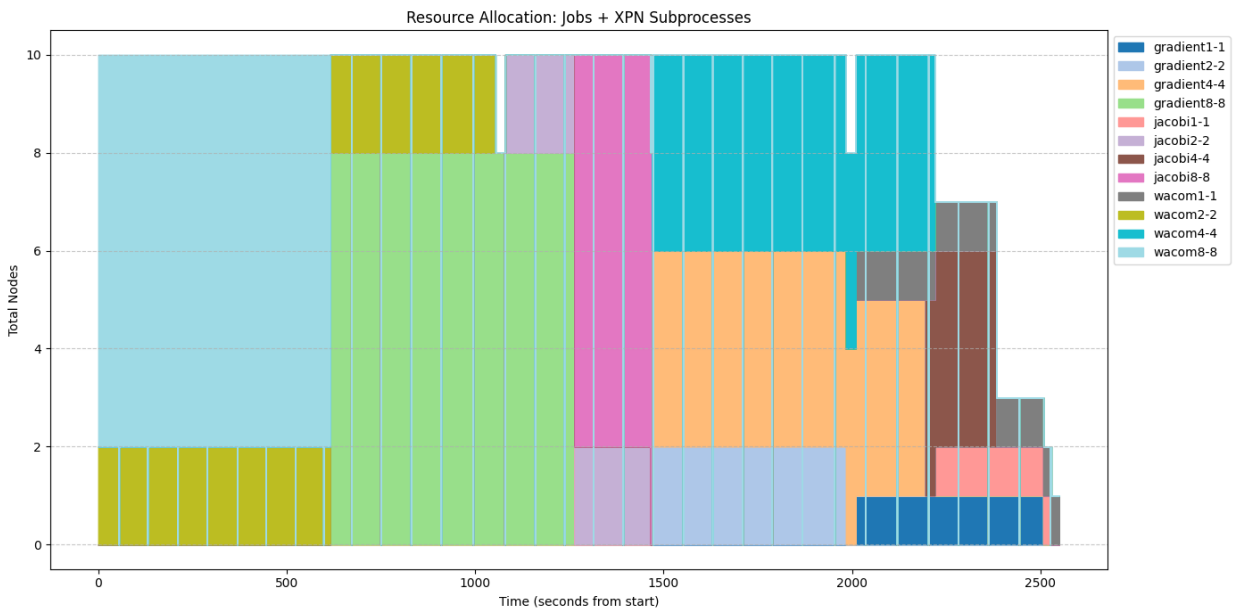


Fig. 4: Casos de uso ejecutados en estático, con configuración ideal.

no se obtengan todos los recursos solicitados (lo que también depende directamente de Slurm), en cuyo caso la ejecución no será la más eficiente, pero se ejecutará antes de tiempo, aunque con menos recursos. Además, en este caso el *backfilling* de Slurm también puede actuar, aunque tiene menos oportunidades de adelantar trabajos debido a que las aplicaciones son capaces de aceptar cualquier espacio disponible para iniciarse. Con todo ello, se puede apreciar un tiempo total de ejecución muy similar al caso estático, pero con un mayor número de recursos libres al final. Esto se puede ver desde dos puntos de vista: nuevas aplicaciones pueden comenzar antes de tiempo, y se produce un ahorro energético debido a un menor uso del sistema.

El último modo de ejecución consiste en permitir que todas las aplicaciones comiencen con un solo nodo, lo que facilitará que todas las aplicaciones se puedan ejecutar en cuanto haya un mínimo recurso libre. Posteriormente, la aplicación solicitará tantos recursos como estime oportunos hasta alcanzar su máximo deseado. Al igual que en la política anterior, esta ejecución permite un inicio adelantado, pero de más aplicaciones, beneficiando a aquellas que son más cortas o tienen una aceleración poco pronunciada. A medida que las aplicaciones más cortas van

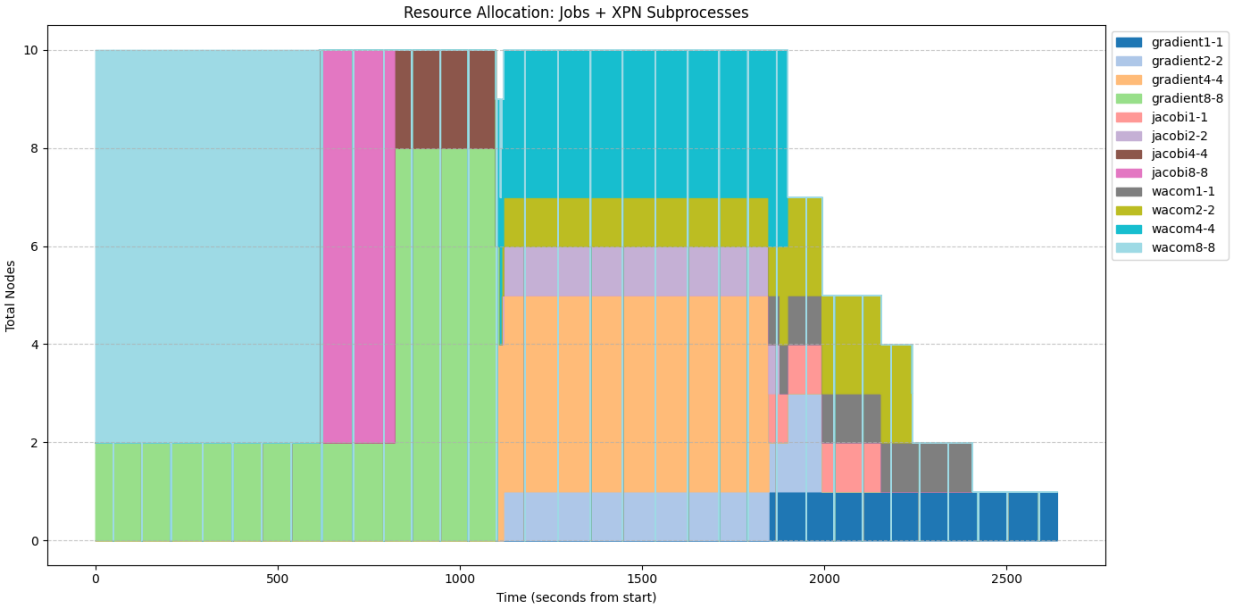


Fig. 5: Casos de uso ejecutados con rango de aceptación $[1-n]$.

Escenario	**Walltime (s)**	**Diferencia (%)**
Estático	2551	0 %
$[1-n]$	2640	+3,4 %
Empezar en 1	1558	−38,9 %

Tabla II: Comparación del *walltime* de los 3 modos de ejecución.

terminando y se liberan recursos, las demás pueden ir adquiriendo éstos para ampliar sus procesos (al igual que en el resto de casos, dependiendo directamente del planificador de Slurm). De la misma forma, es posible que esta política ejecute aplicaciones que no alcancen su número de recursos deseado, pero es un problema delegado al planificador de Slurm, dado que es una línea de trabajo en estudio.

Como resultado principal de este estudio, se puede apreciar que habilitar maleabilidad en las aplicaciones y configurar sistemas con gestión dinámica de recursos son clave para: (1) maximizar el uso de los recursos en sistemas HPC gracias a aprovechar "los huecos"que hay entre aplicaciones de distinto tamaño, así como (2) reducir el makespan o tiempo total de ejecución de un conjunto de tareas agracias a poder realizar ejecuciones tempranas (aunque con menos recursos) que adelantan trabajo hasta que se pueden obtener más recursos. Todo ello apoyado, además, por un potencial ahorro en las reservas de recursos computacionales.

A modo de resumen, la Tabla II muestra los tiempos totales de las ejecuciones. Se puede ver que la política de Empezar en 1 consigue una reducción del tiempo total de ejecución de un 38 %.

V. Conclusiones

La integración de *METRIC-AI* ha facilitado la transición hacia un modelo de gestión de recursos maleable más eficiente, superando las ineficiencias inherentes a las reservas estáticas en entornos HPC. Al integrar un sistema de monitorización activa con un algoritmo de reparto basado en la aceleración de las aplicaciones, se logra una optimización del rendimiento global que prioriza la eficiencia operativa y reduce el tiempo de ejecución total del sistema. Asimismo, la arquitectura descentralizada propuesta no solo garantiza la escalabilidad y la seguridad al operar dentro del contexto de privilegios del usuario, sino que facilita su uso en entornos de producción reales al interactuar de forma nativa con Slurm.

Como trabajo futuro se está estudiando refinar el modelo utilizado para la toma de decisiones sobre maleabilidad, considerando más variables que puedan ofrecer una mejora más relevante, así como reducir la complejidad del modelo con el fin de reducir su coste computacional. Otra vía de mejora consiste en la búsqueda de algoritmos automáticos que permitan obtener las aceleraciones de las aplicaciones (o aproximaciones) sin necesidad de basarse en datos previamente recopilados, generando un sistema .ºn line"de análisis y optimización.

Agradecimientos

Esta publicación es parte del proyecto de I+D+I PID2022-138050NB-I00, financiado por el MICIU/AEI/10.13039/501100011033/"FEDER Una manera de hacer Europa", y también ha sido parcialmente financiado por el proyecto "Monitorizacion y Evaluacion en Tiempo Real de Infraestructuras Cloud y HPC mediante IA" (METRIC-AI) de la Universidad Carlos III de Madrid en el marco de las Ayudas para la Actividad Investigadora de los Jóvenes Doctores, del Programa Propio de Investigación con código 2024/00766/001.

Referencias

[1] Mohak Chadha, Jophin John, and Michael Gerndt, "Extending slurm for dynamic resource-aware adaptive batch scheduling," in *2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2020, pp. 223–232.

[2] Rajat Bhattarai, Howard Pritchard, and Sheikh Ghafoor,

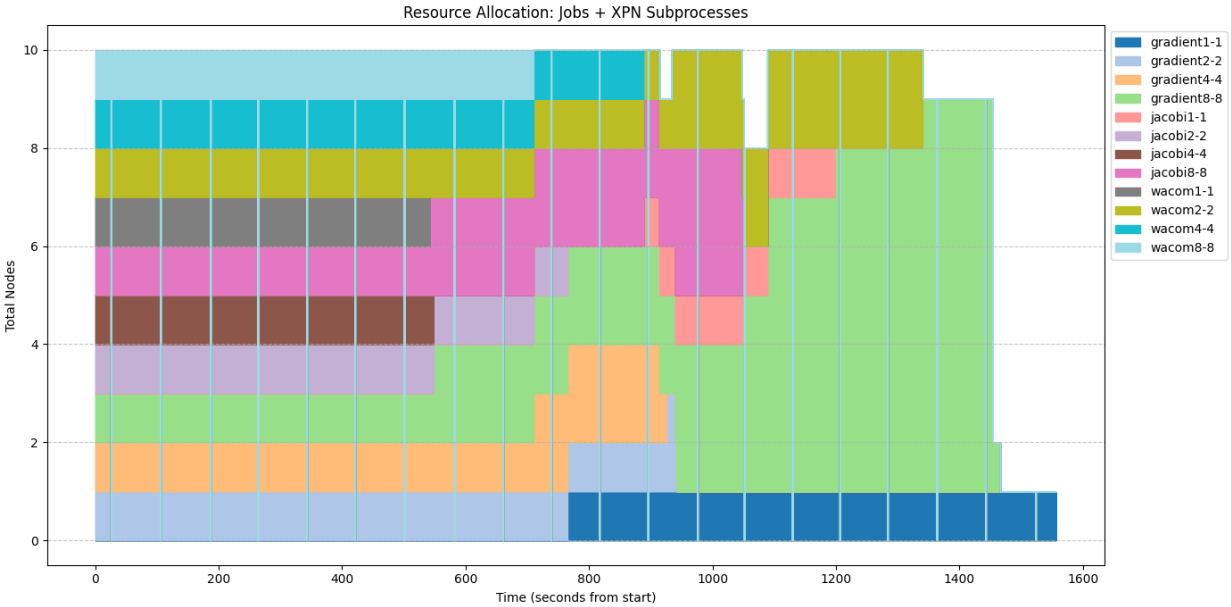


Fig. 6: Casos de uso ejecutados empezando con 1 nodo y pidiendo más recursos.

"Evaluation of a dynamic resource management strategy for elastic scientific workflows," in *Euro-Par 2024: Parallel Processing Workshops*. 2025, pp. 334–345, Springer Nature Switzerland.

[3] José I Aliaga, Maribel Castillo, Sergio Iserte, Iker Martín-Álvarez, and Rafael Mayo, "A survey on malleability solutions for high-performance distributed computing," *Applied Sciences 2022, Vol. 12, Page 5231*, vol. 12, pp. 5231, 5 2022.

[4] Kaoutar El Maghraoui, Boleslaw K. Szymanski, and Carlos Varela, "An architecture for reconfigurable iterative mpi applications in dynamic environments," in *Parallel Processing and Applied Mathematics*, Roman Wyrzykowski, Jack Dongarra, Norbert Meyer, and Jerzy Waśniewski, Eds., Berlin, Heidelberg, 2006, pp. 258–271, Springer Berlin Heidelberg.

[5] Sathish S. Vadhiyar and Jack J. Dongarra, "Srs: A framework for developing malleable and migratable parallel applications for distributed systems," *Parallel Processing Letters*, vol. 13, pp. 291–312, 6 2003.

[6] Gonzalo Martín, Maria-Cristina Marinescu, David E. Singh, and Jesús Carretero, "Flex-mpi: An mpi extension for supporting dynamic load balancing on heterogeneous non-dedicated systems," in *Euro-Par 2013 Parallel Processing*, Felix Wolf, Bernd Mohr, and Dieter an Mey, Eds., Berlin, Heidelberg, 2013, pp. 138–149, Springer Berlin Heidelberg.

[7] David E. Singh and Jesus Carretero, "Combining malleability and i/o control mechanisms to enhance the execution of multiple applications," *Journal of Systems and Software*, vol. 148, pp. 21–36, 2019.

[8] Isaías Comprés, Ao Mo-Hellenbrand, Michael Gerndt, and Hans-Joachim Bungartz, "Infrastructure and api extensions for elastic execution of mpi applications," in *Proceedings of the 23rd European MPI Users' Group Meeting*, New York, NY, USA, 2016, EuroMPI 2016, p. 82–97, Association for Computing Machinery.

[9] Rajesh Sudarsan and Calvin J. Ribbens, "Reshape: A framework for dynamic resizing and scheduling of homogeneous applications in a parallel environment," in *2007 International Conference on Parallel Processing (ICPP 2007)*, 2007, pp. 44–44.

[10] Sergio Iserte, Rafael Mayo, Enrique S. Quintana-Ortí, Vicenç Beltran, and Antonio J. Peña, "Dmr api: Improving cluster productivity by turning applications into malleable," *Parallel Computing*, vol. 78, pp. 54–66, 2018.

[11] Alberto Cascajo, David E. Singh, and Jesús Carretero, "Limitless - light-weight monitoring tool for large scale systems," in *2021 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2021, pp. 220–227.

[12] Gonzalo Martín, Maria-Cristina Marinescu, David E. Singh, and Jesús Carretero, "Flex-mpi: An mpi extension for supporting dynamic load balancing on heterogeneous non-dedicated systems," in *Euro-Par 2013 Parallel Processing*, 2013, pp. 138–149.

[13] Sergei Leonenkov and Sergey Zhumatiy, "Introducing new backfill-based scheduler for slurm resource manager," *Procedia Computer Science*, vol. 66, pp. 661–669, 2015, 4th International Young Scientist Conference on Computational Science.

Simulación del manejo seguro y confiable de datos en entornos IoT-Edge-HPC

Diana Carrizales-Espinoza[1], Dante D. Sánchez-Gallegos[1] y Jesús Carretero[1]

Resumen— **El crecimiento masivo de datos generados por los dispositivos de IoT y la Inteligencia Artificial ha impulsado una transición hacia el paradigma del cómputo continuo, abarcando desde el borde (edge) hasta la nube (cloud) a exaescala. Además, en entornos de Cómputo de Alto Rendimiento (HPC), la velocidad pura de cálculo no es la única métrica crítica; sino que la heterogeneidad de las infraestructuras, así como la convergencia de tecnologías resultan esenciales durante el procesamiento, transporte e intercambio de información. Para abordar este tipo de escenarios donde cantidades masivas de datos deben ser procesadas, en este artículo presentamos un simulador modular centrado en patrones de paralelismo y flujos de trabajo para sistemas de IoT-Edge-HPC, en donde priorizamos la lógica de ejecución sobre la simulación física de hardware. Este simulador permite a los desarrolladores y las organizaciones estimar tiempos de ejecución, costes operativos y necesidades de almacenamiento, así como el impacto de aplicar distintas medidas de seguridad durante la transmisión de datos. La evaluación reveló la coherencia y validez del modelo frente a diversos conjuntos de datos.**

Palabras clave— **Requisitos no funcionales, seguridad de datos, confiabilidad de datos, cómputo continuo, transporte de datos.**

I. Introducción

En la actualidad, el crecimiento de datos, así como la cantidad de dispositivos de IoT que los producen, han experimentado un incremento masivo [1], [2]. Las tecnologías emergentes de inteligencia artificial generativa como ChatGPT y Gemini han abierto un nuevo mercado en el que los datos son generados por miles de millones de usuarios [3], [4], [5]. Gracias a esto, estamos presenciando y viviendo un cambio trascendental en el paradigma de la computación, donde los datos ya no son procesados de forma exclusiva en grandes centros de datos centralizados o en supercomputadores aislados, sino que el cómputo se ha convertido en un continuo donde el flujo de los datos va desde los dispositivos de IoT que los producen hasta los clústeres que se encuentran en el cloud [6], [7], [2].

Debido a este movimiento de datos a través de diferentes infraestructuras, el manejo de requisitos no funcionales (RNFs), como lo son la privacidad y la confiabilidad, se vuelve crucial durante el transporte y almacenamiento de los datos. Esto es especialmente importante cuando se manejan datos sensibles, tales como datos médicos. En la literatura, se han propuesto soluciones de punto a punto para añadir a los contenidos distintas propiedades como lo son la seguridad [8], [9], la confiabilidad [10], [11] y la integridad [12], las cuales ayudan a mitigar y, en algunos casos, a solucionar por completo los riesgos que plantean los incidentes actuales en la nube como lo son las interrupciones del servicio [13], acceso no autorizado, violaciones de la integridad [14], [15] y violaciones de la confidencialidad [14], [15]. Sin embargo, aplicar estas soluciones en escenarios reales representa un desafío para las organizaciones a la hora de hacer frente a los elevados costes de procesamiento e intercambio de datos producidos en diferentes entornos (ya sea en el borde, en la niebla o en la nube). En tal escenario, el número de operaciones enviadas a servicios externos, así como el volumen de datos, resultan en tareas costosas que consumen mucho tiempo, lo que agrava los problemas mencionados anteriormente originados por la acumulación de datos.

Sin embargo, el desplegar y probar estos sistemas en la práctica es complejo y costoso, por lo cual es necesario contar con herramientas que nos permitan evaluar distintos parámetros y configuraciones de los sistemas sin la necesidad de utilizar un ambiente real. Para hacer frente a este desafío, en este artículo se presenta una herramienta que permite la simulación de la gestión de datos en entornos de IoT-edge-HPC, que tiene en cuenta el aprovisionamiento y manejo de RNFs. El objetivo principal del simulador es permitir a los desarrolladores y a las organizaciones estimar las necesidades de almacenamiento, el rendimiento de los servicios y el impacto de la aplicación de medidas de seguridad y confiabilidad en la transmisión de datos. En este sentido, este simulador puede utilizarse para estimar el tiempo de ejecución y los costes de las operaciones de preservación para diferentes esquemas de preparación/recuperación de datos y patrones de paralelismo en sistemas de HPC. Para lograr esto, este simulador sigue un enfoque modular que facilita la inclusión o sustitución de las etapas de procesamiento. Además, permite ajustar de forma dinámica distintos parámetros clave de los esquemas, como el número y los tipos de etapas, así como características específicas de los patrones de paralelismo que permiten brindar eficiencia durante el flujo de los datos en los esquemas.

El resto del artículo se encuentra organizado de la siguiente manera: la Sección II presenta el estado de la literatura en relación con las herramientas de simulación existentes; en la Sección III se describe la base utilizada para diseñar el simulador propuesto; la Sección IV detalla sus características principales; en la Sección V se presenta la evaluación experimental realizada; finalmente, la Sección VI presenta las conclusiones obtenidas durante el diseño y evaluación del simulador.

[1]Dpto. de Informática, Universidad Carlos III de Madrid, e-mails: dcarriza@pa.uc3m.es. y jcarrete@inf.uc3m.es.

II. Antecedentes y trabajo relacionado

En la literatura, podemos encontrar una amplia gama de herramientas que permiten la simulación de los entornos de edge, fog o cloud. En este artículo hemos clasificado estas herramientas en seis categorías en función de la tarea o de los elementos simulados. Estas categorías corresponden a la simulación de redes, sensores, procesamiento de datos, aprovisionamiento de recursos, recopilación de datos y asignación de tareas.

En la simulación de sensores, se pueden encontrar herramientas como OPNET y ATEMU. OPNET es una herramienta ampliamente utilizada para modelar el IoT [16], abordando aspectos como la confiabilidad, la cobertura, el alto uso del canal y las redes Long Term Evolution (LTE). Por otro lado, ATEMU [17] permite la simulación de sensores de grano fino a nivel de instrucción, lo que incluye la integración de red.

Entre las herramientas que permiten la simulación de redes, podemos encontrar a Vue [18], un simulador que permite crear un entorno de prueba orientado a la simulación de redes como LTE-Advanced, 5G-NR, HSPA+, LTE, MIMO, DC-HSPA y señales de radio multiestándar.

En la categoría de simuladores para la recopilación de datos desde el edge hacia el fog, podemos encontrar herramientas como SimIot, DPWSim y iFogSim. SimIot es un simulador centrado en los datos recopilados por los sensores y en su procesamiento en el edge y el fog en escenarios de atención médica [19]. Por otro lado, DPWSim es un conjunto de herramientas de simulación enfocado en la evaluación del rendimiento del proceso de recopilación de datos desde los dispositivos edge. Mientras que iFogSim es una herramienta que permite generar modelos de IoT y entornos fog, los cuales permiten medir el impacto que la gestión de recursos tiene en los costos operativos, la latencia, la congestión de la red y el consumo de energía [20].

Por otro lado, los simuladores que consideran el procesamiento de datos en el cloud pueden clasificarse según el modelo de programación utilizado, centrándose principalmente en el procesamiento por lotes (batch processing) y en el modelo de procesamiento MapReduce (MR) [21]. En este contexto, podemos encontrar herramientas como Mumak y MRSim. Así, Mumak es una herramienta de simulación que implementa diferentes algoritmos de planificación para tareas MR, ampliamente utilizada para estimar los tiempos de finalización del procesamiento de conjuntos de datos en el cloud [22]. Mientras que MRSim es un simulador de clústeres de Hadoop basado en eventos discretos. Esta herramienta permite la simulación de la configuración de un clúster y del rendimiento de las tareas, los cuales se miden en función de la utilización del hardware y del tiempo de finalización del trabajo [23].

Para la asignación de tareas en infraestructuras de edge, fog o cloud, Mohan y Kangasharjy [24] desarrollaron un simulador de edge-fog-cloud que permite distribuir tareas entre los nodos para reducir los costos de procesamiento y de red. Mientras que Minh-Quang Tran et al. [25] ampliaron el conjunto de herramientas de simulación SIMIC mediante la integración de una capa de IoT, la cual lleva a cabo la asignación de máquinas virtuales (VMs). Adicionalmente, Yaozhong Song et al. [26] se enfocaron en la distribución y gestión de tareas a través del edge con el fin de cumplir con los requisitos de calidad de servicio (QoS).

En el último grupo, entre los simuladores diseñados para la provisión de recursos en el cloud podemos encontrar herramientas como CloudSim, ICanCloud, IoTsim, MultiRECloudSim. CloudSim [27] es un conjunto de herramientas utilizado para modelar y simular entornos de computación en la nube, así como para evaluar algoritmos de aprovisionamiento de recursos. CloudSim ha servido de base para muchos de los simuladores existentes en la literatura. ICanCloud [28] es una plataforma de simulación para modelar y simular entornos en la nube, que proporciona un conjunto completo de módulos para representar con gran detalle entornos de computación en la nube y su arquitectura subyacente. IoTim [29] permite la simulación del procesamiento por lotes en sistemas de IoT mediante tareas MR que reciben parámetros de entrada, como los costos de red, los tiempos de ejecución, los tiempos de retardo y los costos computacionales de las máquinas virtuales. Otro ejemplo claro de estos simuladores es MultiRECloudSim [30], que añade la simulación multirrecurso y de energía a CloudSim.

En este artículo avanzamos en el estado del arte mediante la diseño de un nuevo simulador enfocado en la inclusión de requisitos no funcionales (RNFs) en los datos. Esto incluye la estimación de costos para preprocesar los datos, incluyendo la incorporación de características no funcionales, tales como la seguridad y la confiabilidad de los datos. Este simulador incluye la implementación de patrones de paralelismo para estimar el procesamiento de lotes de datos a través de infraestructuras en el edge, en fog, en cloud o en ambientes HPC.

III. FedFlow

El simulador propuesto en este artículo se basa en FedFlow [31], una plataforma federada de distribución de contenidos. Esta plataforma permite la construcción de servicios de intercambio de datos en entornos de cómputo de alto rendimiento, así como en entornos altamente distribuidos, como el cómputo continuo.

En FedFlow, un cliente implementa distintos *esquemas para la gestión de requisitos no funcionales (RNFs)*, que permiten procesar los datos antes de su transmisión a través de la red para su compartimiento o almacenamiento en ubicaciones remotas. Estos esquemas se estructuran como tuberías que contienen n filtros de procesamiento, donde cada filtro es un RNF. Actualmente FedFlow considera los siguientes RNFs:

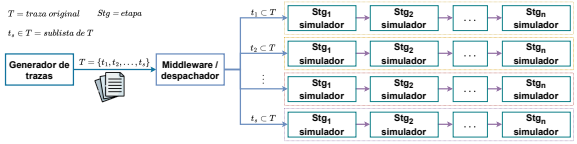


Fig. 1: Representación conceptual del simulador.

- Confiabilidad de los datos mediante el algoritmo de dispersión de información (IDA) [32]. Este RNF divide un objeto en n fragmentos redundantes, donde el objeto original puede ser reconstruido con k fragmentos de los n fragmentos originales (donde $k < n$). Esto permite mitigar la indisponibilidad de los datos causada por fallas en la infraestructura de almacenamiento o en la transmisión del objeto.
- Seguridad (conformada por los RNFs de confidencialidad, integridad y control de acceso) para hacer frente a los retos de almacenar y transmitir datos en entornos públicos o sin control. Este RNF se proporciona mediante algoritmos de hashing, como SHA-3 [33], y cifradores de datos, como CP-ABE [34] y AES [8].
- Costo-eficiencia para reducir el volumen de datos a transmitir. Esto se logra mediante técnicas de deduplicación y compresión de datos (como LZ4 [35]), lo que permite la reducción del volumen de datos a almacenar y transmitir.

Finalmente, para reducir el tiempo de procesamiento de los datos, FedFlow implementa patrones de paralelismo tanto en tareas como en datos. Esto permite crear n tuberías paralelas para procesar datos de forma concurrente.

IV. Diseño del simulador

Esta sección describe un simulador para estimar los costes generados por la adición de RNFs en aplicaciones distribuidas en el cómputo continuo. Para ello, el simulador estima el tiempo de ejecución y los costes de las operaciones de preparación de datos para los esquemas de manejo de RNFs, así como de los patrones de paralelismo aplicados para escalar dichos esquemas. La Figura 1 muestra una representación conceptual del diseño del simulador presentado en este artículo. Como se ilustra, la simulación sigue un enfoque modular que facilita la inclusión o el reemplazo de etapas o filtros *Stg* en la tubería. El número y los tipos de etapas, así como las características de los patrones de paralelismo, son ejemplos claros de parámetros que pueden ajustarse en el simulador.

Utilizando los modelos probabilísticos descritos en este artículo, es posible definir diferentes bancos de pruebas (*benchmarks*). El simulador incorpora un *generador de trazas* diseñado para sensores IoT. Este generador crea trazas de datos que incluyen el tiempo entre llegadas y el tamaño de los registros producidos por los sensores. Además, los usuarios pueden configurar el tamaño de los datos, ya sea a través de un parámetro estático o empleando una distribución normal. En este contexto, se asume que el perfil de tráfico para cada sensor puede ser gestionado por la tecnología inalámbrica subyacente, como IEEE 802.15.4. Esto se realiza durante cada período de transmisión de cada sensor. Adicionalmente, se asume que la densidad de sensores permite un acceso libre de contención al medio de comunicación.

Por lo tanto, el tiempo entre llegadas de la información depende exclusivamente del período de transmisión y del retardo de la misma. En este contexto, los sensores están organizados en una topología de red en estrella con el nodo sumidero (*sink*) conectado a una etapa (*Stg*) del esquema. Además, cada sensor incluye un número de secuencia y una marca de tiempo (*timestamp*) en cada paquete transmitido para facilitar el control de pérdida de paquetes.

Este generador se encuentra conectado a un *despachador de carga de trabajo de datos*. Este despachador recibe las trazas y prepara el entorno de simulación, actuando como *middleware* entre ellas y las tuberías de *Stg*. El *middleware* particiona el conjunto de trazas ($\mathbf{T}$) en s subconjuntos ($\mathbf{t_s} \subset \mathbf{T}$). Luego, cada subconjunto se distribuye entre s tuberías y las *entidades Stg* de manera balanceada.

Un algoritmo de balanceo de carga selecciona dos entidades de las s candidatas de manera pseudoaleatoria, compara sus cargas de trabajo actuales y asigna la traza a la entidad con la carga de trabajo más baja [36]. Este *middleware* se implementa mediante la ejecución de n contenedores virtuales (VCs), en los que cada VC encapsula una tubería compuesta por múltiples *Stgs*. Cada *Stg* simula una etapa específica, la cual realiza una tarea de procesamiento en los esquemas de preparación/recuperación para el aprovisionamiento de RNFs. La Figura 1 muestra una representación conceptual de una tubería compuesta por estos *Stgs* en el simulador.

Adicionalmente, el simulador cuenta con un *módulo de estimación de tiempo de servicio* para cada tubería. Este módulo recibe el tamaño de los datos como parámetro de entrada para generar un tiempo de servicio interpolado para dicho tamaño. Además, proporciona el tiempo entre llegadas y el tiempo de servicio para cada solicitud. Luego, estos datos se envían a un módulo de simulación de colas, el cual se basa en una distribución exponencial [37]. Así, el módulo de simulación de colas devuelve el tiempo de servicio, que ahora incluye la contención de la cola, el cual se acumula con el tiempo de servicio de la *Stg* anterior, si aplica.

A. Interpolación del tiempo de servicio de las etapas de procesamiento (Stg) para un bloque de datos

El tiempo que tarda una etapa en procesar datos se calcula utilizando una interpolación lineal de los valores de las tablas que representan el comportamiento del sistema, de la siguiente manera:

$$f(x) = \left\{ \frac{x - x_1}{x_0 - x_1} \times (y_0 - y_1) \right\} + y_1, \qquad (1)$$

donde x representa el tamaño de los datos de entrada a procesar, mientras que x_0 y x_1 definen el rango para x, tal que $x_0 < x < x_1$. Los valores y_0 y y_1 corresponden a los tiempos de servicio para los tamaños de

entrada x_0 y x_1, respectivamente. Es crucial tener en cuenta que y_0 y y_1 dependen de la etapa específica de los esquemas de preparación/recuperación de datos, así como del tipo de sistema en el que se ejecuta la simulación. Por ejemplo, si necesitamos obtener el tiempo de servicio para un bloque de datos de 60 MB en una etapa de compresión, y se conocen los valores de referencia del tiempo de servicio para bloques de datos de 1 MB y 100 MB, podemos aplicar la ecuación de interpolación sustituyendo los valores de (x_0, y_0), (x_1, y_1) y x de la siguiente manera:

$$\begin{aligned} x_1 &= 100 \text{ MB} \\ y_1 &= 114 \text{ segundos} \\ y_0 &= 20 \text{ segundos} \\ x &= 60 \text{ MB} \end{aligned} \tag{2}$$

Ahora, sustituimos estos valores en la fórmula:

$$f(60) = \left\{ \frac{60-100}{1-100} \times (20-114) \right\} + 114, \tag{3}$$

y resolvemos paso a paso:

$$\begin{aligned} \frac{60-100}{1-100} &\approx 0{,}4040 \\ (20-114) &= -94 \\ 0{,}4040 \times -94 &\approx -37{,}976 \\ f(60) &= -37{,}976 + 114 \approx 76{,}024 \end{aligned} \tag{4}$$

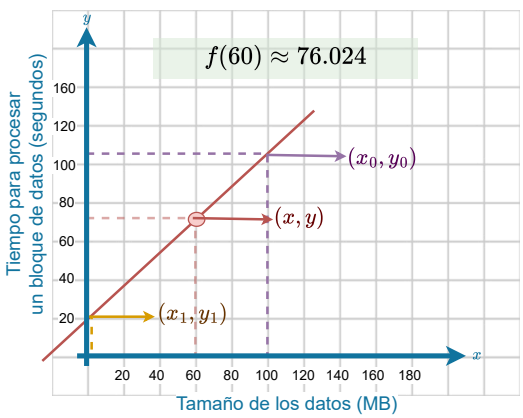


Fig. 2: Ejemplo de cálculo basado en la interpolación del tiempo de servicio para un bloque de datos en la etapa de compresión.

Por lo tanto, el tiempo requerido para procesar un bloque de datos de 60 MB, solamente en la etapa de compresión con la que estamos tratando, es de aproximadamente 76,024 segundos. La Figura 2 muestra una representación gráfica del ejemplo de interpolación descrito anteriormente.

Es importante destacar que los valores de referencia utilizados en el simulador se obtuvieron a través de pruebas exhaustivas realizadas con la plataforma de *FedFlow*. Esto se hizo para cada una de las etapas (Stg) del sistema para plataformas de hardware reales, en las que se incluye principalmente la adición de los RNFs de seguridad, confiabilidad, eficiencia e integridad. Sin embargo, estas tablas deben redefinirse para plataformas que incorporen nuevas características o para variantes de sistemas existentes, como cambios en la configuración de la red o en la capacidad de memoria. Esta redefinición requiere realizar nuevas pruebas en los esquemas de preparación y de recuperación de datos.

B. Simulación de la cola de espera en una etapa (Stg)

Esta sección describe el diseño del algoritmo del simulador de colas (ver Algoritmo 1), desarrollado para estimar la utilización promedio del servidor, el tiempo promedio de simulación, el número de operaciones en la cola y el retardo promedio de la cola en una etapa (Stg). Este algoritmo utiliza una función de generación de variables exponenciales que genera una variable aleatoria uniforme ($\mathcal{U}(\prime, \infty)$). Después, este algoritmo devuelve la variable aleatoria exponencial correspondiente ($-mean * log(u)$), donde el valor resultante es un número aleatorio que sigue una distribución exponencial con la media dada. El Algoritmo 1 fue implementado en C.

Algorithm 1 QueueSimulator(m_i, m_s, n_d)

Require: media entre llegadas (m_i), número de retardos requeridos (n_d), tiempo medio de servicio (m_s), total de peticiones en la cola ($Q_{LIMIT} = 100$), número de eventos ($n_e = 2$) , semilla uno ($s_1 = 99275{,}0$), semilla dos ($s_2 = 48612{,}0$), LIBRE ($IDLE = 0$), OCUPADO ($BUSY = 1$)

1: {Inicializa la simulación.}
2: {Aleatoriedad del tiempo entre llegadas y servicio.}
3: $yA = s_1, y2 = s_2$;
4: {Continua si el número de programas retrasados sea menor o igual a los retrasos requeridos.}
5: {Bucle de eventos}
6: **while** $num_progs_delayed \leq n_d$ **do**
7: {Determina el siguiente evento y actualiza acumuladores estadísticos de promedio temporal}
8: **switch** (next_event)
9: {Evento de llegada}
10: **case** 1**:**
11: time_next_event[1] = $time$ + $expon(m_i, yA)$;{Programa nueva llegada. Suma el tiempo exponencial entre llegadas al tiempo actual.}
12: **if** $server_status == BUSY$ **then**
13: $++num_in_q$; {Servidos ocupado, el cliente se añade a la cola.}
14: **else**
15: {Servidor libre, el cliente que llega tiene un retraso de cero.}
16: $num_progs_delayed+ = 1$;
17: $server_status = BUSY$;
18: $time_next_event[2]$ = $time$ + $expon(m_s, y2)$; {Finaliza servicio}
19: **end if**

```
    {Evento de salida}
    case 2:
      if num_in_q == 0 then
        server_status = IDLE;{Si la cola está vacía, estado del servidor = LIBRE.}
        time_next_event[2] = 1,0e + 30;{Se indica que no hay ningún evento programado actualmente.}
      else
        {La cola no está vacía, se reduce el número de clientes en la cola.}
        delay = time − time_arrival[1];
        total_of_delays+ = delay;{El retardo se añade al acumulador de retardo total.}
        num_progs_delayed+ = 1;
        time_next_event[2] = time + expon(m_s, y2);{Se programa una nueva salida sumando un tiempo de servicio al tiempo actual.}
      end if
      {Avanza cada cliente en la cola (si lo hay) un lugar.}
    end switch
  end while
```

Al final de la simulación, el *middleware* genera un informe que contiene todos los resultados obtenidos por cada *Stg*, ya sea en una tubería o en un patrón de paralelismo dentro de los esquemas.

El informe resultante presenta varias métricas clave, incluyendo el tiempo de espera en la cola para las solicitudes manejadas por cada *Stg*, el número de solicitudes esperando en cada cola, el tiempo promedio de simulación para cada *Stg* y las tasas de utilización de cada *Stg*. Además, describe el tiempo de respuesta requerido por la tubería para procesar cada traza. Las características principales de este algoritmo se pueden resumir de la siguiente manera:

- El sistema avanza de un evento (llegada o salida) al siguiente sin incrementos de tiempo fijos (método *time-stepping*).
- Se realiza una simulación realista de comportamientos de la cola, incluyendo los tiempos de espera y de servicio del sistema.
- Los tiempos entre llegadas y de servicio siguen una distribución exponencial, como es típico en la teoría de colas.
- Permite analizar la carga del servidor, los patrones de retardo y el rendimiento para diferentes condiciones de llegada y servicio.

C. Simulación de una tubería de software

La plataforma de FedFlow en la que se basa nuestro simulador incluye tres planos principales, los cuales incluyen tuberías para la preparación y recuperación de datos: *i)* plano de publicación/suscripción (*pub/sub*), *ii)* plano de control y *iii)* plano de preservación. El plano de preservación es el factor que más contribuye a los costes de tiempo de respuesta de una tubería de software.Como se mencionó anteriormente, cada *Stg* puede conectarse a otra *Stg* para crear, ya sea, una tubería o un patrón de paralelismo.

Además, los esquemas de preparación y recuperación de datos están diseñados como una tubería de procesamiento. En ellos, un conjunto de *Stg*s de procesamiento (P-Stgs = $\{\text{pStg}_1, \ldots, \text{pStg}_\text{i}\}$) se interconectan para formar patrones específicos entre los que se incluyen el patrón de tuberías, el patrón divide y vencerás (*divide&conquer*) y el patrón gestor/trabajador (*manager/worker*). El tiempo de servicio producido por un bloque de procesamiento ($pStg_i$) se basa en el modelo ETL (extracción, transformación, carga) y consta de tres componentes principales: *i)* tiempo de lectura de datos de entrada (IT_{pStg_i}), *ii)* tiempo de procesamiento de datos (ST_{pStg_i}) y *iii)* tiempo de escritura de datos de salida (OT_{pStg_i}). Por lo tanto, el tiempo de servicio total (ST_p) de una tubería se calcula como la suma de los tiempos de servicio de todas las etapas de procesamiento dentro de la tubería, el cual se puede calcular de la siguiente manera:

$$ST_p = \sum_{i=1}^{|\text{P-Stg}|} ST_{pStg_i}. \tag{5}$$

Los tiempos totales de entrada (IT) y de salida (OT) de una tubería se pueden calcular como la suma del tiempo de entrada (IT_{pStg_i}) y el tiempo de salida (OT_{pStg_i}) de cada $pStg_i$, de la siguiente manera:

$$IT = \sum_{i=1}^{|\text{P-Stg}|} IT_{pStg_i}, \tag{6}$$

$$OT = \sum_{i=1}^{|\text{P-Stg}|} OT_{pStg_i}. \tag{7}$$

Como resultado, el tiempo observado por el usuario final durante la ejecución de una tubería corresponde al tiempo de respuesta (RT) calculado como la suma de los tiempos ST, IT y OT de la siguiente manera:

$$RT = IT + ST + OT. \tag{8}$$

La Figura 3 presenta un diagrama conceptual que ilustra cómo se calcula el tiempo total de respuesta (RT) de una tubería de procesamiento de datos mediante su descomposición. La tubería está compuesta por una secuencia de etapas de procesamiento ($pStg_i$), donde cada etapa procesa una porción del flujo de datos. Cada etapa ($pStg_1, pStg_2, \ldots, pStg_i$) contribuye a los siguientes componentes de tiempo:

- **Tiempo de entrada** (IT_{pStg_i}): Tiempo requerido para leer los datos de entrada.
- **Tiempo de procesamiento** (ST_{pStg_i}): Tiempo invertido en el procesamiento de los datos.
- **Tiempo de salida** (OT_{pStg_i}): Tiempo necesario para escribir los datos resultantes.

D. Simulación de patrones paralelos

Esta sección describe el esquema de simulación desarrollado mediante patrones de paralelismo, específicamente los modelos de divide y vencerás (*divide&conquer*) y gestor/trabajador (*manager/worker*).

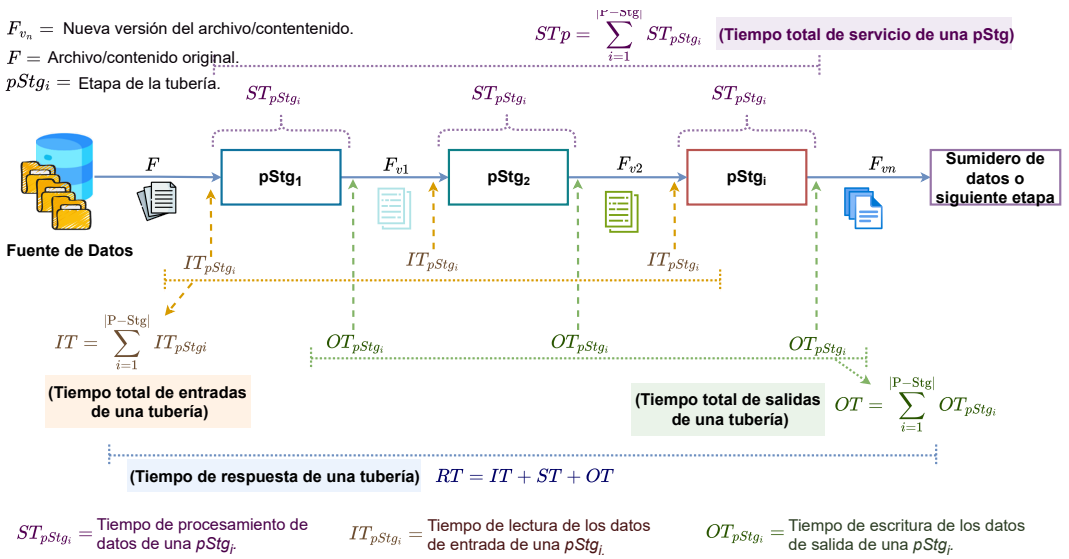


Fig. 3: Representación conceptual de las fórmulas utilizadas para estimar el tiempo de respuesta de una tubería.

D.1 Simulación Gestor/Trabajador (Manager/Worker)

El patrón de procesamiento paralelo, conocido como Gestor/Trabajador, consta de dos roles principales: el gestor (M) y los trabajadores ($\mathrm{W} = \{\mathrm{w}_1, \ldots, \mathrm{w}_\mathrm{n}\}$), donde n representa el número de trabajadores ($Stgs$) ($n \leq 1$). Dada una lista de contenidos de entrada ($\mathrm{L} = \{\mathrm{co}_1, \ldots, \mathrm{co}_\mathrm{o}\}$) que contiene o elementos con $o > 0$, se consideran tres componentes clave de tiempo en el tiempo de servicio del gestor:

i) el tiempo empleado por el gestor para leer la lista de contenidos de entrada (L), denotado como $RSource$;
ii) el tiempo requerido para generar una sublista de contenido ($\mathrm{l_{w_j}} = \{\mathrm{c}_1, \ldots, \mathrm{c_p}\}$) para cada trabajador en el patrón (LB). Esto se realiza de forma balanceada, utilizando el algoritmo de dos opciones (*two choices algorithm*) [36]. Donde p es el tamaño de la sublista ($\mathrm{l_{w_j}}$) y $0 \leq p < o$; y
iii) el tiempo empleado por el gestor para entregar j sublistas a los j trabajadores (LDW).

Así, el tiempo de servicio del gestor se define como:

$$MT = RSource + LB + LDW. \tag{9}$$

El tiempo de servicio de un trabajador (ST_{w_j}) se determina sumando el tiempo de entrada ($IT_{w_j,c}$), el tiempo de procesamiento ($ST_{w_j,c}$) y el tiempo de salida ($OT_{w_j,c}$) para cada elemento de contenido en la lista ($\mathrm{l_{w_j}}$), de la siguiente manera:

$$ST_{w_j} = \sum_{c \in \mathrm{L_{w_j}}} \left[IT_{w_j,c} + ST_{w_j,c} + OT_{w_j,c} \right]. \tag{10}$$

Luego, el tiempo total de servicio de los trabajadores es determinado por el trabajador más lento, calculado de la siguiente manera:

$$ST_W = \max_{j=1}^{|\mathrm{W}|} (ST_{w_j}). \tag{11}$$

Finalmente, el tiempo de servicio total para todo el patrón corresponde a la suma del tiempo de servicio del gestor (MT) y los tiempos de servicio de los trabajadores (ST_W), de la siguiente manera:

$$ST_{MW} = MT + ST_W. \tag{12}$$

La Figura 4 presenta una representación conceptual de un patrón Gestor/Trabajador (MW) para procesar una lista de datos en paralelo. Demuestra cómo se distribuye el tiempo en los diferentes componentes del sistema para calcular el tiempo de servicio total.

En la figura, las flechas indican el flujo desde el origen hacia el gestor, luego hacia múltiples trabajadores y, finalmente, al sumidero de datos (*data sink*). Los corchetes discontinuos de colores agrupan los componentes de tiempo (MT y ST_W).

En resumen, el diagrama modela un sistema de procesamiento en paralelo utilizando el patrón Gestor/Trabajador, en el que el gestor lee y distribuye la carga de trabajo, y los trabajadores procesan sus respectivas particiones de datos. El rendimiento general del sistema está limitado por el trabajador más lento, y las fórmulas que lo acompañan ayudan en el análisis y la optimización de la tubería de procesamiento.

D.2 Simulación del patrón Divide y Vencerás (Divide&Conquer)

El patrón paralelo Divide y Vencerás consta de tres tiempos de servicio principales:

i) el tiempo requerido para que el patrón divida un elemento de contenido c en k segmentos, denominado $DivTime$, donde k representa el número de segmentos, y $k \geq 1$;
ii) el tiempo empleado por cada trabajador en el patrón para procesar un segmento, denotado como ST_{DC_k}, donde k es el índice del trabajador en el patrón; y

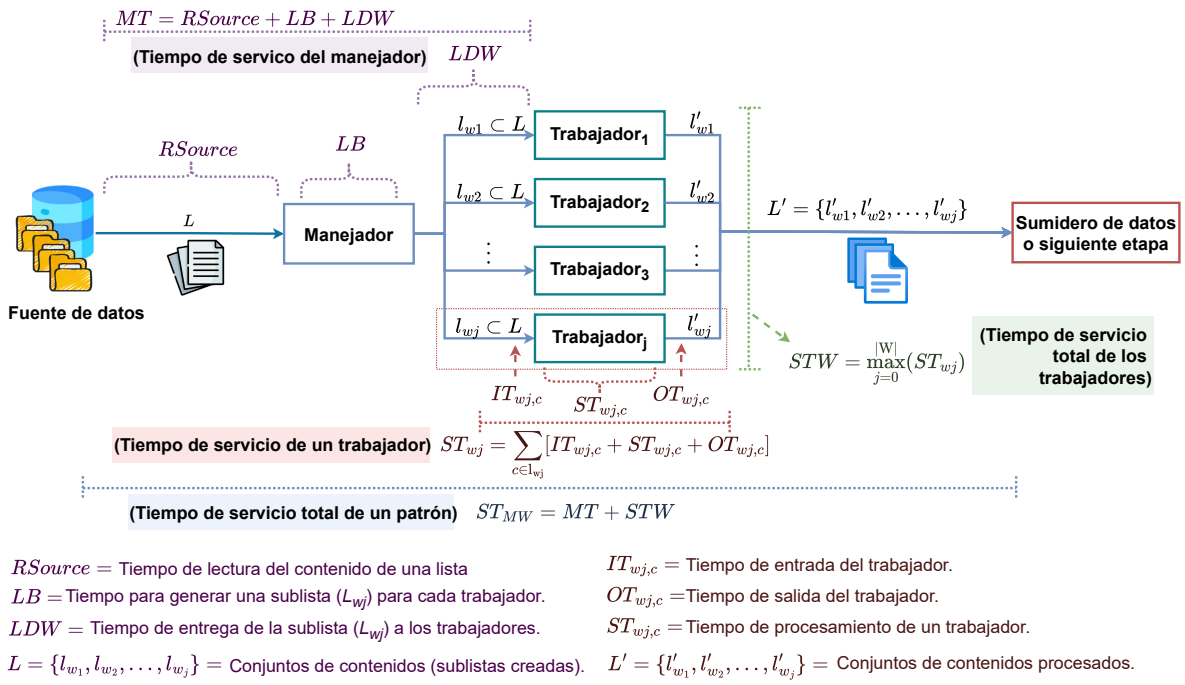


Fig. 4: Tubería de procesamiento de datos con el patrón Gestor-Trabajador (MW).

iii) el tiempo empleado por el patrón para integrar los segmentos procesados, conocido como *ConqTime*, en una nueva versión del contenido original (c').

El *DivTime* se refiere al tiempo empleado por el patrón para leer el contenido c, mientras que el *ConqTime* se refiere al tiempo empleado por el patrón para escribir el nuevo contenido c' ya sea en memoria o en el sistema de archivos. Por lo tanto, el tiempo de servicio total del patrón se define de la siguiente manera:

$$ST_{DC} = DivTime + \sum_{k=1}^{|S|} ST_{DC_k} + ConqTime, \quad (13)$$

donde $S = \{s_1, \ldots, s_k\}$ es el conjunto de segmentos de contenido.

Los componentes principales de este patrón se enumeran a continuación:

- **Segmentación:** Un archivo S se divide en segmentos $s_1, s_2, \ldots, s_k \subset S$. Esto corresponde al tiempo de segmentación (*DivTime*).
- **Trabajadores:** Cada segmento s_k es procesado de forma independiente por un trabajador.
- **Etapa de conquista:** Todos los segmentos procesados s'_k se combinan en una única salida S'. Esto corresponde al tiempo de integración (*ConqTime*).
- **Sumidero de datos o siguiente etapa:** Recibe el resultado final procesado S'.

En este patrón, el cuello de botella está determinado por el trabajador más lento (es decir, el tiempo de procesamiento máximo). El modelo es ideal para problemas que pueden dividirse en segmentos independientes, procesarse en paralelo y recombinarse posteriormente. Este patrón permite el paralelismo al dividir los datos en fragmentos (*chunks*), permitiendo a los trabajadores procesarlos de forma aislada y, finalmente, reensamblar los resultados. Se utiliza ampliamente en marcos de procesamiento de datos a gran escala (por ejemplo, MapReduce, análisis paralelo, etc.).

E. Validación del simulador

El simulador fue validado comparando los tiempos de servicio y respuesta generados por cada etapa (*Stg*) del simulador con las métricas correspondientes obtenidas directamente con FedFlow. Por ejemplo, el generador de datos IoT se evaluó frente a datos reales capturados de sensores de dispositivos IoT, y los tamaños de los datos se derivaron de muestras representativas extraídas de conjuntos de datos de estaciones terrestres.

Los tiempos de servicio de los contenedores virtuales de las *Stg*s, tales como integridad (MD5, SHA3-256), confiabilidad (IDA) y compresión (LZ4), se compararon con las implementaciones de estos componentes de software desplegados en contenedores virtuales reales. Las *Stg*s incluidas en este simulador se ajustaron con precisión (*fine-tuned*) basándose en los resultados obtenidos por el prototipo.

V. Resultados experimentales

Para validar el simulador con diferentes cargas de trabajo, realizamos pruebas a gran escala con el simulador desarrollado en un clúster de cómputo de alto rendimiento denominado C3. Este clúster se encuentra ubicado en la Universidad Carlos III de Madrid (UC3M), en España. Para esta validación,

evaluamos muestras correspondientes a 100, 1,000, 10,000, y 100,000 trazas, todas las cuales simulan datos ambientales con diferentes configuraciones para los trabajadores (1, 5, 10, 20, 40, y 80) para su procesamiento durante la simulación de las diferentes etapas de preparación de datos y el proceso de carga de datos en una tubería. Para esta tubería, consideramos las etapas de compresión, hashing, indexación y codificación para agregar eficiencia, confiabilidad y seguridad a los datos. Para esta evaluación, utilizamos AppTainer como herramienta de contenerización y Slurm para reservar nodos de cómputo en el clúster.

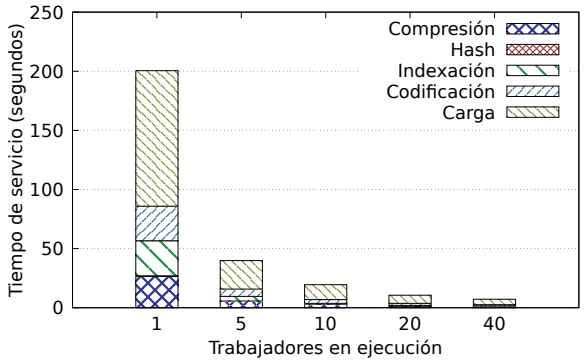


Fig. 5: Tiempo de servicio simulado para cada etapa al procesar un conjunto de 100 muestras.

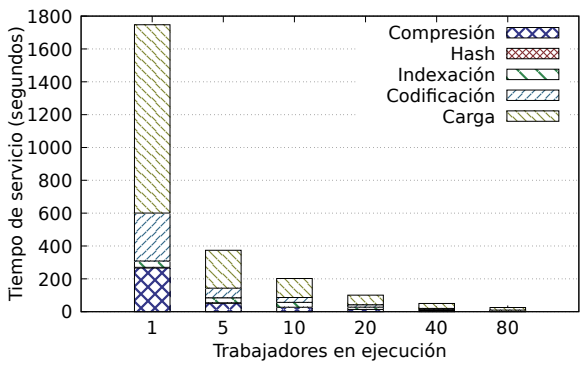


Fig. 6: Tiempo de servicio simulado para cada etapa al procesar un conjunto de 1,000 muestras.

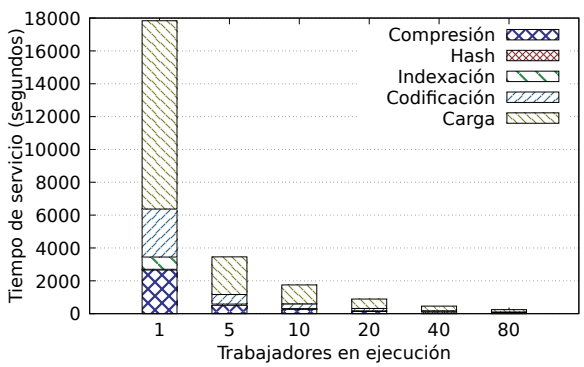


Fig. 7: Tiempo de servicio simulado para cada etapa al procesar un conjunto de 10,000 muestras.

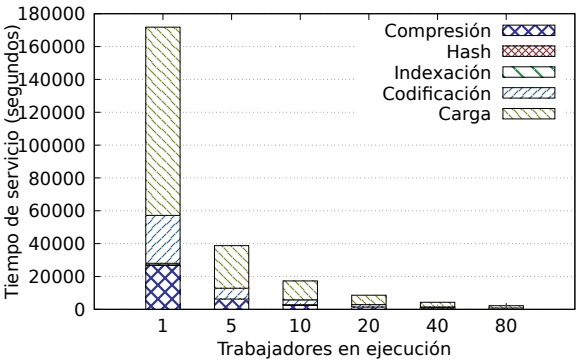


Fig. 8: Tiempo de servicio simulado para cada etapa al procesar un conjunto de 100,000 muestras.

Tabla I: Porcentaje de ganancia obtenido para cada una de las configuraciones al procesar 100, 1,000, 10,000, y 100,000 muestras.

Porcentaje de ganancia %				
Trabajadores	100	1,000	10,000	100,000
1	-	-	-	-
5	80.130	78.598	79.753	77.289
10	90.274	88.427	89.767	89.872
20	94.716	94.228	94.781	94.971
40	96.429	97.138	97.307	97.474
80	97.286	98.535	98.556	98.725

Las Figuras 5, 6, 7 y 8 muestran el tiempo de servicio simulado para cada etapa en la fase de preparación y carga de datos para 100, 1,000, 10,000 y 100,000 muestras, respectivamente. El eje horizontal (eje X) muestra el número de trabajadores en ejecución, mientras que el eje vertical (eje Y) muestra el tiempo de servicio en segundos para cada configuración simulada. Se puede observar que los resultados del simulador ofrecen un rendimiento similar al de los patrones paralelos evaluados en implementaciones reales. Se observa que el tiempo de servicio se reduce a medida que se agregan más trabajadores a los patrones de paralelismo, lo cual permite el procesamiento de una mayor cantidad de datos en una ventana de tiempo reducida. Por ejemplo, si realizamos una comparación del tiempo de servicio al procesar 100 trazas con un solo trabajador (Figura 5) con el tiempo de servicio obtenido para el procesamiento de 1,000 trazas con 10 trabajadores (Figura 6), es posible observar que ambos tienen un tiempo de respuesta similar alrededor de 200 segundos.

La Tabla I muestra el porcentaje de ganancia en el rendimiento obtenido durante el procesamiento de las muestras para cada una de las configuraciones evaluadas en comparación con la configuración de un solo trabajador. Como es posible observar, el rendimiento en el tiempo de respuesta presenta una ganancia de hasta el 90 al 98 % para cada una de las etapas de procesamiento cuando se utilizan 80 trabajadores para cada conjunto de 100, 1,000, 10,000, y 100,000 muestras. La Figura 9 muestra el tiempo de respuesta en el eje vertical, para cada una de las configuraciones (en el eje horizontal x) para cada uno de los conjuntos de muestras. Como es posible observar, el número de archivos procesados afecta el rendimiento de la solución. Además, se puede obser-

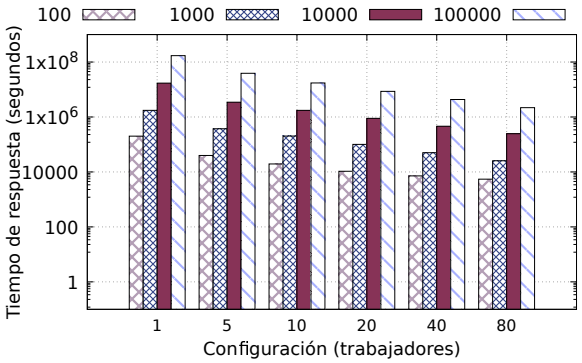


Fig. 9: Comparación entre el tiempo de servicio total para cada prueba de 100, 1,000, 10,000 y 100,000 muestras para 1, 5, 10, 20, 40 y 80 trabajadores.

Tabla II: Error porcentual (%) para cada configuración de cada muestra.

	Number of samples		
Workers	100	1,000	10,000
1	0.004	0.002	0.003
5	1.536	0.250	0.000
10	3.139	0.255	0.029
20	4.889	0.941	0.077
40	0.014	2.122	0.195
80	9.521	0.002	0.281

var que cuando la carga de trabajo se incrementa en un orden de magnitud, el tiempo de servicio obtenido en la simulación, también aumenta en un orden de magnitud, siguiendo así un comportamiento lineal. Este comportamiento es el esperado al utilizar una interpolación lineal, en la que uno de los parámetros a considerar es el tamaño de los archivos.

Finalmente, la Tabla II muestra el error porcentual en el tiempo de respuesta observado durante la ejecución de múltiples simulaciones para la preparación y carga de datos en ambientes de HPC. El error fue calculado utilizando tres subconjuntos de archivos de igual tamaño, cada uno compuesto de 100, 1,000, y 10,000 muestras, y configurados para procesar los datos con 1, 5, 10, 20, 40, y 80 trabajadores. Como se puede observar, los valores con un error porcentual en los rangos de $0-1\,\%$ indican que casi no hay dispersión alrededor de la mediana de los tiempos obtenidos por el simulador. El error en el rango del $1-5\,\%$ indica cierta variabilidad entre los tiempos obtenidos; sin embargo, esta variabilidad se encuentra dentro de un rango razonable. Como se observa, el error del simulador varía en función del tamaño de las muestras que se van a procesar. En general, la muestra de 10,000 trazas presenta los errores medios más bajos (aproximadamente $< 1\,\%$ en valor absoluto) en cada configuración, mientras que la muestra de 100 trazas muestra errores más altos (entre $1\,\%$ y $9\,\%$) para cada configuración, respectivamente.

VI. Conclusiones

En este artículo se ha presentado el diseño de un simulador para estimar los costos del manejo de requisitos no funcionales (RNFs) en sistemas de manejo de datos distribuidos. Nos enfocamos principalmente en los requerimientos de disponibilidad, seguridad, confiabilidad y eficiencia. Se presentó el método empleado para interpolar el tiempo de servicio de cada etapa en un flujo de datos, simular las colas de las etapas y la ejecución de la tubería, así como la evaluación de patrones de paralelismo utilizados para hacer frente a la sobrecarga provocada por el aprovisionamiento de los RNFs. Finalmente, se expuso el proceso de validación del simulador.

La principal contribución de este simulador radica en permitir a desarrolladores y organizaciones estimar los requisitos de almacenamiento, el rendimiento del servicio y el impacto de aplicar esquemas de seguridad y confiabilidad al transporte de datos. Además, este simulador admite explícitamente el modelado y la evaluación de servicios continuos de intercambio de datos a través de infraestructuras heterogéneas de edge–fog–cloud. Integra aspectos de seguridad, sincronización y gobernanza en el proceso de simulación mediante tuberías configurables, lo que permite evaluar RNFs como la disponibilidad, la confiabilidad, la seguridad y la eficiencia bajo diversas condiciones de despliegue.

Al incorporar aspectos tradicionalmente abordados por simuladores independientes, incluidos modelos de almacenamiento, asignación de tareas y cumplimiento de RNFs, este simulador proporciona un entorno de evaluación holístico y transversal, ampliando así el estado del arte hacia una experimentación integral y la validación del diseño de sistemas seguros, a gran escala y de intercambio continuo de datos.

La evaluación experimental realizada con el simulador para el preprocesamiento, procesamiento, almacenamiento y compartición de datos en entornos de continuo demostró su precisión y utilidad para predecir el rendimiento del sistema, al tiempo que permitió identificar oportunidades para mejorar la modelización de las interacciones de las colas y la contención de recursos en procesos computacionalmente exigentes.

Como trabajo futuro, se plantea la incorporación de técnicas avanzadas de distribución de carga, teniendo en cuenta restricciones multidimensionales como la disponibilidad de ancho de banda, el consumo energético y las limitaciones presupuestarias.

Agradecimientos

El presente trabajo ha sido financiado por la Agencia Española de Investigación a través del proyecto “Nuevas técnicas escalables de E/S para cargas de trabajo híbridas de HPC e intensivas en datos (SCIOT)” con referencia PID2022-138050NB-I00 y parcialmente por la Universidad Carlos III de Madrid a través del proyecto “Nuevas técnicas para la gestión de fábricas de datos en el cómputo continuo” (2025/00822/001 - Proyectos JÓVENES PPIT2025)

Referencias

[1] Petroc Taylor, “Data growth worldwide 2029,” Nov 2025.

[2] Petroc Taylor, "Iot connections worldwide 2034," Jan 2026.

[3] Hardial Singh, "Generative ai for synthetic data creation: Solving data scarcity in machine learning," *Available at SSRN 5267914*, 2015.

[4] Pablo Dúo Terrón, "Generative artificial intelligence: Educational reflections from an analysis of scientific production," 2024.

[5] Julia Wendling, "Visualized: All of the world's data," Mar 2025.

[6] Schahram Dustdar, Victor Casamayor Pujol, and Praveen Kumar Donta, "On distributed computing continuum systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 4092–4105, 2022.

[7] Luiz F Bittencourt, Roberto Rodrigues-Filho, Josef Spillner, Filip De Turck, José Santos, Nelson LS da Fonseca, Omer Rana, Manish Parashar, and Ian Foster, "The computing continuum: Past, present, and future," *Computer Science Review*, vol. 58, pp. 100782, 2025.

[8] M. Morales, J. L. Gonzalez, A. Diaz, and V. J. Sosa, "A pairing-based cryptographic approach for data security in the cloud," *IJISP*, vol. 17, no. 4, pp. 441–461, 2018.

[9] J. Zhang and Z. Zhang, "Secure and efficient data-sharing in clouds," *CCPE*, vol. 27, no. 8, pp. 2125–2143, 2015.

[10] J. L. Gonzalez, V. Sosa, A. Diaz, J. Carretero, and J. Yanez0, "Sacbe: A building block approach for constructing efficient and flexible end-to-end cloud storage," *Journal of Systems and Software*, vol. 135, pp. 143–156, 2018.

[11] B. Mao, S. Wu, and H. Jiang, "Improving storage availability in cloud-of-clouds with hybrid redundant data distribution," in *IPDPS 2015m*. IEEE, 2015, pp. 633–642.

[12] H. Xiong, X. Zhang, W. Zhu, and D. Yao, "Cloudseal: End-to-end content protection in cloud-based storage and delivery services," in *SecureComm*. Springer, 2011, pp. 491–500.

[13] Haryadi S Gunawi, Mingzhe Hao, Riza O Suminto, Agung Laksono, Anang D Satria, Jeffry Adityatama, and Kurnia J Eliazar, "Why does the cloud stop computing?: Lessons from hundreds of service outages," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*. ACM, 2016, pp. 1–16.

[14] Justice Opara-Martins, Reza Sahandi, and Feng Tian, "Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective," *Journal of Cloud Computing*, vol. 5, no. 1, pp. 4, 2016.

[15] Kriti Bhushan and Brij B Gupta, "Security challenges in cloud computing: state-of-art," *International Journal of Big Data Intelligence*, vol. 4, no. 2, pp. 81–107, 2017.

[16] Y. Miao, W. Li, D. Tian, M. Hossain, and M. Alhamid, "Narrowband internet of things: simulation and modeling," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2304–2314, 2017.

[17] Polley, Blazakis, McGee, Rusk, and Baras, "Atemu: a fine-grained sensor network simulator," in *SECON 2004*. IEEE, 2004, pp. 145–152.

[18] Acharya, Ranjan, and Mandal, "A novel approach for deployment and throughput analysis of lte-advanced self-organizing network by using system vue," *International Journal of Computational Intelligence & IoT*, vol. 1, no. 1, 2018.

[19] S. Sotiriadis, N. Bessis, E. Asimakopoulou, and N. Mustafee, "Towards simulating the internet of things," in *2014 AINA*. IEEE, 2014, pp. 444–448.

[20] H. Gupta, A. Vahid Dastjerdi, S.K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[21] A. Verma, L. Cherkasova, and R.H. Campbell, "Play it again, simmr!," in *2011 IEEE International Conference on Cluster Computing*. IEEE, 2011, pp. 253–261.

[22] A.C. Murthy, "Mumak: Map-reduce simulator," *MAPREDUCE-728, Apache JIRA*, 2009.

[23] S. Hammoud, M. Li, Y. Liu, N.K. Alham, and Z. Liu, "Mrsim: A discrete event based mapreduce simulator," in *2010 FSKD*. IEEE, 2010, vol. 6, pp. 2993–2997.

[24] Nitinder Mohan and Jussi Kangasharju, "Edge-fog cloud: A distributed cloud for internet of things computations," in *2016 Cloudification of the Internet of Things (CIoT)*. IEEE, 2016, pp. 1–6.

[25] M. Tran, D. Nguyen, V. Le, D. Nguyen, and T. Pham, "Task placement on fog computing made efficient for iot application provision," *Wireless Communications and Mobile Computing*, vol. 2019, 2019.

[26] Y. Song, S.S. Yau, R. Yu, X. Zhang, and G. Xue, "An approach to qos-based task distribution in edge computing networks for iot applications," in *2017 IEEE international conference on edge computing (EDGE)*. IEEE, 2017, pp. 32–39.

[27] R.N. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[28] Gabriel G Castane, Alberto Nunez, and Jesus Carretero, "icancloud: A brief architecture overview," in *2012 IEEE 10th international symposium on parallel and distributed processing with applications*. IEEE, 2012, pp. 853–854.

[29] Zeng, Garg, Strazdins, Jayaraman, Georgakopoulos, and Ranjan, "Iotsim: A simulator for analysing iot applications," *Journal of Systems Architecture*, vol. 72, pp. 93–107, 2017.

[30] Lin, Xu, He, and Li, "Multi-resource scheduling and power simulation for cloud computing," *Information Sciences*, vol. 397, pp. 168–186, 2017.

[31] Diana Carrizales-Espinoza, Dante D Sanchez-Gallegos, JL Gonzalez-Compean, and Jesus Carretero, "Fedflow: a federated platform to build secure sharing and synchronization services for health dataflows," *Computing*, vol. 105, no. 5, pp. 1019–1037, 2023.

[32] Marcelín-Jiménez, Ramírez-Ortíz, De La Colina, Pascoe-Chalke, and González-Compeán, "On the complexity and performance of the information dispersal algorithm," *IEEE Access*, vol. 8, pp. 159284–159290, 2020.

[33] Morris J Dworkin, "Sha-3 standard: Permutation-based hash and extendable-output functions," Tech. Rep., NIST, 2015.

[34] Vanga Odelu, Ashok Kumar Das, Y Sreenivasa Rao, Saru Kumari, Muhammad Khurram Khan, and Kim-Kwang Raymond Choo, "Pairing-based cp-abe with constant-size ciphertexts and secret keys for cloud environment," *Computer Standards & Interfaces*, vol. 54, pp. 3–9, 2017.

[35] Matěj Bartík, Sven Ubik, and Pavel Kubalik, "Lz4 compression algorithm on fpga," in *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*. IEEE, 2015, pp. 179–182.

[36] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE TPDS*, vol. 12, no. 10, pp. 1094–1104, 2001.

[37] Gkoutioudi and Karatza, "Task cluster scheduling in a grid system," *Simulation Modelling Practice and Theory*, vol. 18, no. 9, pp. 1242–1252, 2010.

AudioSAFFE: Fusión Multi-Modal Eficiente con Audio mediante *Encoders* Congelados

Maithri R. Kulasekara[1], Juan F. Inglés-Romero[2], Baldomero Imbernón[3] y José L. Abellán[1]

Resumen— **Los modelos multimodales basados en *Transformers* requieren normalmente entrenamiento costoso a gran escala (i.e., servidores de altas prestaciones en entornos clusterizados como el *cloud*) sobre *datasets* específicos por tarea, multimodalmente emparejados, para alcanzar alto rendimiento y precisión. La metodología SAFFE aborda este problema al permitir construir modelos multimodales precisos mediante entrenamiento mínimo en GPUs de escritorio. Para ello, emplea *encoders* unimodales preentrenados y congelados, conectados mediante un módulo *FusionAlign* ligero (FAM), basado en *bottleneck mid-fusion*, entrenado únicamente con el *dataset* objetivo para alinear semánticamente las salidas de los *encoders*. Al evitar el entrenamiento *end-to-end* del modelo completo, SAFFE mantiene alta precisión reduciendo significativamente los costes computacionales. Sin embargo, SAFFE se limita a las modalidades texto–imagen, sin incorporar audio, cuya inclusión puede aportar mayor riqueza semántica y mejorar el rendimiento. En este trabajo introducimos AudioSAFFE, una extensión a SAFFE que integra audio mediante una FAM que soporta las tres modalidades. Este diseño permite una composición multimodal eficiente con rápida convergencia en GPUs de escritorio, incluso con datos emparejados limitados. Evaluaciones en UrbanSound8K, ESC-50, AudioSet-Balanced, Tri-Modal Balanced AudioSet y AudioSet-500K muestran que AudioSAFFE alcanza precisión competitiva en tareas *audio-centric retrieval* con menores costes de entrenamiento que los sistemas multimodales actuales.**

Palabras clave— **SAFFE, Fusión multimodal, modelo Frozen, *encoder* de *Transformers*, *decoder*, Mid Fusion, AudioSaffe**

I. Introducción

LOS sistemas multimodales son clave para modelar escenarios reales con señales sensoriales complementarias, mejorando la robustez, el razonamiento contextual y la comprensión semántica [1], [2], [3], [4], [5]. Sin embargo, los modelos más avanzados del estado del arte (*state-of-the-art*, o SOTA) multimodales dependen de entrenamiento integral a gran escala, con altos requisitos de cómputo, memoria y *datasets* multimodales masivos, requiriendo infraestructura computacional de grandes centros de procesamiento de datos [6], [4], [7], [8], [3], [5]. Estas demandas limitan la escalabilidad, elevan los costes y dificultan su desarrollo y despliegue en entornos con recursos restringidos, tales como los sistemas basados en GPU del usuario final (e.g., GPUs de escritorio).

Por otra parte, los *Transformers* unimodales alcanzan rendimiento SOTA en texto [9], [10], [11], [12], visión [13], [14], [8], [15], [16], [17] y audio [18], [19], [20], [21]. Estas arquitecturas se componen de bloques de *encoder* y/o *decoder* preentrenados en grandes corpus específicos de cada modalidad, donde aprenden representaciones latentes ricas en contenido estructural y semántico. Muchos de estos *encoders* y *decoders* están disponibles públicamente (e.g., OpenAI, Hugging Face), lo que permite su reutilización directa sin entrenamiento adicional a gran escala. En consecuencia, constituyen una base sólida para *construir sistemas multimodales mediante la composición modular de* encoders *y/o* decoders *"congelados"*.

Sin embargo, la composición sistemática de *encoders/decoders* congelados preentrenados en arquitecturas multimodales sigue estando poco explorada, especialmente para la integración de audio–imagen y audio–texto [4], [7], [1]. La mayoría de enfoques se basan en optimización conjunta o fusiones estrechamente acopladas, incluyendo *unified masked* o *contrastive objectives* (e.g., CLIP [22]) y proyecciones lineales en espacios compartidos de *embeddings* —representaciones vectoriales densas generadas por los propios *encoders/decoders* de los *Transformers*, que codifican información semántica en un espacio latente continuo— [23]. Estos métodos requieren *datasets* emparejados a gran escala y entrenamiento integral costoso en sistemas GPU multinodo durante semanas, lo que limita la reutilización de componentes unimodales preentrenados.

Una clase destacada de estos enfoques conjuntos se basa en espacios de *embedding* unificados, cuyo objetivo es reducir la duplicación de parámetros y mejorar la eficiencia mediante la coordinación del aprendizaje multimodal dentro de una representación compartida [4], [24], [3]. Sin embargo, estos modelos se enfrentan a retos inherentes, entre los que se incluyen dinámicas de aprendizaje heterogéneas entre modalidades, sesgos de representación específicos de cada modalidad, características de ruido diferentes y requisitos de alineación complejos.

SAFFE [25] aborda esta limitación mediante una metodología multimodal composicional que conecta *encoders* unimodales preentrenados y congelados a través de un componente ligero y entrenable, el módulo *FusionAlign* (FAM). En lugar de reentrenar conjuntamente todos los *encoders*, SAFFE actualiza exclusivamente el FAM empleando datos emparejados específicos de la tarea objetivo, lo que permite alinear representaciones previamente aprendidas de forma independiente mientras se mantienen fijos ("congelados") los parámetros de cada *encoder*. Este enfoque reduce de forma significativa los costes de entrenamiento y el consumo de memoria, posibilitando una construcción multimodal eficiente sobre hardware GPU estándar en escalas temporales de ho-

[1]Dpto. de Ingeniería y Tecnología de Computadores, Universidad de Murcia e-mail: {dm.kulasekara,jlabellan}@um.es.
[2]Libelium Lab, e-mail: {juanfran.ingles}@gmail.com.
[3]Grupo de investigación UKEIM, Universidad Católica de Murcia, e-mail: bimbernon@ucam.edu.

ras a pocos días (frente a semanas en infraestructuras multi-GPU y multi-nodo de centros de datos), manteniendo al mismo tiempo un alto rendimiento en tareas de visión y lenguaje.

Sin embargo, SAFFE aún no incorpora la modalidad de audio, pese a que esta aporta señales semánticas y contextuales complementarias que pueden mejorar la precisión en entornos multimodales [26], [27], [6]. Extender SAFFE para integrar audio no es un problema trivial, ya que sus representaciones presentan alta redundancia temporal, una estructura dependiente del dominio de la frecuencia y sensibilidad tanto al ruido como a la variabilidad del muestreo [18], [2]. Además, su alineamiento con *embeddings* visuales y textuales exige preservar la estructura intrínseca de cada modalidad, al tiempo que se garantiza una interacción intermodal estable y semánticamente consistente.

Para abordar esta limitación, presentamos AudioSAFFE, una extensión de SAFFE con soporte para audio dentro del paradigma de composición de *encoders* congelados. AudioSAFFE reutiliza *encoders* preentrenados sin *fine-tuning* y entrena exclusivamente un FAM específico sobre datos emparejados audio–texto o audio–imagen. Este FAM se implementa como un módulo de *mid-fusion* basado en la técnica *multi-head cross attention*, diseñado para preservar las características intrínsecas de cada modalidad y, simultáneamente, habilitar una interacción intermodal controlada [28], [4], [29], [30], [25]. Al integrar selectivamente representaciones de alto nivel y restringir el flujo de atención cruzada, AudioSAFFE minimiza el número de parámetros entrenables y la carga computacional, permitiendo una composición multimodal eficiente en GPUs de escritorio.

Nuestras principales contribuciones son:

- Presentamos AudioSAFFE [1], una extensión de SAFFE que incorpora audio para habilitar tareas de recuperación (*retrieval*) de audio–texto y audio–imagen (a.k.a., tareas *retrieval*).
- Diseñamos un FAM extendido que amplía la fusión de *encoders* a la modalidad de audio. Es un módulo ligero de *mid-fusion* basado en atención cruzada que alinea representaciones de alto nivel a partir de datos emparejados limitados, reduciendo parámetros y coste computacional sin degradar la alineación intermodal. Permite entrenamiento eficiente en hardware de consumo (validado en una RTX 3060).
- Evaluamos AudioSAFFE en UrbanSound8K, ESC-50, AudioSet-500K, AudioSet-Balanced y Tri-Modal Balanced AudioSet en tareas de *retrieval* audio–texto e imagen–audio. Logra mAP *zero-shot* de 0,13 y 0,08 en AudioSet-500K y AudioSet-Balanced, respectivamente, y resultados competitivos frente a SOTA como ImageBind [3], con costes de entrenamiento significativamente menores (RTX 3060/4 días vs. 16×A100/2 semanas).

[1] `https://github.com/CAPS-UMU/SAFFE/tree/AudioSAFFE`

II. Antecedentes: SAFFE

AudioSAFFE se fundamenta en el mecanismo de fusión basado en *Transformers* introducido en la metodología SAFFE (*Semantic-Alignment Fusion of Frozen Encoders*) [25]. Siguiendo los principios estándar de la arquitectura *Transformer* [31] y una estrategia de *mid-fusion* inspirada en MBT [4], SAFFE se plantea como una alternativa computacionalmente eficiente frente a los enfoques SOTA basados en entrenamiento multimodal end-to-end a gran escala.

Más concretamente, SAFFE permite construir modelos multimodales específicos por tarea con un coste de entrenamiento reducido sobre hardware GPU de propósito general. En lugar de optimizar conjuntamente arquitecturas multimodales de gran tamaño —que requieren infraestructuras distribuidas y *datasets* emparejados masivos—, adopta una estrategia composicional basada en *Transformers* unimodales preentrenados (e.g., *encoders* de texto y visión), cuyos parámetros permanecen congelados. Este enfoque modular reduce drásticamente el número de parámetros entrenables y evita la optimización conjunta end-to-end entre modalidades.

El componente central es el módulo *FusionAlign* (FAM), una red ligera de *mid-fusion* [25], [4]. El FAM constituye el único componente entrenable y se optimiza sobre el *dataset* emparejado objetivo para alinear e integrar representaciones latentes generadas por *encoders* preentrenados de forma independiente (e.g., BERT para texto y Vision Transformers para imagen) [9], [13]. Al mantener congelados los *backbones* unimodales, el FAM permite una alineación multimodal eficiente con sobrecarga computacional acotada. Entrenado para tareas de *retrieval* de imágenes y comprensión del lenguaje en una única GPU NVIDIA RTX 3060 en aproximadamente 2,5 días [25], SAFFE alcanza rendimiento competitivo frente a modelos multimodales SOTA, evidenciando la eficacia de la composición de *encoders* congelados con mecanismos de alineación semántica de bajo costo computacional.

III. Trabajos relacionados

Las arquitecturas basadas en *Transformers* [31] constituyen el núcleo del aprendizaje multimodal moderno. Los *Transformers* unimodales, como BERT para texto [9], Vision Transformer (ViT) para imágenes [13] y Audio Spectrogram Transformer (AST) para audio [18], demuestran que el preentrenamiento a gran escala, tanto auto-supervisado como supervisado, permite aprender representaciones latentes de alta capacidad específicas de cada modalidad. Entrenados sobre *datasets* masivos, estos *encoders* capturan características semánticas y estructurales complejas, y actúan como bloques fundamentales en arquitecturas multimodales.

Los modelos visión–lenguaje, como CLIP [22], Chameleon [6], CN-CLIP [32] y los modelos derivados de LAION [33], aprenden espacios de *embeddings* compartidos mediante objetivos contrastivos o unificados, facilitando *retrieval* multimodal y trans-

ferencia *zero-shot*. De forma análoga, los sistemas audio–lenguaje, como Whisper [34] y variantes de CLAP [35], [27], alinean señales de audio con supervisión textual. Estos enfoques suelen requerir *datasets* emparejados a gran escala y optimización end-to-end de todos los componentes para lograr una alineación intermodal robusta.

Las demandas computacionales de estos regímenes de entrenamiento son elevadas. CLIP se entrenó con aproximadamente 4 mil millones de pares imagen–texto utilizando 256 GPU NVIDIA V100 durante dos semanas [22]. Chameleon requirió 1024 GPU y 856 481 horas de cómputo sobre 9,2 billones de tokens [6]. Whisper utilizó alrededor de 680 000 horas de datos audio–texto, mientras que el entrenamiento de AST a gran escala en AudioSet exige configuraciones multi-GPU con alta capacidad de memoria [18]. Estos ejemplos evidencian la magnitud de la infraestructura y los costes asociados a modelos multimodales SOTA.

Los marcos *Contrastive Language–Audio Pretraining* (CLAP) [20] extienden la alineación multimodal al dominio audio–texto. LAION-CLAP [27] se entrena con 630 000 pares audio–texto, combinando *encoders* de audio (e.g., PANN, HTSAT) con *encoders* de texto (e.g., RoBERTa, BERT). HTSAT+GPT2 [35] escala a 4,6 millones de pares y alcanza 0,102 mAP en AudioSet. Estos sistemas dependen de optimización end-to-end y supervisión emparejada intensiva, empleando estrategias de fusión tardía o intermedia para acoplar modalidades.

ImageBind [3] extiende la alineación a seis modalidades dentro de un espacio de *embedding* unificado anclado en representaciones visuales, entrenado con aproximadamente 2 millones de muestras emparejadas. Aunque efectivo para transferencia intermodal, este enfoque impone restricciones representacionales compartidas y mantiene una fuerte dependencia de datos a gran escala. AudioCLIP [26] amplía CLIP mediante la incorporación de un *encoder* de audio ESResNeXt entrenado con 1,7 millones de muestras de AudioSet, logrando buen rendimiento en clasificación y *zero-shot*, pero manteniendo un paradigma intensivo en datos y cómputo.

A diferencia de las propuestas anteriores, AudioSAFFE extiende la metodología SAFFE, que fusiona y alinea semánticamente *encoders* unimodales preentrenados y congelados, optimizando únicamente un módulo de fusión ligero. En lugar de reentrenar conjuntamente los *backbones*, AudioSAFFE adopta una estrategia de composición multimodal eficiente en parámetros, reduciendo significativamente la carga computacional mientras preserva un rendimiento competitivo en tareas de *retrieval* intermodal.

IV. AudioSAFFE

AudioSAFFE amplía SAFFE incorporando la modalidad de audio, preservando el principio de composición mediante *encoders* congelados. Generaliza esta metodología a la fusión audio–texto y audio–imagen sin requerir reentrenamiento end-to-end a gran escala. Al igual que en SAFFE, los *encoders* unimodales basados en *Transformers* permanecen congelados y generan representaciones latentes específicas de cada modalidad, mientras que únicamente la unidad FAM, adaptada a cada par modal (audio–imagen o audio–texto), se optimiza utilizando datos emparejados (véase la Figura 1). Este diseño minimiza los parámetros entrenables y evita la optimización conjunta de los *backbones* multimodales.

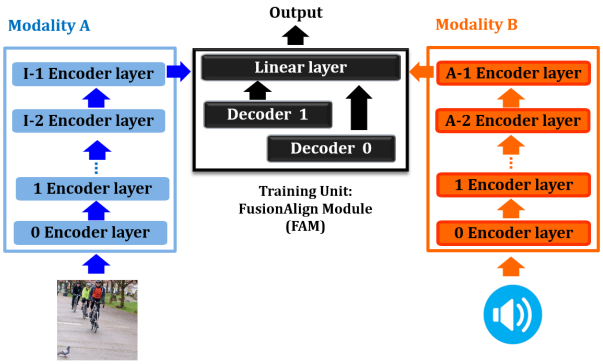


Fig. 1: **Arquitectura AudioSAFFE ilustrando la FAM junto a los *encoders* de imagen y audio congelados.**

A. FusionAlign Module (FAM)

El FAM es un módulo ligero de *mid-fusion* basado en atención cruzada multi-cabeza bidireccional (*bidirectional Multi-Head Cross-Attention*, MCA) [36]. Sean la modalidad A y la modalidad B dos *encoders* preentrenados (e.g., audio e imagen). Sus representaciones ocultas se denotan como:

$$h_i \in \mathbb{R}^{\mathbb{I}\times\mathbb{D}} \tag{1}$$

$$h_a \in \mathbb{R}^{\mathbb{A}\times\mathbb{D}} \tag{2}$$

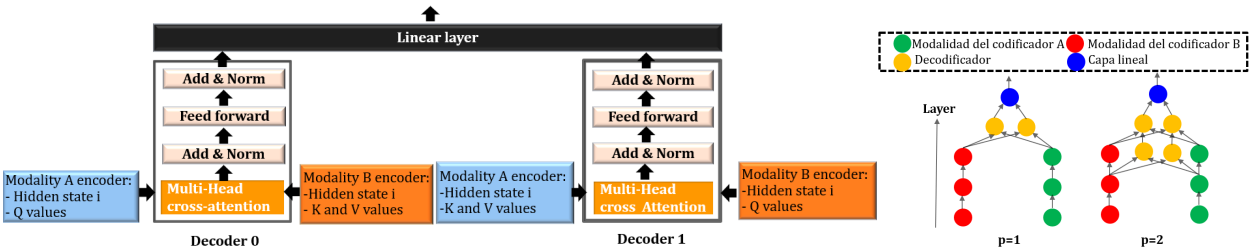
donde I y A representan el número de capas o tokens considerados por cada modalidad, y D la dimensionalidad del espacio latente. A diferencia de las estrategias de fusión temprana o tardía, la *mid-fusion* permite el intercambio de información a nivel de representaciones intermedias de los *encoders*; en particular, la variante *bottleneck* [4] condensa este intercambio en un subconjunto reducido de canales de comunicación [4]. El diseño *bottleneck* restringe explícitamente el número de capas participantes (p), limitando el flujo de atención intermodal y el coste computacional, sin degradar significativamente la capacidad de alineación. Para la evaluación, consideramos dos configuraciones (véase la Figura 2b):

- $p = 1$: fusión de una capa oculta (hlayer) por *encoder*.
- $p = 2$: fusión de dos capas ocultas consecutivas por *encoder*.

B. Decoder Architecture

Como se muestra en la Figura 2a, cada uno de los dos *decoders* realiza una atención cruzada bidireccional entre la modalidad A y la B:

$$\begin{aligned} D'_A &= decoder(h_{\text{a-1}}; h_{\text{i-1}}) \\ D'_B &= decoder(h_{\text{i-1}}; h_{\text{a-1}}) \end{aligned} \tag{3}$$



(a) (b)

Fig. 2: **Arquitectura detallada de FAM. (a)** Las capas de *encoder* seleccionadas proporcionan tensores K, Q y V específicos de la modalidad al *decoder* 0 y al *decoder* 1. **(b)** Arquitecturas FAM para $P = 1$ y $P = 2$.

El *decoder* implementa MCA:

$$\text{MCA}(A; B) = \text{Atención}\left(W^Q h_i,\, W^K h_a,\, W^V h_a\right) \tag{4}$$

MCA permite el intercambio bidireccional de información entre las modalidades A y B, mientras que el *decoder* incluye *Multi-Head Self-Attention* (MSA) residual [36] y una capa *Layer Normalization* (LN):

$$u' = MSA(LN(h_{\text{i-1}})) + h_{\text{i-1}} \tag{5}$$

$$Z' = MCA(A, B) + u' \tag{6}$$

$$D_A^p = LN(Z') \tag{7}$$

Estos elementos definen la arquitectura base del FAM. Para derivar la atención cruzada por pares y estimar una representación invariante a la modalidad, es necesario calcular la representación agregada $D = [D_A^1, \ldots, D_A^p, D_B^1, \ldots, D_B^p]$, a partir de las salidas generadas por los *decoders* (véase la capa lineal en la Figura 2a). Con el fin de reducir la complejidad, restringimos explícitamente el flujo de atención intermodal mediante un esquema de fusión *bottleneck mid-fusion* [25]. De forma más formal, dado un número de capas seleccionadas p, la representación invariante de modalidad se define como:

$$D = \frac{1}{2p}\left(\sum_{i=1}^{p} D_A^i + \sum_{i=1}^{p} D_B^i\right) \tag{8}$$

Aquí, p indexa las capas seleccionadas de cada modalidad. En este esquema, las modalidades A y B intercambian información a través del *decoder bottleneck*, mientras que los *encoders* unimodales congelados extraen representaciones a partir de las entradas, que se procesan de forma independiente y se alinean mediante mecanismos de atención (Ecuaciones 5, 6). Las representaciones cruzadas resultantes se agregan finalmente según la Ecuación 8.

Como se mostrará en la Sección V, tanto la configuración $p = 1$ como $p = 2$ mantienen un rendimiento robusto, lo que indica que es posible lograr una alineación multimodal efectiva sin necesidad de interacción densa entre capas.

V. Evaluación y resultados

En esta sección, evaluamos la eficiencia de los modelos multimodales derivados de AudioSAFFE a lo largo de tres ejes: (i) comparación con modelos multimodales SOTA de referencia; (ii) rendimiento en distintas combinaciones de modalidades (audio–texto y audio–imagen); y (iii) análisis de diferentes configuraciones de fusión en el módulo FAM. Todos los experimentos se ejecutan en una única GPU NVIDIA GeForce RTX 3060 de clase de escritorio (12,74 TFLOPS y 12 GB de memoria GDDR6).

A. Metodología

Conjuntos de datos de entrenamiento. AudioSAFFE sigue un diseño de fusión bimodal: el FAM se entrena exclusivamente con muestras emparejadas de dos modalidades, mientras que todos los *encoders* troncales permanecen congelados. Exploramos *encoders* unimodales preentrenados ampliamente utilizados en la literatura, como MIT/AST [18] para audio y Sentence-BERT [10] para texto (véase la Tabla I).

Como *datasets*, evaluamos AudioSet-500K y AudioSet-Balanced [37], UrbanSound8K [38], ESC-50 [39] y un subconjunto equilibrado de AudioSet, denominado «Tri-Modal Balanced AudioSet» (véase la Tabla II). Este último es el único que contiene tres modalidades alineadas y comprende 12.000 muestras de entrenamiento y 11.000 de prueba. Su tamaño reducido responde a la disponibilidad limitada de muestras válidas con correspondencia completa entre modalidades.

Para el procesamiento de audio, seguimos la frecuencia de muestreo requerida por el *encoder* de audio preentrenado y utilizamos clips completos en AudioSet-500K, UrbanSound8K y ESC-50 durante el entrenamiento y evaluación del FAM.

Procedimiento de entrenamiento. Entrenamos exclusivamente el FAM durante 40 épocas, manteniendo congelados los *encoders* unimodales. Utilizamos el optimizador Adam ponderado [40] con una tasa de aprendizaje de 2×10^{-4} [41] y un tamaño de lote de 2. Evaluamos AudioSAFFE en tareas de *retrieval* bidireccional habilitadas por su espacio de *embeddings* compartido: audio→texto, texto→audio, audio→imagen e imagen→audio. AudioSAFFE genera una representación conjunta tras la fusión de dos modalidades; en consecuencia, la *retrieval* se realiza mediante el ordenamiento por similitud coseno en dicho espacio latente.

Model	Fusion Modality	Text/Image frozen encoders	Audio	EL	ED
AudioSAFFE$_1$	**T+A**	sentence-transformers/all-mpnet-base-v2	AST	12	768
AudioSAFFE$_2$	**T+A**	openai/clip-vit-large-patch14	AST	12	768
AudioSAFFE$_3$	**I+A**	openai/clip-vit-large-patch14	AST	12	768

Tabla I: Configuraciones de AudioSAFFE. EL = número de capas del *encoder*; ED = dimensión de *embedding*; T = texto; I = imagen; A = audio; AST = MIT/ast-finetuned-audioset-14-14-0.443.

Conjunto de datos	#Clases	Entrenamiento	Prueba	Tamaño de imagen	Duración Audio
UrbanSound8K (US8K) [38]	10	7.853	879	–	4s
ESC-50 [39]	50	1.600	400	–	5s
AudioSet-500K [37]	527	500K	20K	–	10s
AudioSet-Balanced [37]	527	22K	20K	320×240–1280×720	10s
Tri-Modal Balanced AudioSet	527	12K	11K	320×240–1280×720	10s

Tabla II: Conjuntos de datos para el estudio de caso bimodal de AudioSAFFE.

Configuraciones *Retrieval*. Para la *retrieval* texto→audio, utilizamos los conjuntos de prueba completos: 400 muestras en ESC-50, 879 en UrbanSound8K y 20.000 en AudioSet-Balanced. Para las tareas audio→imagen e imagen→audio, empleamos Tri-Modal Balanced AudioSet y calculamos el mAP consultando 527 ejemplos seleccionados aleatoriamente (audio o imagen), que cubren todas las clases, sobre un conjunto de *retrieval* de 11.000 muestras.

Métricas de evaluación. El rendimiento en tareas de *retrieval* audio–texto se evalúa mediante precisión (rango [0–100]) en UrbanSound8K y ESC-50, mientras que en el resto de experimentos se usa la *mean Average Precision* (mAP, rango [0–1]).

B. AudioSAFFE frente a modelos SOTA congelados

Como se ha indicado en la Sección II, los modelos multimodales basados en *Transformers* para audio de referencia SOTA adoptan *zero-shot learning* para generalizar a nuevas tareas sin reentrenamiento específico, mitigando así el elevado coste computacional, los largos tiempos de entrenamiento y el gran número de parámetros asociados a la optimización *end-to-end*.

En este contexto, los enfoques existentes se diferencian principalmente por sus estrategias de fusión. VGGish + Word2Vec, HTSAT-22+GPT-2 y MATS-LLaMA emplean esquemas de fusión tardía, mientras que AudioCLIP, CLAP e ImageBind adoptan arquitecturas de *mid-fusion*. En este trabajo, el módulo FAM implementa un mecanismo de *bottleneck mid-fusion* sobre representaciones de alto nivel de *encoders* congelados. Para evaluar su eficacia, comparamos dos variantes de AudioSAFFE (AudioSAFFE$_1$ y AudioSAFFE$_2$) con configuración $p = 1$ frente a estos modelos SOTA en tareas de *retrieval* de audio sobre los conjuntos de datos mostrados en la Tabla II.

Como se muestra en la Tabla III, ambas variantes de AudioSAFFE alcanzan la mayor precisión en UrbanSound8K y ESC-50 utilizando datos emparejados limitados y conjuntos de entrenamiento reducidos (8K y 2K muestras), convergiendo en menos de 24 horas en una única GPU de clase de escritorio, mientras que los métodos comparados suelen requerir infraestructuras multi-nodo de nivel de servidor.

Para el dataset AudioSet, de mayor complejidad y con múltiples clases, se requiere un entrenamiento a mayor escala (500 000 muestras) y una optimización más prolongada del FAM (hasta 4 días); no obstante, los modelos alcanzan un mAP competitivo. En contraste, métodos como ImageBind dependen de millones de pares audio–texto (aprox. 2 millones) y de recursos computacionales sustanciales (e.g., 16 GPU de clase servidor durante aproximadamente dos semanas) para alcanzar su máximo rendimiento. En conjunto, AudioSAFFE logra resultados competitivos con un volumen significativamente menor de datos de entrenamiento y un coste computacional notablemente inferior al de las alternativas a gran escala.

En términos de complejidad de entrenamiento, AudioSAFFE presenta un número significativamente menor de parámetros entrenables (396 millones) que HTSAT-22 + GPT-2 (1530 millones) y MATS-LLaMA (7300 millones), manteniendo una precisión competitiva o superior.

Para evidenciar la flexibilidad de AudioSAFFE en la construcción de modelos bimodales audio–texto, la Tabla III presenta los resultados de dos configuraciones de *encoders*: AudioSAFFE$_1$ y AudioSAFFE$_2$. AudioSAFFE$_1$, que compone *encoders* SBERT y AST congelados (véase la Tabla I), alcanza el mejor rendimiento en UrbanSound8K y ESC-50, así como el segundo mejor mAP en AudioSet.

SBERT fue preentrenado con aproximadamente mil millones de pares de frases, mientras que AST se entrenó con alrededor de dos millones de muestras de audio, dando lugar a representaciones unimodales robustas antes de la fusión. Dado que AudioSAFFE mantiene estos *encoders* congelados y optimiza únicamente un módulo de fusión ligero, el rendimiento final depende directamente de la calidad y escala del preentrenamiento unimodal. En consecuencia, los resultados evidencian que el aprovechamiento de representaciones preentrenadas de alta calidad es un factor determinante para una composición multimodal eficaz dentro del marco de AudioSAFFE.

C. Estudio con múltiples modalidades

AudioSAFFE admite tanto la fusión audio–texto como audio–imagen. Evaluamos AudioSAFFE$_1$ y AudioSAFFE$_3$ utilizando las configuraciones de *encoders* preentrenados y congelados descritas en la Tabla I, en tareas de *retrieval* texto→audio, au-

Modelo	Estructura de texto	Estructura de audio	Parámetros Entrenamiento	Datos Entrenamiento	US8K	ESC-50	AudioSet	[Plataforma de cálculo]: Tiempo de entrenamiento
VGGish + Word2Vec [42]	Word2Vec	VGGish	36M	2,2M	N/A	26,00*	N/A	N/A
VGGish + Word2Vec [43]	Word2Vec	VGGish	116M	2,2M	N/A	33,00*	N/A	N/A
AudioCLIP [26]	CLIP	ESResNeXt	180M	1,7M	68,78*	69,40*	N/A	[4×NVIDIA V100]: 5 días
CLAP [20]	BERT	CNN14	190M	128K	73,24*	82,60*	0,06	[16GB V100 GPU]: 2 días
ImageBind [3]	CLIP	ViT-B *encoder* de audio	86M	2M	N/A	66,90*	**0,17**	[16×A100GPUs]: 2 semanas
HTSAT-22 + GPT-2 [35]	GPT-2	HTSAT	1,53B	4,6M	75,00*	88,20*	0,10	[8×A100]: 5 días
MATS-LLaMA [44]	LLaMA-7B	CLAP	7,3B	5M	67,00*	88,00*	N/A	[2×A100 GPUs]: 25 horas
AudioSAFFE$_1$	SBERT	AST	396M	8K / 2K / 500K	**78,97***	**91,00***	0,13	[RTX-3060 GPU]: 24 horas / 1,5 horas / 4 días
AudioSAFFE$_2$	CLIP	AST	396M	8K / 2K / 500K	78,25*	88,50*	0,11	[RTX-3060 GPU]: 24 horas / 1,5 horas / 4 días

Tabla III: Resultados de tareas *retrieval* de audio a texto en US8K, ESC-50 y AudioSet-500K para AudioSAFFE ($p = 1$), frente a los modelos multimodales SOTA. Los resultados se presentan como clasificación *zero-shot* utilizando mAP o precisión (∗); la negrita indica el mejor rendimiento.

Conjunto de datos	Modalidad		mAP o Acc.	
	Consulta	Resultado	AudioSAFFE	AudioCLIP
AudioSet	Texto	Audio	0,21	**0,30**
	Audio	Texto	0,13	**0,34**
	Audio	Imagen	**0,22**	0,07
	Imagen	Audio	**0,25**	0,07
US8K	Texto	Audio	**0,919**	0,80
	Audio	Texto	78,97*	**90,07***
ESC-50	Texto	Audio	**0,92**	0,78
	Audio	Texto	91*	**97,15***

Tabla IV: Evaluación de AudioSAFFE sin disparo (p = 1) frente a AudioCLIP con disparo para mAP o precisión(∗).

Conjunto de datos	Modalidad		mAP o Acc. / #Épocas	
	Consulta	Resultado	$p = 1$	$p = 2$
AudioSet	Texto	Audio	0,212/3	**0,215/3**
	Audio	Texto	0,074/3	**0,08/3**
US8K	Texto	Audio	0,917/39	**0,919/20**
	Audio	Texto	78,97/40*	**78,98/15***
ESC-50	Texto	Audio	0,91/12	**0,91/7**
	Audio	Texto	92,25/13*	**92,75/19***

Tabla V: Estudio de AudioSAFFE con $p = 1$ y $p = 2$ utilizando mAP/precisión (*). Épocas en las que se alcanzó la máxima precisión.

dio→texto, imagen→audio y audio→imagen. Los valores de mAP correspondientes se recogen en la Tabla IV. Los experimentos se realizan bajo la configuración $p = 1$, lo que permite un análisis comparativo directo entre AudioSAFFE y AudioCLIP en distintas modalidades y tareas posteriores.

Las tareas de *retrieval* texto→audio se evalúan en AudioSet-500K tras el entrenamiento, mientras que las tareas audio→imagen e imagen→audio se evalúan sobre Tri-Modal Balanced AudioSet. En todos los conjuntos de datos y tareas consideradas, AudioSAFFE alcanza valores de mAP y precisión consistentemente elevados. En algunos escenarios, AudioCLIP supera a nuestro modelo, probablemente debido a su entrenamiento con aproximadamente 1,7 millones de pares en el dominio objetivo, lo que favorece una alineación más específica por tarea frente a un enfoque de generalización sin *fine-tuning*.

No obstante, el mAP de AudioCLIP tras ajuste fino para fusión audio–imagen resulta inferior al obtenido por AudioSAFFE sin entrenamiento adicional en varios casos. En particular, AudioCLIP muestra ventaja principalmente en tareas audio→texto, mientras que AudioSAFFE obtiene un rendimiento notablemente superior en texto→audio en ESC-50 y UrbanSound8K. En conjunto, estos resultados indican que AudioSAFFE ofrece un rendimiento competitivo y consistente en múltiples modalidades para tareas de *retrieval* bidireccional.

D. Estrategias de fusión en FAM

El módulo FAM agrega p estados ocultos de cada *encoder* congelado, donde $p \in \{1, \ldots, L\}$ y L denota el número total de capas del *encoder*. En nuestros experimentos, evaluamos $p = 1$ y $p = 2$, que corresponden a la fusión de una y dos capas, respectivamente (véase la Figura 2b). El *encoder* backbone considerado consta de 12 capas de *Transformer*, de las cuales seleccionamos subconjuntos específicos para analizar el impacto de la selección de capas en el rendimiento de *retrieval*. El objetivo es cuantificar la variación en precisión y mAP al fusionar representaciones jerárquicas de distinto nivel, manteniendo todos los *encoders* congelados. La variante AudioSAFFE_free utiliza directamente las salidas de los *encoders* congelados para calcular similitud coseno entre modalidades, sin entrenamiento del FAM.

Para AudioSAFFE$_1$ (Tabla I), evaluamos siete configuraciones de capas: capas 2, 5, 9, 10 y 11 para $p = 1$, y combinaciones 9+10 y 10+11 para $p = 2$. El FAM implementa atención cruzada multi-cabeza (MCA), donde las salidas seleccionadas del *encoder* se proyectan a los espacios Q, K y V (Figura 2a). Los resultados de *retrieval* audio–texto se presentan en la Figura 3.

Para $p = 1$, la capa 11 obtiene consistentemente el mejor rendimiento en todos los conjuntos de datos, tanto en texto→audio como en audio→texto. Para $p = 2$, la combinación de capas 10 y 11 proporciona el mejor rendimiento global. Además, $p = 2$ converge más rápidamente y alcanza mayor precisión en menos épocas. Como se muestra en la Tabla V, la configuración $p = 2$ supera sistemáticamente a $p = 1$ en todos los conjuntos de datos con menor tiempo de entrenamiento. Por ejemplo, en audio→texto, $p = 1$ alcanza una precisión de 0,0743 en la época 3, mientras que $p = 2$ alcanza 0,08 en la misma época. Estos resultados evidencian la eficacia del marco AudioSAFFE para optimizar la integración multimodal, destacando el papel del FAM como mecanismo eficiente de alineación intermodal.

VI. Conclusión y trabajo futuro

En este trabajo hemos propuesto AudioSAFFE, una extensión de la metodología SAFFE que integra la modalidad de audio dentro de un marco multimodal basado en *encoders* congelados. El método reutiliza *Transformers* unimodales preentrenados sin *fine-tuning* y optimiza exclusivamente un FAM ligero basado en atención cruzada bajo el paradigma *bottleneck mid-fusion*, evitando el entrenamiento end-to-end del modelo multimodal. Este diseño preserva representaciones latentes específicas de cada modalidad, al tiempo que permite una alineación intermo-

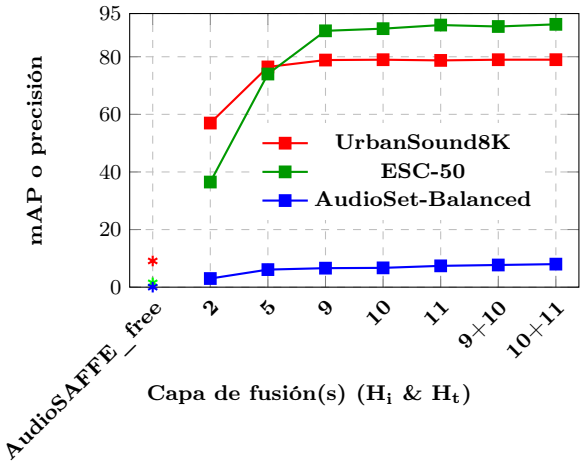


Fig. 3: Estudio de los métodos de fusión AudioSAFFE con *retrieval* de audio a texto. La unidad FAM empleó tanto la integración del *encoder* $p = 1$ (por ejemplo, 11 en el eje x) como $p = 2$ (por ejemplo, $10 + 11$ en el eje x). *AudioSAFFE_free* = FAM no está entrenado.

dal eficiente y controlada. En tareas de *retrieval* audio–texto y audio–imagen, AudioSAFFE alcanza un rendimiento competitivo con un número significativamente menor de parámetros entrenables y un coste computacional reducido, evidenciando que la composición multimodal efectiva no requiere infraestructuras multi-GPU a gran escala ni *datasets* emparejados masivos.

Como trabajo futuro, se plantea generalizar AudioSAFFE a configuraciones multimodales de mayor orden que permitan la integración de modalidades adicionales. Asimismo, se explorarán estrategias de optimización ortogonales, incluyendo técnicas de adaptación eficientes en parámetros como LoRA, con el objetivo de mejorar la capacidad de los *encoders* congelados y el rendimiento específico por tarea, manteniendo al mismo tiempo la eficiencia computacional.

Agradecimientos

Trabajo realizado en el contexto del proyecto PID2022-136315OB-I00 financiado por MICIU/AEI/10.13039/501100011033/ y "FEDER Una manera de hacer Europa", UE. También por MICIU/AEI/10.13039/501100011033 y la "Unión Europea NextGenerationEU/PRTR", bajo la subvención RYC2021-031966-I.

Referencias

[1] Peng Wang, Shijie Wang, Junyang Lin, Shuai Bai, Xiaohuan Zhou, Jingren Zhou, Xinggang Wang, and Chang Zhou, "One-peace: Exploring one general representation model toward unlimited modalities," 2023.

[2] Songtao Li and Hao Tang, "Multimodal alignment and fusion: A survey," 2025.

[3] Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra, "Imagebind: One embedding space to bind them all," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 15180–15190.

[4] Arsha Nagrani, Shan Yang, Anurag Arnab, Aren Jansen, Cordelia Schmid, and Chen Sun, "Attention bottlenecks for multimodal fusion," *Advances in neural information processing systems*, vol. 34, pp. 14200–14213, 2021.

[5] Rohit Girdhar, Mannat Singh, Nikhila Ravi, Laurens Van Der Maaten, Armand Joulin, and Ishan Misra, "Omnivore: A single model for many visual modalities," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 16102–16112.

[6] Louis-Philippe Morency and Tadas Baltrušaitis, "Multimodal machine learning: integrating language, vision and speech," in *Proceedings of the 55th annual meeting of the association for computational linguistics: Tutorial abstracts*, 2017, pp. 3–5.

[7] Hassan Akbari, Liangzhe Yuan, Rui Qian, Wei-Hong Chuang, Shih-Fu Chang, Yin Cui, and Boqing Gong, "Vatt: Transformers for multimodal self-supervised learning from raw video, audio and text," *Advances in Neural Information Processing Systems*, vol. 34, pp. 24206–24221, 2021.

[8] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid, "Vivit: A video vision transformer," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 6836–6846.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[10] N Reimers, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[11] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," 2020.

[12] Yinhan Liu, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, vol. 364, 2019.

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021.

[14] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick, "Masked autoencoders are scalable vision learners," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 16000–16009.

[15] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick, "Momentum contrast for unsupervised visual representation learning," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 9726–9735.

[16] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton, "A simple framework for contrastive learning of visual representations," *CoRR*, vol. abs/2002.05709, 2020.

[17] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin, "Emerging properties in self-supervised vision transformers," 2021.

[18] Yuan Gong, Yu-An Chung, and James Glass, "AST: Audio Spectrogram Transformer," in *Proc. Interspeech 2021*, 2021, pp. 571–575.

[19] Ho-Hsiang Wu, Prem Seetharaman, Kundan Kumar, and Juan Pablo Bello, "Wav2clip: Learning robust audio representations from clip," 2022.

[20] Benjamin Elizalde, Soham Deshmukh, Mahmoud Al Ismail, and Huaming Wang, "Clap learning audio concepts from natural language supervision," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023, pp. 1–5.

[21] Junyi Ao, Rui Wang, Long Zhou, Chengyi Wang, Shuo Ren, Yu Wu, Shujie Liu, Tom Ko, Qing Li, Yu Zhang, Zhihua Wei, Yao Qian, Jinyu Li, and Furu Wei, "Speecht5: Unified-modal encoder-decoder pre-training for spoken language processing," 2022.

[22] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al., "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[23] Jack Merullo, Louis Castricato, Carsten Eickhoff, and Ellie Pavlick, "Linearly mapping from image to text spa-

ce," in *The Eleventh International Conference on Learning Representations*, 2023.

[24] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al., "Flamingo: a visual language model for few-shot learning," *Advances in neural information processing systems*, vol. 35, pp. 23716–23736, 2022.

[25] Maithri Kulasekara, Juan F Inglés-Romero, Baldomero Imbernón, and José L Abellán, "Saffe: Multimodal model composition with semantic-alignment fusion of frozen encoders: M. kulasekara et al.," *The Journal of Supercomputing*, vol. 81, no. 10, pp. 1114, 2025.

[26] Andrey Guzhov, Federico Raue, Jörn Hees, and Andreas Dengel, "Audioclip: Extending clip to image, text and audio," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 976–980.

[27] Yusong Wu, Ke Chen, Tianyu Zhang, Yuchen Hui, Taylor Berg-Kirkpatrick, and Shlomo Dubnov, "Large-scale contrastive language-audio pretraining with feature fusion and keyword-to-caption augmentation," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023, pp. 1–5.

[28] Valerio Guarrasi, Fatih Aksu, Camillo Maria Caruso, Francesco Di Feola, Aurora Rofena, Filippo Ruffini, and Paolo Soda, "A systematic review of intermediate fusion in multimodal deep learning for biomedical applications," *Image and Vision Computing*, p. 105509, 2025.

[29] Weiyao Wang, Du Tran, and Matt Feiszli, "What makes training multi-modal classification networks hard?," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 12695–12705.

[30] Fanyi Xiao, Yong Jae Lee, Kristen Grauman, Jitendra Malik, and Christoph Feichtenhofer, "Audiovisual slowfast networks for video recognition," 2020.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. 2017, vol. 30, Curran Associates, Inc.

[32] An Yang, Junshu Pan, Junyang Lin, Rui Men, Yichang Zhang, Jingren Zhou, and Chang Zhou, "Chinese clip: Contrastive vision-language pretraining in chinese," *arXiv preprint arXiv:2211.01335*, 2022.

[33] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al., "Laion-5b: An open large-scale dataset for training next generation image-text models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 25278–25294, 2022.

[34] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever, "Robust speech recognition via large-scale weak supervision," 2022.

[35] Benjamin Elizalde, Soham Deshmukh, and Huaming Wang, "Natural language supervision for general-purpose audio representations," 2024.

[36] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah, "Transformers in vision: A survey," *ACM computing surveys (CSUR)*, vol. 54, no. 10s, pp. 1–41, 2022.

[37] Jort F Gemmeke, Daniel PW Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, Mano Moore, Manoj Plakal, Marvin Ritter, et al., "Audio set: An ontology and human-labeled dataset for audio events," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 776–780.

[38] J. Salamon, C. Jacoby, and J. P. Bello, "A dataset and taxonomy for urban sound research," in *22nd ACM International Conference on Multimedia (ACM-MM'14)*, Orlando, FL, USA, Nov. 2014, pp. 1041–1044.

[39] Karol J. Piczak, "ESC: Dataset for Environmental Sound Classification," in *Proceedings of the 23rd Annual ACM Conference on Multimedia*. 2015, pp. 1015–1018, ACM Press.

[40] Diederik P Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[41] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi, "Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation," in *International conference on machine learning*. PMLR, 2022, pp. 12888–12900.

[42] Huang Xie and Tuomas Virtanen, "Zero-shot audio classification based on class label embeddings," in *2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2019, pp. 264–267.

[43] Huang Xie and Tuomas Virtanen, "Zero-shot audio classification via semantic embeddings," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 29, pp. 1233–1242, 2021.

[44] Wen Wang, Ruibing Hou, Hong Chang, Shiguang Shan, and Xilin Chen, "Mats: An audio language model under text-only supervision," *arXiv preprint arXiv:2502.13433*, 2025.

Arquitecturas del subsistema de memoria y almacenamiento secundario

Mecanismo avanzado de gestión de metadatos para sistemas de almacenamiento de gran escala

Genaro Sánchez-Gallegos[1], Javier García-Blas[1] y Jesús Carretero[1]

Resumen— **La creciente demanda de procesamiento de datos de nuevas aplicaciones intensivas en datos está ejerciendo una gran presión sobre el rendimiento y la capacidad de los sistemas de almacenamiento en la Computación de Alto Rendimiento. Para hacer frente a estas demandas se han introducido avances en las tecnologías de almacenamiento, como NVMe y la memoria persistente. Sin embargo, depender exclusivamente de la velocidad de los dispositivos de almacenamiento no es rentable, por lo que resultan necesarias jerarquías de almacenamiento en múltiples niveles para gestionar los datos según su uso. En la literatura se han propuesto sistemas de archivos *ad-hoc* como solución, los cuales utilizan los recursos de almacenamiento disponibles en los nodos de cómputo, incluidos la memoria y el almacenamiento persistente, para crear sistemas de archivos temporales que se adaptan al comportamiento de las aplicaciones en el entorno HPC. Por otro lado, las aplicaciones modernas de HPC, Inteligencia Artificial y analítica de datos generan una presión sobre los sistemas de almacenamiento tradicionales debido al manejo masivo de millones de archivos pequeños. Este fenómeno crea un cuello de botella en la gestión de los metadatos, donde las operaciones de gestión (como crear, abrir o borrar archivos) representan hasta el 80 % de la carga de trabajo, limitando la escalabilidad del sistema.**

En este trabajo se presenta el diseño, la implementación y la evaluación de un nuevo modelo de gestión de metadatos para HERCULES, un sistema de archivos ad-hoc distribuido en memoria. La mayor contribución radica en la evolución técnica a lo largo de cuatro versiones del gestor de metadatos, considerando sus ventajas y limitaciones. Como última técnica implementada se presentan los Mapas Jerárquicos (HHMaps). Esta técnica sustituye las estructuras de árbol tradicionales mediante la vinculación de mapas, lo que permite acceder directamente a los datos a través de rutas absolutas y elimina la necesidad de navegar por jerarquías complejas. Los resultados de la evaluación realizada demuestran un mejor rendimiento que BeeGFS y valores competitivos respecto a Lustre, los cuales son dos sistemas de archivos paralelos ampliamente utilizados en sistemas de Computación de Alto Rendimiento.

Palabras clave— **Sistemas de ficheros, Metadatos, Paralelismo, Computación de Alto Rendimiento**

I. Introducción

En la última década, el panorama de la Computación de Alto Rendimiento (HPC) ha experimentado una gran transformación, pasando de simulaciones puramente numéricas a un ecosistema híbrido en el que convergen las aplicaciones científicas tradicionales y las cargas de trabajo emergentes de Inteligencia Artificial (IA). A medida que la escala de estos sistemas se aproxima a la era de la exaescala, el cuello de botella del rendimiento se ha desplazado significativamente del cómputo puro al subsistema de entrada/salida (E/S).

Las aplicaciones científicas modernas, que abarcan desde la genómica hasta la astrofísica, generan volúmenes masivos de datos distribuidos en millones de archivos en el orden de los bytes y los kilobytes [1]. Los sistemas de archivos paralelos (PFS) tradicionales, como Lustre o GPFS, fueron optimizados para transferencias de datos de gran ancho de banda y archivos de gran tamaño (en el orden de los gigabytes). Sin embargo, la naturaleza atómica y masiva de los metadatos en las simulaciones actuales somete a los servidores de metadatos a una gran presión, lo que genera latencias que degradan la eficiencia global del sistema.

Tradicionalmente, los entornos de HPC han dependido de sistemas de archivos paralelos centralizados [2], pero el cambio hacia infraestructuras distribuidas y nubes públicas ha obligado a replantear la gestión de datos, con enfoques que a menudo se inspiran en los sistemas de Big Data [3]. Sin embargo, la mayoría de las plataformas de análisis de datos no están diseñadas para manejar operaciones de E/S pequeñas y aleatorias típicas de las cargas de trabajo científicas.

Paralelamente, el auge del aprendizaje profundo ha introducido patrones de acceso a datos drásticamente distintos. Tanto en las fases de entrenamiento como en las de inferencia, los modelos requieren el acceso aleatorio y repetitivo a vastos conjuntos de datos (*datasets*) compuestos por imágenes, fragmentos de audio o tensores [4]. Como demostraron Chowdhury *et al.* [5], las aplicaciones de *Machine Learning* (*ML*) están dominadas principalmente por una gran cantidad de archivos pequeños y por operaciones de lectura y búsqueda bajo POSIX. En la etapa de entrenamiento surge la necesidad de realizar *shuffling* de datos y lecturas aleatorias que saturan las operaciones de metadatos, ya que el sistema debe localizar y abrir miles de archivos por segundo para alimentar a las GPUs. Por otro lado, en la etapa de inferencia la baja latencia es crítica y cualquier retraso en la resolución de metadatos se traduce directamente en una pérdida de rendimiento en tiempo real.

La disparidad entre la capacidad de procesamiento de las arquitecturas aceleradas y la agilidad de los sistemas de almacenamiento actuales evidencia una necesidad crítica: la evolución de la gestión de metadatos. Las soluciones actuales de almacenamiento no escalan de forma lineal con el número de archivos, creando un "muro de metadatos" que limita el progreso científico y la escalabilidad de los modelos

[1]Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: {fjblas,jcarrete}@inf.uc3m.es, {gesanche,cpetre}@pa.uc3m.es

de IA más ambiciosos.

En este contexto, se observa que uno de los principales desafíos es el manejo de metadatos. Los metadatos, que incluyen información sobre archivos, directorios, permisos y sus ubicaciones, pueden representar un gran porcentaje de las operaciones totales del sistema de archivos, alcanzando en ocasiones hasta el 80 % [6]. Muchos sistemas de ficheros usan estructuras de datos en memoria para gestionar los metadatos, como GTree/GNode. Aunque estas son rápidas en escenarios a pequeña escala, presentan varias limitaciones: carecen de persistencia, por lo que los metadatos se pierden en caso de caída del sistema; escalan con dificultad debido a las restricciones de memoria; y requieren un bloqueo (*locking*) complejo para gestionar la concurrencia, lo que aumenta la latencia y reduce el rendimiento [7].

El sistema de ficheros ad-hoc HERCULES [8] se creó para mejorar el rendimiento de E/S mediante el despliegue de sistemas de archivos temporales y específicos para aplicaciones que pueden ejecutarse sobre almacenamiento local. Aunque se han demostrado resultados prometedores, el diseño original dependía de estructuras GNode en memoria para los metadatos y utilizaba ZeroMQ para la comunicación, la cual carece de soporte para redes de alto rendimiento y de una interfaz compatible con POSIX [9]. Este artículo presenta una propuesta técnica para mejorar la eficiencia en el acceso a metadatos, centrada en la eliminación de cuellos de botella de E/S en entornos de exaescala.

El resto del trabajo se organiza de la siguiente manera. En la sección II se resumen trabajos relacionados con este trabajo, destacando las diferencias y contribuciones técnicas. La sección III introduce las características técnicas del sistema de ficheros para datos efímeros HERCULES, necesarias para contextualizar el trabajo. Los mecanismos de gestión de metadatos presentados en este trabajo se detallan en la sección IV. En la sección V se presentan los resultados experimentales de la evaluación realizada. Finalmente, en la sección VI se presentan las conclusiones y los trabajos futuros que se llevarán a cabo en esta línea de investigación.

II. Trabajos relacionados

Una de las técnicas más empleadas para reducir el impacto del procesamiento de metadatos es el particionamiento (cómo dividir los datos) y el desacoplamiento (separar el qué del dónde). En el sistema Ceph [10] se elimina la necesidad de una tabla de metadatos centralizada. En lugar de que un servidor le indique al cliente dónde está un archivo, Ceph utiliza el algoritmo CRUSH (Controlled Replication Under Scalable Hashing). Mediante CRUSH, el cliente calcula la ubicación de los datos mediante una función matemática, utilizando un mapa que describe la topología de la red. Con este método se elimina el cuello de botella del servidor de metadatos central, permitiendo que miles de clientes contacten directamente con los nodos de almacenamiento en paralelo.

Otra aproximación para particionar la gestión de metadatos es la empleada por HDFS. HDFS proporciona escalabilidad mediante federación [11]. Originalmente, HDFS dependía de un único *NameNode* (servidor de metadatos), lo que limitaba el sistema a unos 400 millones de archivos. El sistema federado permite tener múltiples *NameNodes* independientes, cada uno gestionando una parte del espacio de nombres. Esta solución permite escalar horizontalmente el servicio de metadatos, añadiendo más servidores de nombres a medida que aumenta el volumen de archivos pequeños. Sin embargo, esta propuesta no es adecuada cuando el número de metadatos crece, ya que no resulta ser operativamente eficiente. Por ello, FileScale [12] se presenta como una mejora a HDFS añadiendo una arquitectura distribuida de tres capas que consisten en: 1) un servidor proxy, 2) una capa de enrutamiento y caché distribuida en múltiples servidores *NameNode* y 3) un sistema de gestión de base de datos distribuido (*DDBMS*) *shared-nothing* el cual gestiona los metadatos de forma persistente. Para ello, en FileScale han diseñado su propio diagrama relacional en su base de datos almacenando la ruta completa como clave primaria, lo que permite realizar operaciones de búsqueda de forma paralela, además de ahorrar espacio en disco y realizar operaciones *rename* de forma rápida al solo tener que modificar la raíz de las ramas modificadas.

Por otro lado, GIGA+ [13] aborda el problema de la saturación en los servidores de metadatos cuando los directorios crecen hasta contener millones de archivos. Los autores proponen una técnica de particionamiento distribuido basada en un *hashing* indexado que permite que el espacio de nombres crezca de forma incremental y asíncrona entre múltiples servidores. A diferencia de los métodos tradicionales, GIGA+ evita la necesidad de una serialización global o de detener el sistema para reequilibrar los datos, logrando un rendimiento que escala de forma lineal con el número de servidores añadidos, lo cual es vital para sistemas de almacenamiento a exaescala.

Otra de las técnicas más empleadas es la utilización de memorias no volátiles (NVM) para acelerar la gestión de metadatos. NOVA [14] combina la baja latencia de la memoria RAM con la persistencia del almacenamiento tradicional. El sistema utiliza una arquitectura de "estructuras de registro" (log-structured) por cada *inodo*, lo que elimina la contención entre hilos y permite realizar operaciones de metadatos extremadamente rápidas y concurrentes.

Una tendencia creciente es el despliegue de capas de almacenamiento transitorias y de alto rendimiento. En este contexto, [15] presenta un nuevo sistema de archivos paralelo Ad-Hoc basado en el sistema *Expand*. Esta propuesta prioriza la flexibilidad en entornos HPC, permitiendo la creación de instancias de almacenamiento bajo demanda que aprovechan los recursos locales para mitigar los cuellos de botella de los sistemas de archivos paralelos tradicionales.

Por otro lado, la escalabilidad de los servicios de metadatos sigue siendo un desafío crítico. Mientras

que algunos enfoques optimizan la centralización, como *SingularFS* [16], que demuestra la viabilidad de gestionar un sistema a escala de miles de millones de archivos mediante un único servidor de metadatos altamente optimizado, otros trabajos apuestan por la distribución del servicio.

En la línea de la descentralización, *CFS* [17] mejora la escalabilidad reduciendo el alcance de las secciones críticas en el servicio de metadatos, lo que incrementa la concurrencia y reduce la sobrecarga por sincronización. Complementariamente, *InfiniFS* [18] propone un servicio de metadatos eficiente diseñado específicamente para entornos de gran escala, utilizando una arquitectura desacoplada para ofrecer un alto rendimiento y baja latencia en cargas de trabajo intensivas en metadatos. En conjunto, estos avances representan un cambio hacia soluciones de almacenamiento más granulares y especializadas para las demandas computacionales modernas.

III. HERCULES

HERCULES es un sistema de archivos *ad-hoc* en memoria diseñado para mejorar el rendimiento de aplicaciones intensivas en datos en entornos de computación de alto rendimiento (HPC). Utiliza la memoria principal de los nodos de cómputo para crear un sistema de almacenamiento temporal que se adapta dinámicamente al comportamiento de las aplicaciones, lo que facilita un acceso más rápido a los datos y reduce la carga en los sistemas de almacenamiento tradicionales.

La Fig. 1 presenta la arquitectura de HERCULES, diseñada bajo el modelo cliente-servidor. HERCULES es un sistema de archivos *ad-hoc* que se despliega junto con cada aplicación, gestionando tanto los metadatos como los datos. Se integra en cada aplicación como una capa *front-end* adicional configurable de forma independiente para adaptarse a los requisitos específicos de E/S de datos.

En este trabajo mostramos cómo HERCULES ofrece flexibilidad en su despliegue mediante el uso de múltiples servidores distribuidos, cada uno ejecutando un gestor de tareas y un conjunto de subprocesos. El gestor de tareas asigna la carga de trabajo entrante entre los subprocesos para equilibrar el procesamiento en un entorno multiproceso. La arquitectura de HERCULES se compone principalmente de procesos cliente (*front-end*), nodos de metadatos (de 1 a N) y nodos de datos (de 1 a M), que conforman el *back-end* del sistema.

En cuanto a la interacción entre estos componentes, el *front-end* de HERCULES se comunica únicamente con los servidores de metadatos para ejecutar operaciones de esta categoría, como *create* y *open*. Por otro lado, las operaciones relacionadas con los datos se manejan directamente en un servidor de datos determinado en el *front-end*, eliminando la necesidad de un mapeo global de datos en todos los niveles.

En cualquier despliegue *ad-hoc* de HERCULES, al crear un conjunto de datos (*dataset*), sus metadatos registran la lista de servidores que almacenan la información. Esta lista se utiliza para determinar la distribución de los bloques del *dataset* entre los nodos de almacenamiento según la política de mapeo seleccionada. De este modo, es posible identificar qué servidores de HERCULES participan en las operaciones de E/S de cada *dataset*. Las políticas de mapeo o distribución de datos se establecen durante el despliegue de cada sistema *ad-hoc*. Como optimización, además de registrar los servidores disponibles, HERCULES almacena el número total de servidores que contienen el *dataset*, reduciendo así la necesidad de realizar operaciones adicionales sobre la lista de nodos.

IV. Gestión de Metadatos

La gestión de metadatos es fundamental para el rendimiento de los sistemas de archivos. La mayoría de los PFS tradicionales suelen centrar sus esfuerzos en la gestión de datos, bajo la premisa de que la carga de trabajo de E/S de las aplicaciones consistía principalmente en unos pocos archivos de gran tamaño leídos o escritos periódicamente. Sin embargo, las nuevas cargas de trabajo de E/S provenientes de aplicaciones de IA, streaming o analítica de datos presentan características distintas, ya que en algunos casos pueden utilizar millones de archivos pequeños. Teniendo en cuenta todas estas nuevas cargas de trabajo, en HERCULES hemos dedicado un gran esfuerzo a optimizar la gestión de metadatos. En este trabajo, presentamos la evolución de las versiones del gestor de metadatos de HERCULES, detallando las mejoras diseñadas e implementadas para la optimización del rendimiento.

A. *HV1: Almacenamiento en memoria basado en estructuras de árbol N-ary mediante GNode*

HV1 es la línea base inicial de la gestión de metadatos proporcionada en HERCULES. En esta versión, todas las llamadas de metadatos se remiten a los servidores de almacenamiento de metadatos, que gestionan dichas llamadas mediante estructuras de datos GNode formando un árbol N-ary. En este diseño, primero se parte de un nodo raíz (*root*, p. ej., *imss://*). Para cada nodo en un nivel de profundidad más baja se crea un nuevo elemento GNode que almacena la ruta absoluta al padre (p. ej., *imss://parent-dir/* o *imss://parent-dir/FileX*).

Un ejemplo del proceso de búsqueda se representa en la Fig. 2. Este caso representa la búsqueda del fichero *imss://parent-dir/sub-parent-dir-Q/FileY*. Para ello, es necesario realizar la comparación de la ruta del nodo actual (comenzando por la raíz) con la del fichero deseado. Si la ruta del nodo actual está contenida dentro de la ruta deseada pero no son iguales, se realiza una comparación con la ruta de todos los hijos (en este ejemplo es *parent-dir/*). Puesto que *imss://parent-dir/* forma parte de la ruta deseada se continúa el recorrido en todos sus hijos. Como *sub-parent-dir-P* no se encuentra dentro de la ruta deseada, se continúa con *sub-parent-dir-Q* que sí for-

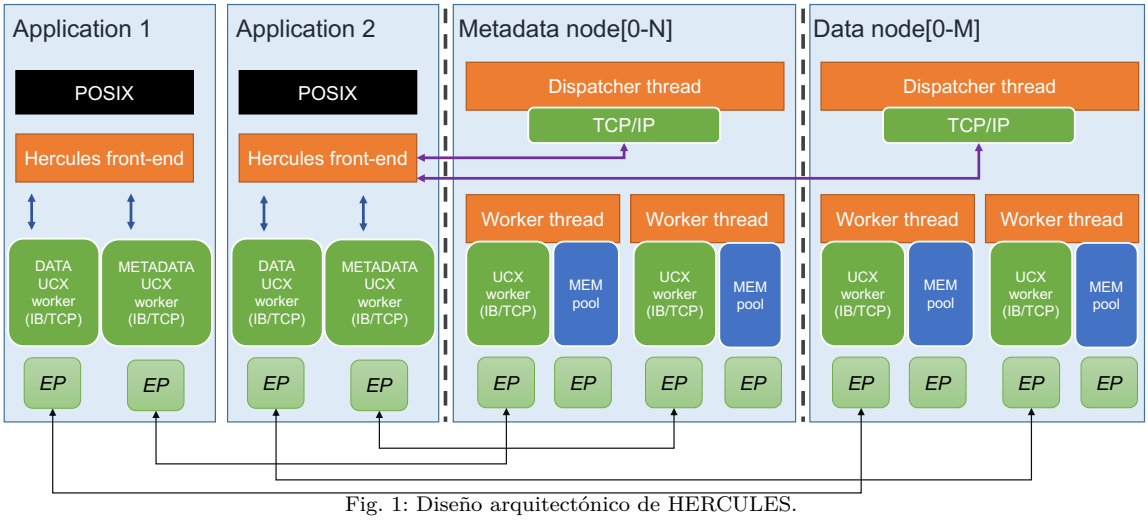


Fig. 1: Diseño arquitectónico de HERCULES.

ma parte, y al aplicar el mismo proceso de forma recursiva es donde finalmente se encuentra el fichero deseado.

Como se observa en el ejemplo gráfico, este proceso de búsqueda necesita 6 etapas para realizarse, lo que aumenta a medida que el árbol se hace más grande.

En esta versión, se aplican optimizaciones de tipo "caché", en el que una vez que se inserta/busca un elemento en el árbol se almacena un puntero al padre del último nodo visitado. De esta forma, si se realizan operaciones en un mismo directorio, el tiempo de búsqueda pasa a ser constante. Sin embargo, dependiendo del tipo de aplicación utilizada, esta optimización puede ser poco ventajosa, por ejemplo, si se escriben ficheros de forma aleatoria en varios niveles.

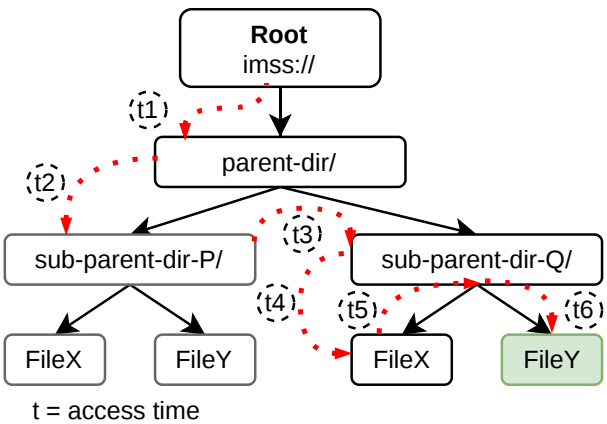


Fig. 2: Representación del proceso de búsqueda del fichero *imss://parent-dir/sub-parent-dir-Q/FileY* (peor caso) en un árbol de directorios jerárquico basado en estructuras GNode.

B. *HGetAttrb: localidad de los atributos de los ficheros en tablas de hashes.*

La segunda versión se denomina *HGetAttrb*, donde se detecta y reduce el número de veces que el *front-end* llama al servidor de almacenamiento de metadatos para obtener los atributos de un fichero/directorio. Para ello, se aprovecha la localidad de datos almacenando una copia de la estructura "stat" en cada cliente cuando este llama al servidor de metadatos por primera vez para obtener los metadatos de un fichero específico. En esta versión, también se observa que el uso de la estructura `GArray` para almacenar nuestros conjuntos de datos en el *front-end* no presenta un buen rendimiento cuando el número de elementos escala (con más de 100.000 archivos, la operación de búsqueda tarda más de 30 segundos). El cuello de botella se encontraba en una operación implementada para *HV1*, donde se almacenaba localmente una copia de los conjuntos de datos en un `GArray` para ser recuperada desde allí, evitando una llamada extra al servicio de almacenamiento de metadatos. Como desconocíamos la posición del elemento en el `GArray`, iterábamos a través de todos los elementos (complejidad de $O(n)$ [19]), comparando la clave esperada con la clave almacenada en cada conjunto de datos en el índice dado. En otro caso, conociendo el índice, la complejidad es $O(1)$.Por esta razón, decidimos cambiar a estructuras `GHashTable` [20], que ofrecen una complejidad $O(1)$ para buscar, insertar y eliminar, y $O(n)$ al iterar sobre todos los elementos. La desventaja de usar estructuras `GHashTable` es que, teóricamente, consumen más memoria. Sin embargo, tras realizar nuestras pruebas, hemos decidido que el rendimiento es más importante, dado que los sistemas HPC actuales disponen de gigabytes e incluso terabytes de memoria.

C. *HRM: optimización de la búsqueda y eliminación de ficheros.*

La versión *HRm* añade mejoras al recolector de basura (*garbage collector*) y a la estructura de árbol jerárquica utilizada en el servidor de almacenamiento de metadatos. Originalmente, cuando un cliente solicitaba una operación de metadatos, el servidor de almacenamiento de metadatos recorría el árbol para encontrar el nodo correspondiente (estructura `Gnode`). A medida que el árbol crecía, el tiempo de búsqueda también aumentaba, creando un nuevo cuello de botella, ya que el árbol se bloqueaba hasta que se completaba la operación. Para acelerar este procedimiento, se implementó un Mapa en el que la clave es la ruta absoluta del archivo, mientras que

el valor almacena el puntero al `GNode` del árbol para esa ruta (ver Figura 3). De esta forma, si el nodo corresponde a un directorio, podemos iterar directamente a través de sus hijos, lo cual es muy útil al realizar la operación de renombrar un directorio. En este punto, nos vimos obligados a mantener tanto la estructura de Mapa como la de Árbol, ya que la primera mejora el rendimiento de búsqueda dentro del árbol, pero no establece una jerarquía (directorios y sus hijos), mientras que la segunda sí lo hace.

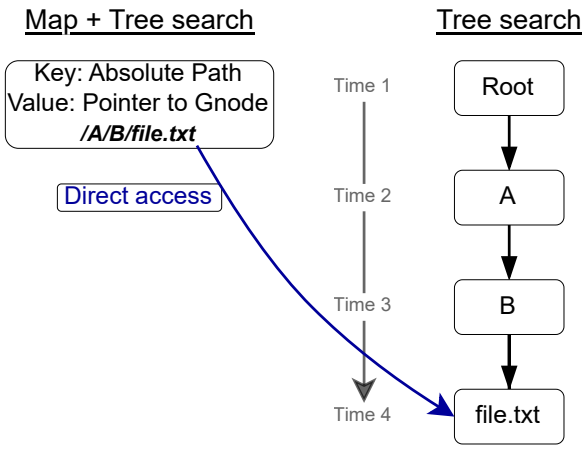


Fig. 3: Representación de la versión mejorada de *HRm*, que utiliza una combinación de árbol y mapa para acelerar las búsquedas en el servidor de metadatos. En una estructura de árbol tradicional, la búsqueda del archivo */A/B/file.txt* puede tardar el doble que utilizando un mapa indexado con punteros a los nodos del árbol.

Además, en *HRm* se reimplementó el componente del recolector de basura. En esta versión, cuando el cliente solicita una operación de borrado (*e.g.*, *rm* o *rmdir*), los servidores de almacenamiento de datos y metadatos no liberan la memoria realmente; en su lugar, se almacena la clave absoluta en un vector STL de C++ y se marca el conjunto de datos como *dirty* (sucio). Esto significa que, aunque los datos sigan en memoria e insertados en los mapas y en el árbol (en el caso del servidor de almacenamiento de metadatos), cuando un cliente quiera acceder a un archivo, primero comprobaremos si está marcado como *dirty*. Si es así, el servidor de almacenamiento del *back-end* devolverá al cliente del *front-end* que el archivo no existe. El inconveniente de este método es la sobrecarga añadida cuando el vector de archivos eliminados crece, ya que para cada listado de directorios se debe comprobar qué archivos están marcados como *dirty* y descartarlos de la lista de nombres devuelta al *front-end*. Sin embargo, más adelante en los resultados mostrados, veremos que la ganancia en operaciones por segundo en las tareas de eliminación de directorios y archivos es mayor que si no se implementara este método. Además, los archivos marcados como *dirty* se eliminarán una vez que la memoria utilizada por un servidor de almacenamiento de HERCULES alcance el 80 % de utilización. Si esto sucede, se lanzará el recolector de basura, el cual escaneará el vector de archivos *dirty*, identificándolos dentro del mapa principal, y entonces comenzará

RootMap	
imss://	DirMap1
imss://parent-dir	DirMap2
imss://parent-dir/sub-parent-dir-P	DirMap3
imss://parent-dir/sub-parent-dir-Q	DirMap4

DirMap1	
imss://parent-dir$0	Record 1

DirMap2	
imss://parent-dir/sub-parent-dir-P$0	Record 1
imss://parent-dir/sub-parent-dir-Q$0	Record 2

DirMap3	
imss://parent-dir/sub-parent-dir-P/fileX$0	Record 1
imss://parent-dir/sub-parent-dir-P/fileX$*n*	Record *n*-1
imss://parent-dir/sub-parent-dir-P/file*Y*$0	Record n
imss://parent-dir/sub-parent-dir-P/file*Y*$k	Record n+k

DirMap4	
imss://parent-dir/sub-parent-dir-Q/fileX$0	Record 1
imss://parent-dir/sub-parent-dir-Q/fileX$*n*	Record *n*-1
imss://parent-dir/sub-parent-dir-Q/file*Y*$0	Record n
imss://parent-dir/sub-parent-dir-Q/file*Y*$k	Record n+k

Fig. 4: Diseño gráfico del modelo de mapas jerárquicos utilizado en HERCULES para la gestión de datos y metadatos.

a liberar los bloques correspondientes a esos archivos. La liberación de bloques puede realizarse de dos formas: llamando a la función *free*, o almacenando el puntero de memoria en un *memory pool* para ser reutilizado. Es importante destacar que, al ser un sistema de archivos *ad-hoc*, HERCULES permite ajustar el porcentaje anterior según las necesidades de la aplicación.

D. HMaps: implementación de mapas jerárquicos.

Los cambios realizados en esta última implementación se centran en lo que hemos denominado Mapas Jerárquicos. La base de estos mapas se enfoca en la vinculación de mapas de C++. La Figura 4 muestra una representación visual del modelo de Mapas Jerárquicos utilizado, donde cada mapa corresponde a múltiples entradas de pares clave-valor. La estructura principal del Mapa es el *RootMap*, que almacena las rutas absolutas de todos los directorios creados en HERCULES (claves). Para cada clave, su valor apunta a un mapa diferente (ver *DirMap1*, *DirMap2*, *DirMap3* y *DirMap4*), que almacena los bloques de datos de todos los archivos guardados en ese directorio en particular.

La Figura 5 muestra la representación de la Figura anterior 4 siguiendo una estructura de árbol. Como se puede observar, en un árbol jerárquico convencional debemos navegar a través de él con el objetivo de encontrar el directorio o archivo requerido. Al utilizar el modelo de Mapa Jerárquico, se debe conocer la ruta absoluta del archivo, a partir de la cual se obtendrá la ruta del directorio padre. Por ejemplo, a partir de la ruta imss://parent-dir/sub-parent-dir-P/fileX, simplemente buscamos la última barra diago-

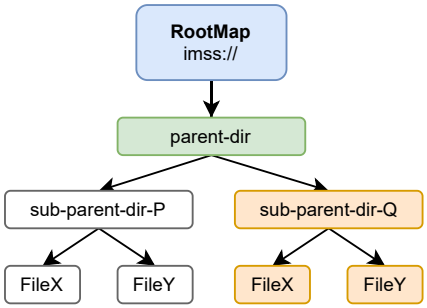


Fig. 5: Representación en árbol del mapa jerárquico mostrado en la Figura 4.

nal y así sabemos que el directorio padre es imss://parent-dir/sub-parent-dir-P. Tras conocer el directorio padre debemos utilizarlo como clave en el *RootMap*, de donde se extraerá el *DirMap* correspondiente. Siguiendo el ejemplo anterior, al directorio padre (clave) imss://parent-dir/sub-parent-dir-P le corresponde el *DirMap3* (valor), donde podemos localizar el fileX.

V. Evaluación

Esta sección describe la evaluación experimental realizada en este trabajo. Esta sección incluye la descripción del entorno experimental, las herramientas de prueba de rendimiento (*benchmarks*) y las aplicaciones reales empleadas, así como los resultados de la evaluación de la gestión de metadatos.

Ejecutamos el benchmark *Mdtest* para analizar el rendimiento de HERCULES, BeeGFS y Lustre en operaciones de metadatos. Hemos ejecutado 256 procesos en 16 nodos (16 procesos por nodo). Se ha desplegado un único servidor de almacenamiento de datos de HERCULES con 16 hilos, independiente de los procesos del *front-end*.

Esta evaluación de metadatos se llevó a cabo en la infraestructura HPC4AI. HPC4AI[1] es un clúster que consta de 68 nodos con Ubuntu 20.04.5 LTS. Cada nodo está equipado con dos procesadores Intel Xeon CPU E5-2697 v4 de 16 núcleos, con un total de 32 núcleos físicos y una velocidad de reloj por núcleo de 2.6 GHz. La topología de red se compone de tres conmutadores (*switches*) que conforman una red de tipo *fat-tree* de dos niveles. Todos los nodos de cómputo están conectados a través de una red Intel Omnipath, alcanzando un rendimiento pico de 100 Gbps. Los software empleados son UCX 1.15, OpenMPI 4.1 y *glib*. UCX expuso la red OPA mediante la librería *ibverbs*, alcanzando un ancho de banda similar al de la instalación nativa de OpenMPI.

A. Operaciones de metadatos basadas en directorios

La Figura 6 muestra, en escala logarítmica, los resultados obtenidos por Mdtest para operaciones de directorios, medidos en operaciones por segundo (ops). Las principales conclusiones que se pueden obtener son:

- En la mayoría de los casos, excepto en *Directory stat*, Lustre ha superado a HERCULES y BeeGFS debido a la caché del cliente.
- Para *Directory creation*, HERCULES alcanzó un mayor rendimiento en todas sus versiones en comparación con BeeGFS. Sin embargo, no se ha observado una mejora significativa entre las distintas versiones de HERCULES.
- En el caso de *Directory stat*, notamos que HERCULES superó a BeeGFS y a Lustre. *HGetAttrb* alcanzó el mejor rendimiento con 480,76 kops. Después de esto, *HRm* y *HHMaps* redujeron su rendimiento en 0,242 y 0,226 kops, respectivamente, consiguiendo aceleraciones de 7,22X y 6,80X respecto a BeeGFS.
- Para la operación *Directory rename*, BeeGFS alcanzó los 9,79 kops, demostrando una aceleración de 8,93X, 8,59X y 22,80X en comparación con *HV1*, *HGetAttrb* y *HRm*, respectivamente. Sin embargo, se ha demostrado que la última versión, *HHMaps*, incrementó su rendimiento en 5,34X.
- De manera similar, para la operación *Directory removal*, podemos observar un incremento en la aceleración de BeeGFS de 6,71X y 6,71X frente a *HV1* y *HGetAttrb*. No obstante, mientras BeeGFS alcanzó los 9,19 kops, *HRm* obtuvo 9,85 kops, lo que representa una ligera mejora. *HHMaps* presenta el mejor rendimiento para este caso, con 55,41 kops, alcanzando un *speedup* de 5X.

B. Operaciones de metadatos basadas en archivos

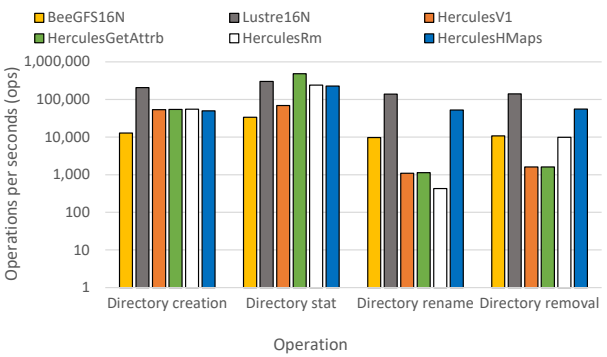


Fig. 6: Resultados de Mdtest para operaciones de directorio ejecutando diferentes versiones de HERCULES y comparadas con BeeGFS y Lustre.

La Figura 7 muestra los resultados obtenidos en el experimento para las operaciones de metadatos basadas en archivos. En estos experimentos, podemos observar resultados competitivos superiores en comparación con los obtenidos por Lustre.

Inicialmente, para *File creation*, *HV1* ya contaba con un mejor rendimiento que BeeGFS (44,96 frente a 8,70 kops), aunque Lustre alcanzó los 99,25 kops. *HGetAttrb* siguió el mismo comportamiento con 46,76 kops. Por el contrario, *HRm* ofreció un rendimiento deficiente con 8,68 kops, lo que podría deberse a la sobrecarga añadida al comprobar si un archivo existe en el vector de archivos eliminados, como se explicó anteriormente. Sin embargo, al utilizar

[1]https://hpc4ai.unito.it/documentation/

Tabla I: Comparación de las versiones para la Gestión de Metadatos en HERCULES. La complejidad algorítmica representa el peor caso.

Versión	Enfoque Principal	Estructuras de Datos	Complejidad Algorítmica	Ventajas	Limitaciones
HV1	Línea base con gestión centralizada.	`GArray` en el *front-end* para *stat* y la tabla de descriptores. `GNode` en el *backend* para el árbol jerárquico.	`GArray`; ordenación: $\boldsymbol{O(n^2)}$, búsqueda: $\boldsymbol{O(n)}$. `GNode`; búsqueda, inserción y borrado: $\boldsymbol{O(n)}$.	Menor complejidad en la gestión de jerarquías y almacenamiento de datos.	Cuello de botella en búsquedas al escalar (más de 30s con 100,000 archivos).
HGetAttrb	Localidad de los atributos de los datos (*stat*).	`GHashTable`.	Búsqueda, inserción y borrado: $\boldsymbol{O(1)}$. Iteración: $\boldsymbol{O(n)}$.	Reduce llamadas al servidor. Operaciones en tiempo constante.	Mayor consumo de memoria a cambio de un mayor rendimiento.
HRM	Aceleración de búsquedas y eliminación de ficheros.	Mapa clave/-valor con las direcciones de memoria al GNode de árbol jerárquico. Vector STL C++.	`Mapa`; búsqueda, inserción y borrado: $\boldsymbol{O(\log n)}$. `GNode`; búsqueda, inserción y borrado: $\boldsymbol{O(1)}$ para el árbol generado.	Evita bloqueos del árbol en las búsquedas. Eliminación optimizada mediante el etiquetado de ficheros "*dirty*".	Requiere mantener dos estructuras (Mapa y GNode). Sobrecarga añadida al tener que filtrar archivos *dirty* al listar.
HMaps	Localidad de los atributos de los datos (stat) y optimización en la gestión de la tabla de descriptores en el front-end y de metadatos en el back-end.	GHashTable en el front-end para *stat*. Mapas C++ enlazados en el front/back-end para la gestión de jerarquías.	`GHashTable`: búsqueda, inserción y borrado: $\boldsymbol{O(1)}$, iteración: $\boldsymbol{O(n)}$, `Mapa`; búsqueda, inserción y borrado: $\boldsymbol{O(\log n)}$.	Acceso casi directo a los datos, eliminando la necesidad de navegar un árbol tradicional.	Mayor complejidad en implementación. Requiere un mayor control para diferenciar directorios de ficheros regulares.

la versión *HHMaps*, se observa que el rendimiento se ha mejorado parcialmente hasta los 36,18 kops.

C. Operaciones de metadatos (Stat)

Para las operaciones *File stat* existe una degradación en el rendimiento desde *HGetAttrb* hasta *HHMaps* de 2,30X, y para *File read* hay una degradación de 2,61X. La causa principal es la sobrecarga añadida por las operaciones de eliminación de archivos (el vector de archivos eliminados y los mapas jerárquicos).

Como se puede ver en la Figura 6 y en la Figura 7, las pruebas realizadas utilizando 16 nodos en el supercomputador HPC4AI, muestran que las versiones iniciales de HERCULES funcionaban peor que BeeGFS al eliminar archivos/directorios. Sin embargo, en la versión *HHMaps* este problema se ha abordado, con cierta sobrecarga en las operaciones *stat* y *read*. No obstante, incluso con esta sobrecarga, *HHMaps* supera a BeeGFS en todas las operaciones, lo que nos permite realizar optimizaciones adicionales como trabajo futuro. De esta forma, para la operación *File removal*, la versión *HHMaps* representa una aceleración de 4,22X en comparación con BeeGFS y de 38,54X en comparación con *HV1*.

Finalmente, la Figura 8 muestra los resultados de Mdtest para operaciones de creación y destrucción del árbol de directorios con 16 nodos en el supercomputador HPC4AI.

En este caso, HERCULES ha mostrado un desempeño superior a BeeGFS y Lustre. Para *Tree create*, las operaciones por segundo de *HHMaps* disminuyen de 711 a 211, mientras que se observan mejoras con respecto a BeeGFS y Lustre, que han alcanzado 4,37 ops y 2,7 ops, respectivamente. Mientras tanto, para la operación *Tree removal*, *HGetAttrb* tiene un *speedup* de 9,73X en comparación con *HHMaps*, y este último tiene un *speedup* de 3,21X en comparación con BeeGFS y de 4,15X en comparación con Lustre.

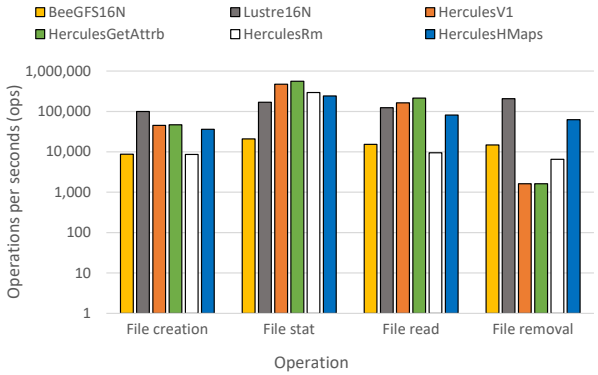


Fig. 7: Resultados de Mdtest para operaciones de archivos al ejecutar diferentes versiones de HERCULES y compararlas con BeeGFS y Lustre.

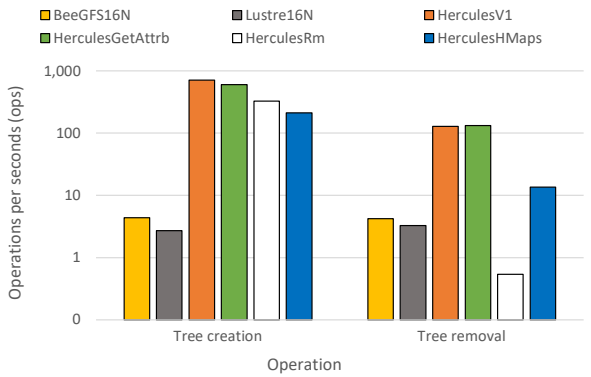


Fig. 8: Resultados de Mdtest para operaciones de creación y destrucción de un árbol de directorios ejecutando diferentes versiones de HERCULES y comparando BeeGFS y Lustre.

VI. Conclusiones

En este trabajo se han presentado cuatro técnicas diferentes para la gestión de metadatos en el sistema de ficheros *ad-hoc* HERCULES.

Se ha mostrado una comparativa con sistemas de ficheros consolidados como BeeGFS y Lustre arrojando resultados prometedores. HERCULES supera ampliamente en la creación y destrucción de estructuras de directorios en forma de árbol. Además, aplicar mapas jerárquicos (HHMaps) logra aceleraciones de hasta 5X en el borrado de directorios y supera a BeeGFS en casi todas las métricas de consulta de metadatos (stat). Hemos observado que la complejidad de los mapas jerárquicos introduce una ligera sobrecarga en las operaciones de lectura de archivos debido a las comprobaciones del estado de los ficheros (dirty), lo que representa una oportunidad para optimizaciones futuras.

Entre los trabajos futuros se incluye el estudio del comportamiento de las aplicaciones mediante técnicas de inteligencia artificial para predecir la mejor combinación entre las políticas de distribución de datos y las operaciones de E/S de las aplicaciones, de modo que puedan seleccionarse de forma transparente para el usuario.

Agradecimientos

Esta publicación es parte del proyecto de I+D+i PID2022-138050NB-I00, financiado por MICIU/AEI/10.13039/501100011033/"FEDER Una manera de hacer Europa" y por el Ministerio de Ciencia, Innovación y Universidades bajo la ayuda predoctoral con referencia PREP2022-000076. Adicionalmente, este trabajo ha recibido apoyo del supercomputador de la Universidad Carlos III de Madrid del proyecto "Centro Para El Análisis Y Modelado De Sistemas Complejos En Ingeniería Y Biomedicina" con referencia EQC2021-007184-P, financiado por el Programa Estatal para Impulsar la Investigación Científico-Técnica de la Agencia Estatal de Investigación (AEI).

Referencias

[1] Hao Dai, Yang Wang, Kenneth B. Kent, Lingfang Zeng, and Chengzhong Xu, "The State of the Art of Metadata Managements in Large-Scale Distributed File Systems - Scalability, Performance and Availability," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 3850–3869, 2022.

[2] Karan Vahi, Mats Rynge, Gideon Juve, Rajiv Mayani, and Ewa Deelman, "Rethinking data management for big data scientific workflows," in *2013 IEEE International Conference on Big Data*. IEEE, 2013, pp. 27–35.

[3] Raghavendra Kune, Pramod Kumar Konugurthi, Arun Agarwal, Raghavendra Rao Chillarige, and Rajkumar Buyya, "The anatomy of big data computing," *Software: Practice and Experience*, vol. 46, no. 1, pp. 79–105, 2016.

[4] Amazon Web Services, "Unlock higher performance for file system workloads with scalable metadata performance on Amazon FSx for Lustre," Feb. 2025.

[5] Fahim Chowdhury, Yue Zhu, Todd Heer, Saul Paredes, Adam Moody, Robin Goldstone, Kathryn Mohror, and Weikuan Yu, "I/O Characterization and Performance Evaluation of BeeGFS for Deep Learning," in *Proceedings of the 48th International Conference on Parallel Processing*. 2019, ICPP '19, Association for Computing Machinery.

[6] K. S. Shwetha, P. M. Javali, R. R. C, R. S. Shet, and N. Madhu, "A literature review of big data and metadata management systems," *International Journal for Research in Applied Science and Engineering Technology*, vol. 11, no. 2, pp. 1827–1836, Feb. 2023.

[7] JuiceFS, "Reduce metadata memory usage," `https://juicefs.com/en/blog/engineering/reduce-metadata-memory-usage`, Dec. 2024.

[8] Francisco Rodrigo Duro, Javier Garcia Blas, Florin Isaila, Jesus Carretero, Justin M Wozniak, and Rob Ross, "Exploiting data locality in swift/t workflows using hercules," in *Proc. NESUS Workshop*, 2014.

[9] J. Garcia-Blas et al., "Preliminary work of hercules," in *Lecture Notes in Computer Science*, 2022, Placeholder from user, please verify full details.

[10] Kyuli Park, Sungmin Yoon, Farid Talibli, Sungyong Park, Jae-Hyuck Kwak, Kimoon Jeong, Awais Khan, and Youngjae Kim, "DoCeph: DPU-Offloaded Messaging in Ceph for Reduced Host CPU Utilization," in *Proceedings of the SC'25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2025, pp. 1658–1666.

[11] Praveen M Dhulavvagol and SG Totad, "Performance enhancement of distributed system using HDFS federation and sharding," *Procedia Computer Science*, vol. 218, pp. 2830–2841, 2023.

[12] Gang Liao and Daniel J Abadi, "Filescale: Fast and elastic metadata management for distributed file systems," in *Proceedings of the 2023 ACM Symposium on Cloud Computing*, 2023, pp. 459–474.

[13] Swapnil Patil, Milo Polte, Kai Ren, Wittawat Tantisiriroj, Linus Salmon, Garth J. Gibson, Gregory R. Ganger, and Julio Lopez, "GIGA+: Scalable Directories for Shared File Systems," in *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST)*. 2011, pp. 153–166, USENIX Association.

[14] Jian Xu and Steven Swanson, "NOVA: A Log-Structured File System for Hybrid Volatile/Non-volatile Main Memories," in *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST)*. 2016, pp. 323–338, USENIX Association.

[15] Felix Garcia-Carballeira, Diego Camarmas-Alonso, Alejandro Caderon-Mateos, and Jesus Carretero, "A new Ad-Hoc parallel file system for HPC environments based on the Expand parallel file system," in *2023 22nd International Symposium on Parallel and Distributed Computing (ISPDC)*, 2023, pp. 69–76.

[16] Hao Guo, Youyou Lu, Wenhao Lv, Xiaojian Liao, Shaoxun Zeng, and Jiwu Shu, "SingularFS: A Billion-Scale distributed file system using a single metadata server," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, Boston, MA, July 2023, pp. 915–928, USENIX Association.

[17] Yiduo Wang, Yufei Wu, Cheng Li, Pengfei Zheng, Biao Cao, Yan Sun, Fei Zhou, Yinlong Xu, Yao Wang, and Guangjun Xie, "CFS: Scaling Metadata Service for Distributed File System via Pruned Scope of Critical Sections," in *Proceedings of the Eighteenth European Conference on Computer Systems*, New York, NY, USA, 2023, EuroSys '23, p. 331–346, Association for Computing Machinery.

[18] Wenhao Lv, Youyou Lu, Yiming Zhang, Peile Duan, and Jiwu Shu, "InfiniFS: An efficient metadata service for Large-Scale distributed filesystems," in *20th USENIX Conference on File and Storage Technologies (FAST 22)*, Santa Clara, CA, Feb. 2022, pp. 313–328, USENIX Association.

[19] Tom Copeland, "Manage c data using the GLib collections," June 2005.

[20] GLib Development Team, "GLib.HashTable," 2024.

Prolongando el Tiempo de Vida Útil de los Aceleradores ORB

Nicolás Landeros Muñoz, Alejandro Valero y Rubén Gran-Tejero[1]

Resumen— **El efecto *Negative Bias Temperature Instability* (NBTI) es una de las principales amenazas para la fiabilidad en los nodos tecnológicos actuales. El envejecimiento por NBTI afecta significativamente a los transistores PMOS utilizados para implementar las celdas de memoria SRAM, ya que degrada su tensión umbral (V_{th}) a lo largo del tiempo de vida útil de un circuito, lo que provoca una ralentización de los transistores y, en última instancia, da lugar a un funcionamiento defectuoso cuando el camino crítico supera el tiempo de ciclo.**

Este trabajo presenta dos contribuciones. En primer lugar, se realiza un estudio de caracterización de la aplicación ORB que identifica las celdas del subsistema de memoria integrado que sufren un mayor desgaste, y se cuantifica el impacto de los fallos provocados por NBTI en la precisión de los aceleradores ORB. En segundo lugar, se propone un mecanismo que garantiza un desgaste uniforme por NBTI en todas las celdas, basado en técnicas de inversión y rotación de bits del estado del arte, al que se añade el truncado de valores para permitir el *power gating* de las celdas y, así, su recuperación parcial con una afectación limitada a la precisión del acelerador ORB.

Los resultados experimentales muestran que, en comparación con el diseño convencional y el estado del arte, el ahorro en la degradación de V_{th} para la celda en el peor caso es de un 65 % y 15 %, respectivamente, mientras se incurre en pérdida mínima de precisión y penalización en el área. Al mismo tiempo, el ahorro de potencia es de un 35 % en comparación con el diseño convencional.

Palabras clave— **Aceleradores ORB, fallos permanentes, memoria integrada, Negative Bias Temperature Instability, power gating.**

I. Introducción

LOS flujos de trabajo de la visión artificial, robótica y realidad virtual/aumentada suelen basarse en la detección y descripción de puntos clave (*keypoints*), destacando particularmente el algoritmo *Oriented* FAST *and Rotated* BRIEF (ORB), debido a su robustez y eficiencia [1]. A medida que estas tareas se trasladan a plataformas con restricciones de energía, han surgido múltiples aceleradores ORB [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], organizados típicamente en torno a memorias estáticas de acceso aleatorio (SRAM) integradas en el chip para el almacenamiento de pirámides de imágenes, parches de imágenes, descriptores o colas de *keypoints*.

Sin embargo, la fiabilidad durante el tiempo de vida útil sigue sin explorarse para esta clase de aceleradores centrados en el uso de memoria. Los aceleradores ORB suelen integrarse en sistemas autónomos, donde los fallos son críticos y difíciles de mitigar, lo que puede comprometer el funcionamiento general del sistema. En tecnología CMOS con tamaño de nodo por debajo del micrómetro, los fenómenos de envejecimiento, como *Negative Bias Temperature Instability* (NBTI), provocan cambios en la tensión umbral del transistor (V_{th}) y reducen la corriente de conducción, lo que aumenta los retardos de propagación y erosiona los márgenes en el tiempo de ciclo. Es fundamental señalar que la degradación de los transistores depende de la carga de trabajo: el estrés DC prolongado (desbalance del *duty cycle* o ciclo de trabajo) agrava el efecto NBTI en los dispositivos PMOS. El impacto de NBTI es especialmente pronunciado en los transistores de las celdas SRAM porque, aunque el dispositivo sometido a estrés puede variar con el tiempo, siempre existe al menos un transistor en la celda experimentando degradación por NBTI [13].

En la práctica, el envejecimiento de la memoria se mitiga a través de dos enfoques principales: (i) la aplicación de márgenes de temporización (*guardbanding*) —por ejemplo, mediante el aumento de la tensión de alimentación (V_{dd}) y la relajación de las restricciones de tiempo— y (ii) el endurecimiento de la memoria (*memory hardening*) a través de diseños de celdas robustos. *Guardbanding* amplía la vida útil del sistema, pero degrada el rendimiento y la eficiencia energética, ya que la energía dinámica escala cuadráticamente con V_{dd}. Por el contrario, las celdas endurecidas mejoran la estabilidad de lectura a expensas de la densidad, la simplicidad, el rendimiento, el consumo de energía y el coste del chip. En general, las técnicas convencionales sacrifican rendimiento, eficiencia energética o área de silicio en favor de la fiabilidad, lo que motiva la búsqueda de soluciones que prolonguen la vida útil de los dispositivos sin incurrir en estas penalizaciones.

A nuestro mejor saber y entender, este es el primer trabajo que caracteriza el efecto de envejecimiento por NBTI en aceleradores ORB y su impacto global en las métricas de precisión de ORB. Este artículo aborda esta carencia mediante: (i) la construcción de un modelo de acelerador ORB preciso a nivel de ciclo (*cycle-accurate*) que captura la organización de la memoria y la señalización de control relevantes para la exposición a NBTI, (ii) la caracterización del estrés de los transistores utilizando conjuntos de datos representativos de ORB, (iii) el mapeo del estrés hacia la degradación proyectada de V_{th} (ΔV_{th}) utilizando un modelo de envejecimiento establecido y (iv) la cuantificación de cómo los fallos inducidos por NBTI afectan a métricas de ORB como *Image Matching* (IM), *Keypoint Retrieval* (KR) y *Keypoint Verification* (KV) [14].

[1]Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Instituto de Investigación en Ingeniería de Aragón – I3A, España, e-mail: {nlanderos,alvabre,rgran}@unizar.es.

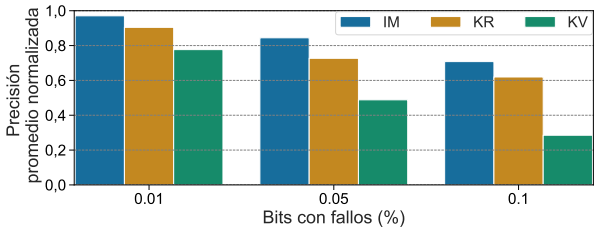


Fig. 1: Precisión promedio de un acelerador ORB con fallos inducidos por NBTI en su memoria integrada, normalizada con respecto al diseño ideal sin fallos.

La Figura 1 ilustra la degradación relativa de las métricas IM, KR y KV en nuestro acelerador ORB tras la inyección de fallos permanentes en un subconjunto de toda la memoria integrada. Estos fallos se introducen en las celdas más sensibles a NBTI de acuerdo con el estudio de caracterización del envejecimiento (véase la Sección III). Las tasas de fallos de 0,01 %, 0,05 % y 0,1 % corresponden a 2, 11 y 23 bits defectuosos, respectivamente, en toda la memoria. La precisión se deteriora notablemente con el número de fallos. Para las métricas KR y KV, la pérdida de rendimiento supera el 10 % y el 20 %, respectivamente, con la presencia de solo dos bits defectuosos.

Guiado por la caracterización del envejecimiento, este es el primer trabajo que propone un diseño microarquitectónico para distribuir uniformemente el estrés por NBTI entre todas las celdas de memoria de los aceleradores ORB. El diseño propuesto mitiga eficazmente la ΔV_{th} proyectada sin modificar el algoritmo ORB ni sus resultados. Además, en comparación con el diseño convencional, el enfoque propuesto reduce significativamente la disipación de potencia con una pérdida de precisión y una penalización en el área mínimas.

El mecanismo propuesto no solo aprovecha técnicas de nivelación de desgaste del estado del arte consistentes en operaciones de inversión y rotación periódica de bits, sino que también añade un esquema de truncamiento de datos a través de *power gating* adaptado a los aceleradores ORB. Además, este trabajo proporciona una evaluación rigurosa y detallada del impacto de la arquitectura propuesta en la degradación de la memoria, apoyándose en un análisis minucioso de diseño VLSI y en un modelado a nivel de circuito basado en SPICE.

El impacto de las técnicas propuestas en la precisión se evalúa utilizando métricas ORB tradicionales, así como una aplicación de visión en tiempo real empleando ORB en una tarea de localización y mapeo simultáneo (SLAM) [15]. Los resultados experimentales indican que, en comparación con el diseño convencional y el estado del arte, el enfoque propuesto reduce la ΔV_{th} en el peor caso en un 65 % y 15 %, respectivamente, al tiempo que incurre en una pérdida de precisión y penalización de área mínimas. Además, el diseño propuesto logra una reducción del 35 % en la disipación de potencia en comparación con el diseño convencional.

El resto de este trabajo se organiza de la siguiente manera. La Sección II proporciona los antecedentes. La Sección III presenta el estudio de caracterización. La Sección IV describe las técnicas propuestas. La Sección V cubre la evaluación experimental. La Sección VI resume los trabajos relacionados. Finalmente, la Sección VII presenta las conclusiones de este trabajo.

II. Antecedentes

Esta sección describe el algoritmo ORB, la organización de memoria típica de los aceleradores ORB del estado del arte, el efecto de envejecimiento por NBTI, el estado del arte en la mitigación de NBTI para memorias de vídeo integradas y la técnica de *power gating* como posible medida para combatir el efecto NBTI.

A. Algoritmo ORB

ORB es una técnica avanzada de visión por computador utilizada principalmente para la detección y descripción de características locales en imágenes [1]. Fue desarrollada para ofrecer una alternativa más rápida y computacionalmente eficiente a métodos tradicionales como *Scale-Invariant Feature Transform* (SIFT) [16] o *Speeded-Up Robust Features* (SURF) [17]. ORB destaca por su rendimiento en tiempo real y su robustez frente a transformaciones geométricas y cambios de iluminación.

El flujo de datos del algoritmo ORB comienza escaneando una imagen de entrada en escala de grises fila por fila. Los píxeles de la imagen se envían en paralelo a los módulos de detección, orientación y descripción. El módulo de detección analiza un parche de imagen de 7×7 píxeles en una unidad detectora de *keypoints* y determina si el píxel central es un *keypoint* [18]. El módulo de orientación recibe la posición del *keypoint* y calcula su ángulo basándose en el centroide de intensidad sobre un parche de imagen de 37×37 píxeles [19]. Por último, el módulo de descripción toma el ángulo y la posición producidos por los módulos precedentes, calculando su descriptor realizando múltiples comparaciones dentro de un parche de imagen de 37×37.

B. Aceleradores ORB

Los aceleradores ORB mejoran el rendimiento (*throughput*) y la eficiencia energética en comparación con los procesadores de propósito general. Dado que la mayoría de las operaciones del algoritmo son comparaciones simples y, por lo tanto, baratas de implementar en hardware, el componente hardware principal es el subsistema de memoria integrado en el chip, necesario para almacenar los parches de la imagen.

La Figura 2 muestra la implementación habitual del módulo de detección para parches de imagen de 7×7, la cual consta de un subsistema de memoria y una ventana de registros de desplazamiento [7], [8], [10], [11]. La ventana de registros de desplazamiento se utiliza comúnmente para dar soporte a convoluciones 2D en el hardware de procesamiento de imágenes, proporcionando almacenamiento temporal y un mecanismo de sincronización a través del subsistema

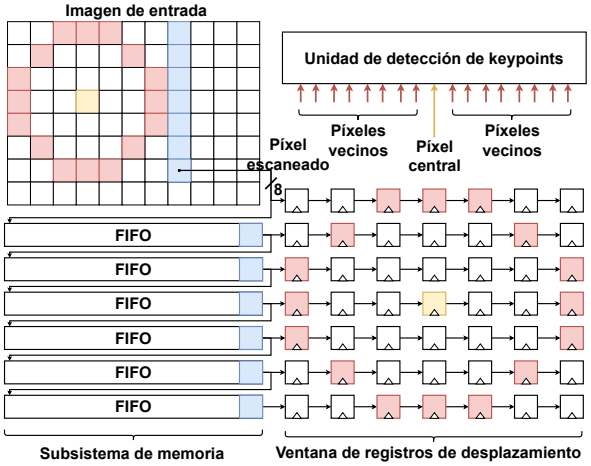


Fig. 2: Organización de la memoria en el módulo de detección.

de memoria compuesto por memorias SRAM FIFO que almacenan en caché múltiples filas de la imagen de entrada. Cabe destacar que se utiliza una implementación análoga para los módulos de orientación y descripción, pero escalada para procesar parches de 37×37 píxeles. En este caso, ambos módulos comparten el mismo subsistema de memoria.

Para permitir la operación de ventana deslizante mostrada en la Figura 2, el subsistema de memoria, compuesto por seis FIFOs, almacena las seis filas de la imagen leídas más recientemente. Estas memorias utilizan punteros de lectura y escritura *round-robin* para acceder simultáneamente a las cabezas y colas de la memoria (resaltadas en azul), respectivamente. Los píxeles se introducen en la ventana de registros de desplazamiento, dimensionada para almacenar parches de imagen de 7×7, donde el flujo de píxeles avanza de forma secuencial a través de los registros en cada ciclo. Los cálculos se realizan una vez que toda la ventana de registros de desplazamiento se llena tras un número fijo de ciclos. Estos cálculos son ejecutados por la unidad de detección de *keypoints*, que toma el píxel central (coloreado en amarillo) y sus píxeles vecinos (resaltados en rojo) de la ventana deslizante para determinar si cumplen con los criterios para ser clasificados como *keypoints*. Cuando la ventana alcanza el límite del bloque, la arquitectura requiere un número fijo de ciclos para alinear la ventana con la siguiente fila.

C. Fenómeno de Envejecimiento por NBTI

NBTI afecta predominantemente a las memorias SRAM FIFO en lugar de a la ventana deslizante basada en registros debido a sus diferencias de robustez frente a las variaciones de parámetros. Mientras que los registros de desplazamiento de celdas estándar toleran bien el aumento de V_{th}, las celdas SRAM dependen de un emparejamiento preciso de las características eléctricas de sus transistores. Esta dependencia hace a la celda SRAM mucho más susceptible a los fallos de lectura/escritura inducidos por el envejecimiento [20]. A su vez, entre los transistores que componen una celda SRAM diferencial de 10T, la cual es adecuada para aplicaciones de doble puerto de alta velocidad en el procesamiento de vídeo [21], los dos dispositivos PMOS que constituyen los inversores acoplados en cruz, responsables de almacenar el estado lógico, son los principales transistores susceptibles al NBTI.

$$\Delta V_{th} = A \times t_{ox} \times \sqrt{C_{ox} \times (V_{gs} - V_{t_0})} \times e^{\frac{V_{gs}-V_{t_0}}{t_{ox}\times E} - \frac{Ea}{k\times T}} \times {t_{stress}}^{0,25} \times (1 - \sqrt{\eta \times \frac{t_{rec}}{t_{stress} + t_{rec}}}) \quad (1)$$

Los transistores PMOS experimentan envejecimiento cuando se aplica un '0' lógico a los terminales de su puerta. En una celda SRAM, esto se manifiesta en dos condiciones de funcionamiento complementarias. Cuando la celda almacena de forma estable un '0' lógico, uno de estos transistores se somete a estrés, mientras que el almacenamiento de un '1' lógico impone estrés en el dispositivo complementario. Debido a la naturaleza alterna de las fases de estrés y recuperación durante el funcionamiento normal, el transistor que no se encuentra actualmente bajo estrés experimenta una recuperación parcial, mitigando así los efectos de la degradación acumulativa. Como resultado, si una celda alterna de manera uniforme entre el '0' lógico y el '1' lógico, el desgaste se reparte simétricamente entre los dos dispositivos. De este modo, enfoques previos consistentes en técnicas de inversión periódica y rotación de bits han equilibrado con éxito el desgaste por NBTI entre las celdas de memoria de diferentes sistemas de almacenamiento [22], [23], [24].

La Ecuación 1 proporciona la expresión estándar para la degradación de V_{th} (ΔV_{th}) causada por NBTI en los transistores PMOS de la celda. A excepción de t_{stress} y t_{rec}, que cuantifican el tiempo (en segundos) bajo estrés y recuperación, respectivamente, y de la temperatura T, que depende de la simulación y se establece en 80 ^{o}C, todos los demás parámetros se consideran constantes para un nodo tecnológico de 45 nm [25], [26], [27], [28], [29].

D. Estado del Arte en Memorias de Vídeo y Power Gating Consciente de NBTI

Los estudios existentes no han abordado el envejecimiento por NBTI en los aceleradores ORB. El trabajo más cercano al nuestro es el enfoque *Application-Specific Configurable Aging Resilience* (ASCAR) [30], [31]. Mediante la aplicación periódica de la inversión y rotación de bits, ASCAR equilibra la degradación por NBTI en la memoria integrada utilizada para almacenar fotogramas de vídeo en bruto en una arquitectura genérica de procesamiento de vídeo. Este enfoque establece un objetivo de extensión de la vida útil definido por el usuario, ya que impone un compromiso entre el equilibrio del desgaste y la disipación de potencia.

La técnica de *power gating* ha sido ampliamente adoptada para reducir significativamente el con-

sumo de energía de los circuitos de memoria [32]; además, también puede aprovecharse para mitigar el efecto NBTI. Un esquema de *power gating* consciente de NBTI coloca un transistor *footer* NMOS entre la celda de memoria y tierra. Cuando este transistor conduce (estado activo), la celda funciona con normalidad manteniendo estados lógicos complementarios. Por el contrario, cuando el transistor *footer* se desactiva (estado de apagado), ambos nodos internos transitan hacia un '1' lógico, lo que sitúa efectivamente a ambos transistores PMOS en un estado de recuperación profunda [33]. Dicho estado de apagado proporciona potencialmente una mayor mitigación de NBTI en comparación con las técnicas de inversión y rotación de bits del estado del arte como ASCAR.

III. Estudio de Caracterización

Esta sección profundiza en el impacto de NBTI en los transistores PMOS de las celdas SRAM usadas en memorias de aceleradores ORB. En primer lugar, se presenta la metodología para el estudio de caracterización. Luego se reportan los resultados, incluyendo una evaluación cuantitativa de la degradación de la precisión.

A. Metodología

Hemos desarrollado un modelo preciso a nivel de ciclo del acelerador ORB sobre el *framework* de extracción de características de ORB implementado en Python. Utilizando este modelo, obtuvimos el tiempo de ejecución y toda la actividad de memoria relevante —incluyendo direcciones, datos y tipo de operación (lectura/escritura) para cada acceso a memoria— necesaria para cuantificar el tiempo de estrés causado por NBTI.

Hemos empleado el conjunto de datos HPatches, ampliamente utilizado, como entrada [14]. Este conjunto de datos consta de 116 secuencias de imágenes para evaluar descriptores de características locales en condiciones variables. HPatches define *Image Matching* (IM), *Keypoint Retrieval* (KR) y *Keypoint Verification* (KV) como métricas de precisión.

Finalmente, hemos extendido el modelo del acelerador con un *framework* de inyección de fallos a nivel de celda de memoria. De este modo, los fallos inducidos por NBTI se inyectan sobre 100 muestras en las celdas identificadas como vulnerables por el siguiente estudio de caracterización.

B. Resultados

La Figura 3 ilustra las distribuciones de peor caso de estrés[1] para los subsistemas de memoria tanto de detección como de orientación/descripción, donde el estrés se define como la fracción de tiempo en la cual la celda almacena un valor lógico dado, ya sea '0' o '1', sobre el tiempo total de ejecución ($\frac{t_{stress}}{t_{total}}$), para cada palabra de 8 bits representando un píxel. El

[1]Los extremos de los rectángulos indican los percentiles 75 y 25, la línea interior marca la mediana y las barras representan los valores máximo y mínimo.

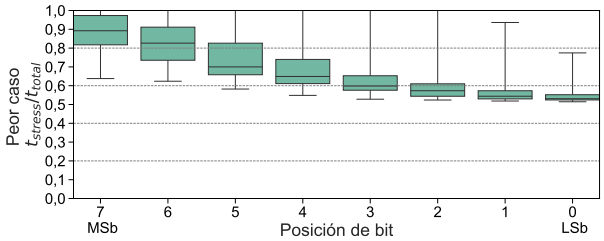
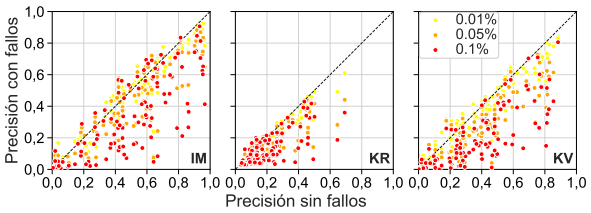
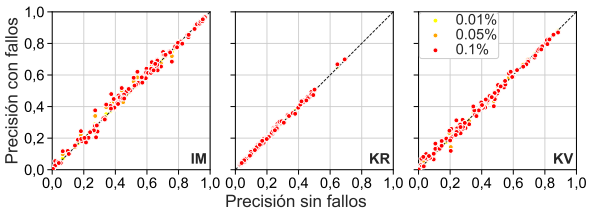


Fig. 3: Distribuciones de peor caso de $\frac{t_{stress}}{t_{total}}$ para todos los transistores PMOS en cada posición de bit de las palabras de 8 bits.



(a) MSb (bits 7-5)



(b) LSb (bits 2-0)

Fig. 4: Precisión en bruto con fallos contra precisión sin fallos para cada imagen de la secuencia.

peor caso reportado corresponde al máximo estrés observado entre un '0' y un '1' lógico. Las distribuciones se agrupan por posición de bit dentro de la palabra, desde el bit 7 (más significativo o MSb) hasta el bit 0 (menos significativo o LSb). Cada distribución comprende 116 valores correspondientes al número de secuencias en HPatches.

Tal como se indica en [31], las distribuciones de estrés aumentan de manera consistente en las posiciones de bit de mayor orden. En otras palabras, los bits más significativos tienden a permanecer sesgados hacia un valor lógico '0' o '1', lo que induce una mayor degradación por NBTI en dichas posiciones de bit. Notablemente, los valores medios para los bits 7 y 6 exceden el 80 %, mientras que el valor máximo alcanza el 100 % desde el bit 7 hasta el 2. A diferencia de [31], los bits 1 y 0 tienen al menos una celda con condición de estrés sesgada excediendo el 75 %, sugiriendo que el desgaste por NBTI debería distribuirse uniformemente a través de todas posiciones de bits dentro de una palabra.

La Figura 4 muestra la degradación de las métricas de precisión IM, KR, y KV en función de la tasa de fallos relativa al tamaño total de la memoria (0,01 %, 0,05 %, y 0,1 %). Cada gráfica compara la precisión con fallos contra la precisión original sin fallos para cada una de las 116 secuencias.

La precisión original es consistente con la reportada por Balntas *et al.* [14] y Bojanić *et al.* [34]. Los puntos situados en la diagonal indican una pérdida de precisión nula, mientras que los puntos situados

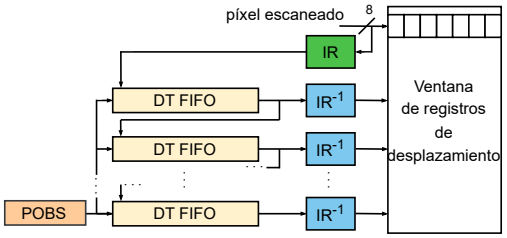


Fig. 5: Enfoque propuesto consistente en memorias DT FIFO, unidades IR/IR^{-1} y un registro POBS.

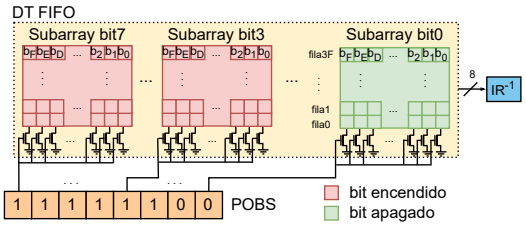


Fig. 6: Detalle de implementación de una memoria DT FIFO.

bajo la diagonal denotan una degradación en la presión causada por la inyección de fallos. El análisis contrasta el efecto de inyección de fallos enfocado en los bits más significativos desde el 7 al 5 con los menos significativos desde el 2 al 0. Los fallos en los bits más significativos tienen un impacto severo en la precisión de ORB, resultando en una degradación más temprana y pronunciada a medida que la tasa de fallos se incrementa. Por el contrario, los fallos en los bits menos significativos conducen a una degradación mínima, indicando un grado de robustez mayor a los fallos. Sorprendentemente, unas pocas secuencias (por encima de la diagonal) se benefician del ruido introducido por los fallos, mejorando levemente la precisión.

Tres observaciones clave emergen de este estudio de caracterización. En primer lugar, la fiabilidad de ORB depende en gran medida de la posición del bit afectado por los fallos. En segundo lugar, un envejecimiento irregular de la memoria puede inducir una degradación prematura de la precisión del acelerador ORB. Tercero, ORB exhibe tolerancia intrínseca a fallos en los bits menos significativos, una propiedad que es aprovechada explícitamente por nuestra propuesta en la siguiente sección.

IV. Diseño Propuesto

En base a la observación de que los valores convencionales de píxeles de 8 bits típicamente proveen más precisión de la que el algoritmo de ORB realmente requiere, el mecanismo propuesto combina: i) truncamiento de datos a través de *power gating* selectivo a subarrays de bits para reducir el estrés eléctrico y ii) rotación intra-palabra para distribuir uniformemente la reducción de estrés a través de todas las celdas de memoria, además de un mecanismo de inversión de bits para forzar una alternancia equilibrada entre los estados lógicos de la celda.

A. Visión General

La Figura 5 ilustra una visión general de la arquitectura propuesta para ambos subsistemas de memoria. La unidad de Inversión-Rotación (IR), localizada en la cola de la primera memoria FIFO, transforma los píxeles escaneados de acuerdo a las operaciones de inversión de bits y rotación de bits. Por el contrario, la unidad IR inversa (IR^{-1}), posicionada en la cabeza de cada FIFO, revierte la transformación antes de enviar los píxeles a la ventana de registros de desplazamiento. Además, las memorias FIFO incorporan transistores *footer* para realizar *power gating* selectivo de subarrays, resultando en FIFOs con truncado de datos (DT FIFOs). Finalmente, un registro global de apagado de subarrays (POBS) indica qué subarrays se apagan.

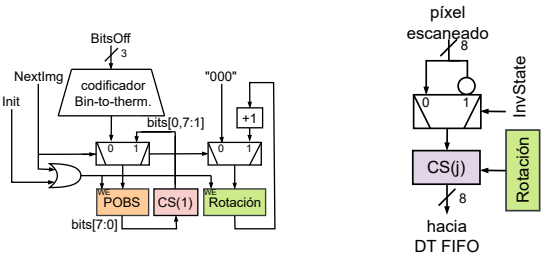
B. Truncado de Datos

La Figura 6 muestra el diseño de un DT FIFO. Para minimizar la penalización de *routing*, los DT FIFO se organizan en ocho 64×16 subarrays, cada uno almacenando una posición de bit específica de todos los píxeles. Dichas dimensiones de subarray establecen un compromiso adecuado entre latencia, potencia y área. Cada columna de subarray se conecta a un transistor *ooter* de *power gating*. De esta forma, los DT FIFO son capaces de apagar selectivamente subarrays específicos de acuerdo a un ancho de bit de píxel predefinido (véase la Sección IV-C). Los transistores *footer* están dimensionados de forma conservadora para no superar un *offset* de tierra virtual de 50 mV [33].

El registro POBS establece qué subarrays se encienden/apagan conectando el bit i-ésimo del registro a la puerta de todos los *footers* del subarray correspondiente al bit$_i$. Específicamente, un bit de POBS establecido a un '1' ('0') lógico indica que el subarray correspondiente está encendido (apagado). Este registro se comparte a través de todos los DT FIFO, asegurando que los mismos subarrays están apagados a través de todos ellos. Cuando un píxel se escribe en un DT FIFO, los bits correspondientes a subarrays apagados se truncan (no se almacenan) sin importar su valor. De forma similar, cuando un píxel se lee, los bits de salida asociados a subarrays apagados se retornan siempre con valor lógico '0'.

C. Rotación e Inversión Intra-Palabra

La Figura 7a muestra el circuito responsable de inicializar y actualizar el registro POBS. En un estado inicial (*Init*=1) para el cual las imágenes de entrada no son procesadas aún por el acelerador (*NextImg*=0), este circuito configura el registro POBS de acuerdo a un ancho de bit definido por software (es decir, el número de subarrays apagados en los DT FIFO) establecido por la señal *BitsOff*. Para este fin, *BitsOff* se codifica mediante una representación unaria, la cual establece el número correspondiente de bits menos significativos en el registro POBS con estado apagado ('0' lógico), mientras que los bits restantes se establecen con estado encendido



(a) Rotación de POBS (b) Unidad IR

Fig. 7: Implementación de inversión y rotación de bit.

('1' lógico). Por ejemplo, si *BitsOff*= 010 entonces POBS=11111100. Al mismo tiempo, el registro de Rotación se inicializa a cero.

Para evitar la recuperación continua de los mismos subarrays, el mecanismo propuesto rota periódicamente los subarrays apagados. Este comportamiento es controlado en conjunto por los registros de Rotación y POBS. Este último realiza un seguimiento del *offset* de rotación actual. Al recibir una nueva imagen (*NextImg*=1), el patrón del registro POBS es rotado cíclicamente a la izquierda en una posición de bit mediante un desplazador circular de 1 bit (CS-1), mientras que el registro de Rotación es incrementado en uno (módulo ocho). Después de eso, *NextImg* se mantiene en '0' hasta que se procese una nueva imagen.

La Figura 7b muestra el diseño de la unidad IR. Esta unidad aplica una inversión del valor de los bits al flujo de píxeles de entrada. La inversión de bit se realiza a través de un multiplexor manejado por la señal *InvState*, la cual se actualiza tras ocho imágenes procesadas. [2] Además, la unidad IR se asegura de que los bits truncados (apagados) sean siempre los menos significativos. Dado que POBS rota cada nueva imagen de entrada, esta unidad aplica una rotación circular a la izquierda igual que el *offset* j indicado por el registro de Rotación, manteniendo el mapeo correcto de las posiciones de bits de acuerdo a los subarrays apagados. [3]

Las unidades IR^{-1} (no mostradas en la figura) realizan las operaciones inversas: revierten la rotación de bit y deshacen la inversión (en caso de haberla) de tal manera que los valores del píxel reconstruido son consistentes con la semántica original de los datos y suministrados a la ventana de registros deslizante.

V. Evaluación Experimental

Esta sección cuantifica los sobrecostes del diseño propuesto y evalúa su impacto en la mitigación de envejecimiento y la pérdida de precisión. Los conjuntos de datos bajo evaluación incluyen no solo HPatches sino también todas las secuencias de KITTI [35] y EuRoC [36] utilizadas en SLAM.

[2]Nótese que las frecuencias de rotación e inversión (una y ocho imágenes, respectivamente) no coinciden para evitar el acoplamiento del estado de inversión y el patrón de apagado.

[3]La inversión de bits propuesta no agrava el efecto de envejecimiento *hot carrier injection* asociado con la actividad de conmutación, dado que se aplica a granularidad de imagen de entrada. Más aún, la rotación distribuye uniformemente la actividad de conmutación a través de todas las celdas.

Componente	Potencia (mW)	
	Estática	Dinámica
Conv	0,136	4,431
DT FIFO-0	0,138	4,552
DT FIFO-1	0,129	4,131
DT FIFO-2	0,120	3,710
DT FIFO-3	0,111	3,289
DT FIFO-4	0,102	2,868
Rot. de POBS	$6{,}291 \times 10^{-5}$	$3{,}472 \times 10^{-4}$
Unidad $\mathrm{IR}/\mathrm{IR}^{-1}$	$1{,}499 \times 10^{-5}$	$2{,}637 \times 10^{-2}$

Tabla I: Disipación de potencia para distintos componentes. Los DT FIFO varían el número de subarrays apagados de 0 a 4.

A. Sobrecostes de Potencia, Temporización y Área

El truncado de datos mediante *power gating* a nivel de subarray fue implementada inicialmente generando un diseño en OpenRAM basado en el nodo tecnológico FreePDK45 de 45 nm [25], [26]. Posteriormente, dicho diseño inicial se modificó añadiendo un esquema de *power gating* a nivel de columna, combinando transistores *footer* para los subarrays de memoria con aislamiento de las señales del circuito periférico.

La lógica auxiliar (rotación de POBS y las unidades $\mathrm{IR}/\mathrm{IR}^{-1}$) corresponden a un diseño *full-custom*. Particularmente, la rotación de POBS fue diseñada utilizando transistores con $\mathrm{V_{th}}$ elevada para minimizar el sobrecoste de consumo. Por el contrario, dado que las unidades $\mathrm{IR}/\mathrm{IR}^{-1}$ están en el camino crítico de los subsistemas de memoria, se implementaron mediante transistores VTG, mientras que el circuito CS(j) se diseñó como un *crossbar barrel shifter*. Los números de potencia y temporización se obtuvieron mediante simulaciones de SPICE.

La Tabla I reporta la disipación de potencia (en mW) de los diferentes componentes del subsistema de memoria. Las etiquetas *Conv* y *DT FIFO-k* denotan una implementación FIFO convencional y el diseño DT FIFO propuesto con k subarrays apagados, respectivamente. La potencia estática se corresponde con las corrientes de fuga, mientras que la potencia dinámica asume dos accesos a la FIFO por ciclo (uno de lectura y otro de escritura) y un factor de actividad derivado del análisis de caracterización de cada componente.

Las diferencias entre Conv y DT FIFO-0 reflejan el sobrecoste de *power-gating* y aislamiento de periferia. Como era de esperar, cuanto mayor es el número de subarrays apagados, mayores son los ahorros de potencia de DT FIFO. Las reducciones de consumo dinámico se deben principalmente a anchos de palabra efectivos más estrechos y a una menor actividad de conmutación tanto en el subarray como en los circuitos periféricos, compensando en gran medida el sobrecoste de la gestión de la DT FIFO y de la lógica adicional, el cual es al menos dos órdenes de magnitud inferior en comparación con el de DT FIFO.

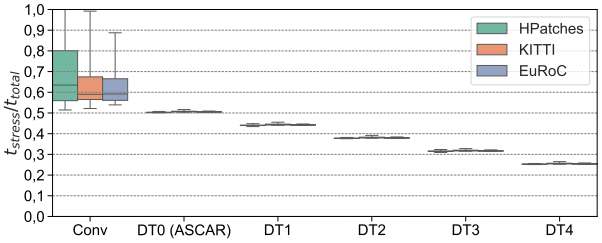


Fig. 8: Distribuciones de $\frac{t_{stress}}{t_{total}}$ de todos los transistores PMOS en el subsistema de memoria.

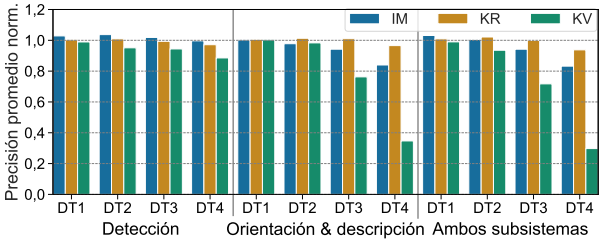


Fig. 9: Precisión promedio para las métricas de HPatches normalizada respecto al diseño ideal sin fallos.

Dataset	Conv	DT1	DT2	DT3	DT4
KITTI	2,6	2,7	2,7	2,9	3,1
EuRoC	11,1	10,5	10,3	10,2	10,5

Tabla II: Error absoluto de trayectoria promedio (ATE) medido en metros y centímetros para KITTI y EuRoC, respectivamente.

Por otro lado, el sobrecoste de consumo asociado a la reactivación (encendido) de un subarray es despreciable (0,192 nW por DT FIFO) y la corriente de irrupción es inferior a 0,13 mA por transistor *footer*. En resumen, para ambos subsistemas de memoria, el ahorro total de consumo del enfoque propuesto con $k = 4$ es de un 35 % en comparación con el diseño convencional.

El sobrecoste de retardo se considera despreciable. Las unidades IR^{-1}, las cuales se encuentran en el camino crítico, aumentan la latencia en 150 ps. Sin embargo, este sobrecoste puede ocultarse eficazmente mediante segmentación. Del mismo modo, el aislamiento periférico solo introduce 20 ps adicionales a las operaciones de lectura y escritura.

Cabe destacar también que la latencia de reactivación del subarray, cuantificada en 1,78 ns, no se encuentra en el camino crítico, ya que los subarrays necesarios se alimentan antes de recibir la siguiente imagen de acuerdo con la rotación cíclica.

Finalmente, la complejidad lógica del circuito externo de rotación de POBS más la unidad IR es estimado en 345 puertas equivalentes (GE), donde una GE se define como el área física de una puerta NAND estándar de 2 entradas. Adicionalmente, las unidades IR^{-1} combinadas con la lógica de *power gating* y el aislamiento de señales suponen 298 GE por FIFO. Evaluando dichos módulos usando la librería estándar FreePDK45 (1 GE = 0,798 μm^2) y aplicando un factor de utilización conservativo de 0,5 para acomodar la congestión de *routing* [37], el sobrecoste total de área se mantiene estrictamente por debajo de 1,2 % para ambos subsistemas de memoria.

B. Impacto en el Envejecimiento y la Precisión

La Figura 8 ilustra las distribuciones de estrés inducido por NBTI para todos los transistores PMOS de ambos subsistemas de memoria. Las etiquetas *Conv* y *DTk* hacen referencia al diseño convencional y el enfoque propuesto con k subarrays apagados, respectivamente. Conv exhibe un estrés significativo en transistores específicos, alcanzando un estrés máximo para el caso de HPatches y KITTI. En contraste, al realizar inversión y rotación periódica, DT0 equilibra uniformemente el estrés a través de todos los transistores, estableciendo el tiempo de estrés a 0,5. Nótese que este es el mejor resultado posible para enfoques basados puramente en inversión y rotación, incluyendo la técnica de estado del arte ASCAR (véase la Sección II-D).

La combinación de inversión y rotación con el truncamiento de datos ayuda a minimizar el tiempo de estrés, ya que el estado de apagado en una celda sitúa a ambos transistores PMOS en fase de recuperación. Como era de esperar, cuanto mayor es el número de subarrays apagados (k), mayor es la reducción de estrés. Comparado con Conv y DT0 (la configuración óptima de ASCAR), DT4 reduce el tiempo de estrés en al menos un 74 % y 25 %, respectivamente. Cabe señalar que la degradación por NBTI es uniforme y se vuelve independiente de la carga de trabajo, lo que mejora la predictibilidad del envejecimiento sin requerir un perfilado detallado de dicha carga.

Al proyectar el tiempo de estrés en la degradación de $\mathrm{V_{th}}$ ($\Delta \mathrm{V_{th}}$) para un tiempo de vida útil de 3 años [38] mediante el modelo de envejecimiento de la Ecuación 1, DT4 reduce la $\Delta \mathrm{V_{th}}$ en el peor caso en un 65 % y 15 % en comparación con Conv y ASCAR, respectivamente. Concretamente, el desplazamiento de $\mathrm{V_{th}}$ en el peor caso es de 58 mV para Conv, lo cual se considera habitualmente como un criterio de fallo del dispositivo [39], [40], mientras que dicha degradación de $\mathrm{V_{th}}$ es de solo 16 mV para DT4.

El truncamiento de datos implica un compromiso entre la mitigación del envejecimiento y la pérdida de precisión. La Figura 9 muestra el impacto en las métricas de precisión para HPatches, dependiendo de si el truncamiento de datos se aplica en el subsistema de memoria del módulo de detección, en los módulos de orientación y descripción o en ambos subsistemas.

Independientemente del subsistema de memoria, la métrica KR apenas se ve afectada por el truncamiento de datos, mientras que IM se mantiene por encima del 90 % de la precisión libre de fallos a pesar de truncar hasta 3 bits. Por otro lado, en orientación y descripción, la métrica KV se ve significativamente afectada por DT3 y DT4.

En general, todos los niveles de truncamiento de datos ofrecen un compromiso razonable entre la mitigación de envejecimiento y la pérdida de precisión para detección, mientras que DT1 y DT2 resultan más adecuados para orientación y descripción.

A pesar de que el truncamiento de datos agresivo afecta significativamente a KV, no compromete

la precisión global en la aplicación de SLAM. La Tabla II muestra el error absoluto de la trayectoria (ATE) promedio para todas las secuencias de los conjuntos de datos KITTI y EuRoC, considerando 100 muestras por secuencia.

En el caso de KITTI, el ATE aumenta con la agresividad del truncamiento de datos, incrementando el ATE promedio en 0,5 m para DT4. Sin embargo, en comparación con la distancia media de 2016,3 m recorrida por las secuencias, el error relativo es de tan solo un 0,025 %.

Por el contrario, EuRoC mejora ligeramente el ATE con el truncamiento de datos, lo cual es coherente con algunas secuencias de HPatches en escenarios con fallos (véase la Sección III-B).

VI. Trabajo Relacionado

Los trabajos previos que abordan el efecto NBTI en aceleradores hardware se han centrado en aplicaciones de redes neuronales profundas (DNN). Estos enfoques se basan en la inversión periódica de bits [41], técnicas de rotación de direcciones [42], cuantización del modelo para compensar el aumento de retardo inducido por el envejecimiento [43], *frameworks* de entrenamiento de modelos conscientes del envejecimiento [44] o mejoras en el flujo de diseño para corregir violaciones de tiempo [45].

Se han propuesto soluciones basadas en *power gating* consciente de NBTI para memorias integradas de GPUs de propósito general [46], [47] y aceleradores DNN [42]. Estos trabajos aplican *power gating* a regiones de memoria inactivas, aprovechando la infrautilización de memoria inherente de las aplicaciones que se ejecutan en dichos dispositivos. Sin embargo, la memoria integrada en los aceleradores ORB se utiliza por completo; por lo tanto, dichas estrategias previas no pueden aplicarse directamente en dispositivos ORB.

Finalmente, con el objetivo de reducir los recursos hardware, algunos aceleradores ORB truncan los valores de los cálculos intermedios de detección y orientación [5], [4], mientras que el acelerador FSLAM trunca los dos bits menos significativos del píxel tras el módulo de detección [7].

VII. Conclusiones

El envejecimiento inducido por NBTI degrada la tensión umbral (V_{th}) de los transistores de memoria a lo largo de la vida operativa del circuito, ralentizando la conmutación de los dispositivos y provocando, en última instancia, fallos funcionales cuando los retardos del camino crítico superan el periodo de reloj. Este trabajo ha presentado una caracterización de la fiabilidad frente al efecto NBTI en el subsistema de memoria de los aceleradores ORB y ha cuantificado su impacto en la precisión.

El análisis ha revelado que unos pocos fallos inducidos por NBTI en los bits más significativos afectan ampliamente a las métricas ORB, mientras que los fallos en los bits menos significativos son tolerados en gran medida por el algoritmo.

Aprovechando esta tolerancia intrínseca a los fallos, hemos propuesto una arquitectura de memoria eficiente y consciente de NBTI que combina una técnica de truncado de datos a través de *power gating*, específicamente diseñada para aceleradores ORB con técnicas de rotación e inversión del estado del arte.

Los resultados experimentales han demostrado que, en comparación con el diseño convencional y el estado del arte, la arquitectura propuesta reduce la degradación de V_{th} en el peor caso hasta en un 65 % y 16 %, respectivamente, con un impacto mínimo en la precisión y el área. Finalmente, el ahorro de consumo es de un 35 % respecto al diseño convencional.

Agradecimientos

Todos los autores reconocen el apoyo de las subvenciones: (i) PID2022-136454NB-C22/AEI/10.13039/501100011033 de la Agencia Estatal de Investigación (AEI), (ii) el grupo de investigación gaZ: T58_23R del Departamento de Ciencia, Universidad y Sociedad del Conocimiento del Gobierno de Aragón y (iii) el proyecto prueba de concepto RETORNNA MICIU/AEI/10.13039/501100011033 (grant PDC2023-145851-I00). Los financiadores no tuvieron ningún papel en el diseño del estudio, la recopilación y el análisis de datos, la decisión de publicar ni en la preparación del manuscrito.

Referencias

[1] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proceedings of the International Conference on Computer Vision*, 2011, pp. 2564–2571.

[2] J. Weberruss, L. Kleeman, and T. Drummond, "ORB feature extraction and matching in hardware," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2015.

[3] J. Weberruss, L. Kleeman, D. Boland, and T. Drummond, "FPGA acceleration of multilevel ORB feature extraction for computer vision," in *Proceedings of the 27th International Conference on Field Programmable Logic and Applications*, 2017, pp. 1–8.

[4] W. Fang, Y. Zhang, B. Yu, and S. Liu, "FPGA-based ORB feature extraction for real-time visual SLAM," in *Proceedings of the International Conference on Field Programmable Technology*, 2017, pp. 275–278.

[5] P. Tran, T. H. Pham, S. K. Lam, M. Wu, and B. A. Jasani, "Stream-Based ORB Feature Extractor with Dynamic Power Optimization," in *Proceedings of the International Conference on Field-Programmable Technology*, 2018, pp. 94–101.

[6] R. Liu, J. Yang, Y. Chen, and W. Zhao, "eSLAM: An Energy-Efficient Accelerator for Real-Time ORB-SLAM on FPGA Platform," in *Proceedings of the 56th ACM/IEEE Design Automation Conference*, 2019, pp. 1–6.

[7] V. Vemulapati and D. Chen, "FSLAM: an Efficient and Accurate SLAM Accelerator on SoC FPGAs," in *Proceedings of the International Conference on Field-Programmable Technology*, 2022, pp. 1–9.

[8] R. Taranco, J.-M. Arnau, and A. González, "LOCATOR: Low-power ORB accelerator for autonomous cars," *Journal of Parallel and Distributed Computing*, vol. 174, pp. 32–45, 2023.

[9] N. M. Belmessaoud, Y. Bentoutou, and M. Chikr El-Mezouar, "FPGA implementation of feature detection and matching using ORB," *Microprocessors and Microsystems*, vol. 94, 2022.

[10] A. Costa, J. D. Lopes, P. Tomás, N. Roma, and N. Neves, "Real-Time ORB Accelerator for Embedded FPGA-Based SoCs With ROS Integration," *IEEE Transactions*

on Very Large Scale Integration (VLSI) Systems, vol. 34, no. 1, pp. 61–70, 2026.

[11] Q. Zhang, H. Sun, Q. Deng, H. Yu, and Y. Ha, "NORB: A Stream-Based and Non-Blocking FPGA Accelerator for ORB Feature Extraction," in *Proceedings of the 30th IEEE International Conference on Electronics, Circuits and Systems*, 2023, pp. 1–4.

[12] R. Xue, W. Li, H. Wu, C. Guo, Y. Li, X. Ye, and D. Fan, "A High-Performance Dataflow-Based ORB Extractor Accelerator for SLAM," in *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design*, 2025, pp. 1–7.

[13] A. Bhattacharjee, A. Nag, K. Das, and S. N. Pradhan, "Design of Power Gated SRAM Cell for Reducing the NBTI Effect and Leakage Power Dissipation During the Hold Operation," *Journal of Electronic Testing*, vol. 38, pp. 91–105, 2022.

[14] V. Balntas, K. Lenc, A. Vedaldi, and K. Mikolajczyk, "HPatches: A Benchmark and Evaluation of Handcrafted and Learned Local Descriptors," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3852–3861.

[15] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[16] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999, vol. 2, pp. 1150–1157.

[17] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-Up Robust Features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[18] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," in *Proceedings of the 9th European Conference on Computer Vision*, 2006, pp. 430–443.

[19] P. L. Rosin, "Measuring Corner Properties," *Computer Vision and Image Understanding*, vol. 73, no. 2, pp. 291–307, 1999.

[20] A. Yazdanbakhsh, R. Balasubramanian, T. Nowatzki, and K. Sankaralingam, "Comprehensive Circuit Failure Prediction for Logic and SRAM Using Virtual Aging," *IEEE Micro*, vol. 35, no. 6, pp. 24–36, 2015.

[21] S. Ahmad, N. Alam, M. Hasan, and B.-S. Kong, "A Comprehensive Review of Design Challenges and Techniques for Nanoscale SRAM: A Cell Perspective," *TechRxiv*, 2022.

[22] S. V. Kumar, K. H. Kim, and S. S. Sapatnekar, "Impact of NBTI on SRAM read stability and design for reliability," in *Proceedings of the 7th International Symposium on Quality Electronic Design*, 2006, pp. 210–218.

[23] J. Abella, X. Vera, and A. Gonzalez, "Penelope: The NBTI-Aware Processor," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007, pp. 85–96.

[24] S. Kothawade, K. Chakraborty, and S. Roy, "Analysis and mitigation of NBTI aging in register file: An end-to-end approach," in *Proceedings of the 12th International Symposium on Quality Electronic Design*, 2011, pp. 1–7.

[25] M. R. Guthaus, J. E. Stine, S. Ataei, Brian C., Bin W., and M. Sarwar, "OpenRAM: An open-source memory compiler," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2016, pp. 1–6.

[26] NC State University, "FreePDK45 — NC State EDA," `https://eda.ncsu.edu/freepdk/freepdk45/`, 2024, Visited: 26-01-2026.

[27] A. Tiwari and J. Torrellas, "Facelift: Hiding and Slowing Down Aging in Multicores," in *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, 2008, pp. 129–140.

[28] R. Vattikonda, W. Wang, and Y. Cao, "Modeling and Minimization of PMOS NBTI Effect for Robust Nanometer Design," in *Proceedings of the 43rd Design Automation Conference*, 2006, pp. 1047–1052.

[29] F. Oboril and M. B. Tahoori, "ExtraTime: Modeling and Analysis of Wearout due to Transistor Aging at Microarchitecture-Level," in *Proceedings of the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2012, pp. 1–12.

[30] M. Shafique, M. U. K. Khan, O. Tüfek, and J. Henkel, "EnAAM: Energy-efficient anti-aging for on-chip video memories," in *Proceedings of the 52nd ACM/EDAC/IEEE Design Automation Conference*, 2015, pp. 1–6.

[31] M. Shafique, M. U. K. Khan, and J. Henkel, "Content-Aware Low-Power Configurable Aging Mitigation for SRAM Memories," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3617–3630, 2016.

[32] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power," in *Proceedings of the 28th Annual International Symposium on Computer Architecture*, 2001, pp. 240–251.

[33] A. Calimera, E. Macii, and M. Poncino, "Analysis of NBTI-Induced SNM Degradation in Power-Gated SRAM Cells," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2010, pp. 785–788.

[34] D. Bojanić, K. Bartol, T. Pribanić, T. Petković, Y. D. Donoso, and J. S. Mas, "On the Comparison of Classic and Deep Keypoint Detector and Descriptor Methods," in *Proceedings of the 11th International Symposium on Image and Signal Processing and Analysis*, 2019, pp. 64–69.

[35] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361.

[36] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

[37] K.-P. Niu, *Compact Area and Performance Modelling for Coarse-Grained Reconfigurable Architectures*, Ph.D. thesis, University of Toronto, 2019.

[38] E. Mintarno, V. Chandra, D. Pietromonaco, R. Aitken, and R. W. Dutton, "Workload dependent NBTI and PBTI analysis for a sub-45nm commercial microprocessor," in *Proceedings of the IEEE International Reliability Physics Symposium*, 2013, pp. 3A.1.1–3A.1.6.

[39] U. Dutta, M. Soni, and M. Pattanaik, "A Review of NBTI Degradation and its Impact on the Performance of SRAM," *International Journal of Modern Education and Computer Science*, vol. 8, pp. 57–65, 2016.

[40] D. K. Schroder and J. A. Babcock, "Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing," *Journal of Applied Physics*, vol. 94, pp. 1–18, 2003.

[41] M. A. Hanif and M. Shafique, "DNN-Life: An Energy-Efficient Aging Mitigation Framework for Improving the Lifetime of On-Chip Weight Memories in Deep Neural Network Hardware Architectures," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, 2021, pp. 729–734.

[42] N. Landeros Muñoz, A. Valero, R. Gran Tejero, and D. Zoni, "Gated-CNN: Combating NBTI and HCI aging effects in on-chip activation memories of Convolutional Neural Network accelerators," *Journal of Systems Architecture*, vol. 128, 2022.

[43] S. Salamin, G. Zervakis, O. Spantidi, I. Anagnostopoulos, J. Henkel, and H. Amrouch, "Reliability-Aware Quantization for Anti-Aging NPUs," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, 2021, pp. 1460–1465.

[44] S. Das, S. Kundu, A. Menon, Y. Ren, S. Kharel, and K. Basu, "Analyzing and Mitigating Circuit Aging Effects in Deep Learning Accelerators," in *Proceedings of the IEEE 42nd VLSI Test Symposium*, 2024, pp. 1–7.

[45] F. Ramadan, G. Shomron, and F. Gabbay, "The Effect of Asymmetric Transistor Aging on Systolic Arrays for Mission Critical Machine Learning Applications," *IEEE Access*, vol. 13, pp. 44041–44061, 2025.

[46] M. Namaki-Shoushtari, A. Rahimi, N. Dutt, P. Gupta, and R. K. Gupta, "ARGO: Aging-aware GPGPU Register File Allocation," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, 2013, pp. 1–9.

[47] A. Valero, F. Candel, D. Suárez-Gracia, S. Petit, and J. Sahuquillo, "An Aging-Aware GPU Register File Design Based on Data Redundancy," *IEEE Transactions on Computers*, vol. 68, no. 1, pp. 4–20, 2019.

Checkpointing transparente en aplicaciones que utilizan sistemas de ficheros ad-hoc

Dario Muñoz-Muñoz[1], Felix Garcia-Carballeira[1], Alejandro Calderon-Mateos[1], Diego Camarmas-Alonso[1] y Jesus Carretero[1]

Resumen— **El crecimiento exponencial de las cargas de trabajo intensivas en datos en la computación de alto rendimiento, particularmente en simulaciones científicas y big data, ha convertido a los sistemas de ficheros paralelos tradicionales en un cuello de botella de rendimiento significativo. Para abordar este desafío, los sistemas de ficheros paralelos ad-hoc han surgido como una alternativa de alto rendimiento, aprovechando los recursos locales de los nodos para proporcionar almacenamiento temporal de baja latencia. Sin embargo, la naturaleza efímera de estos sistemas introduce un desafío crítico de fiabilidad, ya que el almacenamiento está vinculado al ciclo de vida de la asignación del trabajo en los nodos de cómputo. En consecuencia, cualquier fallo de un nodo o la terminación de un trabajo resulta naturalmente en la pérdida total de los datos del sistema de ficheros ad-hoc. Además, las técnicas tradicionales de checkpointing son ineficaces para abordar este problema porque generalmente se centran en la memoria de la aplicación y no tienen en cuenta la posible pérdida de datos dentro del propio sistema de ficheros ad-hoc.**

Este trabajo aborda esta limitación presentando la integración de un flujo de trabajo de checkpointing transparente con el sistema de ficheros paralelos Expand Ad-Hoc. La solución propuesta utiliza MPI-Agnostic Network-Agnostic (MANA), proporcionando una solución unificada que guarda el estado de la memoria de la aplicación junto con los datos almacenados en la capa de almacenamiento ad-hoc. Este enfoque implica el almacenamiento directo de las imágenes de checkpoint en Expand Ad-Hoc, que proporciona tolerancia a fallos. Esto garantiza la integridad de los datos incluso en caso de fallo de un nodo individual. Adicionalmente, la integración emplea un mecanismo inteligente de preservación de estado que captura selectivamente solo los ficheros modificados durante la ejecución. Este mecanismo reduce eficazmente la redundancia de E/S y minimiza la sobrecarga para los ficheros de solo lectura.

Esta solución fue evaluada en el supercomputador C3 utilizando el benchmark IOR y Quantum ESPRESSO, una aplicación científica del mundo real. Los resultados experimentales demuestran que nuestra solución integrada reduce el tiempo de checkpointing entre un 44 % y un 83 % en comparación con los sistemas de ficheros backend tradicionales, en este caso utilizando Lustre. En escenarios con intervalos de checkpointing frecuentes, la implementación de Expand Ad-Hoc resulta en una reducción sustancial del tiempo total de ejecución, demostrando su eficacia como una solución fiable, transparente y de alto rendimiento para la computación a gran escala.

Palabras clave— **Checkpoint transparente, Checkpoint and restore, Expand Ad-Hoc, Sistema de ficheros paralelo, Tolerancia a fallos**

[1]Universidad Carlos III de Madrid, ROR: https://ror.org/03ths8210, Departamento de Informática, Avenida de la Universidad, 30 (Edificio Sabatini), 28911 Leganés (Madrid), España. e-mail: {damunozm, fgcarbal, acaldero, dcamarma, jcarrete}@inf.uc3m.es

I. Introducción

En los últimos años, el crecimiento exponencial de las cargas de trabajo intensivas en datos, como las simulaciones científicas y el análisis de big data, ha impulsado la demanda de entornos de Computación de Alto Rendimiento (HPC) con capacidades de E/S masivas. Dado que los sistemas de ficheros paralelos compartidos convencionales son incapaces de satisfacer estas demandas, resultando en considerables cuellos de botella en el rendimiento, han surgido los sistemas de ficheros paralelos ad-hoc como una alternativa de alto rendimiento. Al aprovechar los recursos de los nodos locales, estos sistemas ad-hoc proporcionan un almacenamiento temporal de baja latencia adaptado a las necesidades específicas de un trabajo.

Sin embargo, el alto rendimiento del almacenamiento ad-hoc tiene un coste en forma de volatilidad inherente. Dado que estos sistemas suelen estar vinculados al ciclo de vida de los nodos de cómputo o a la asignación del trabajo, cualquier fallo de hardware o la terminación del trabajo da lugar a la pérdida inmediata de todos los datos almacenados en la capa ad-hoc. Aunque el checkpointing es la técnica predominante para asegurar el progreso en HPC, los métodos tradicionales suelen ser insuficientes para este escenario. La mayoría de las herramientas existentes priorizan la preservación del estado de la memoria de la aplicación, ignorando los datos críticos almacenados en el almacenamiento transitorio ad-hoc. Además, el desacoplamiento de los mecanismos de checkpointing de la capa de almacenamiento subyacente suele dar lugar a una sobrecarga de E/S sustancial y a problemas de consistencia.

Para abordar estos desafíos, este trabajo utiliza Expand Ad-Hoc, un sistema de ficheros paralelo diseñado para proporcionar almacenamiento temporal de alto rendimiento con mecanismos internos de tolerancia a fallos mediante la replicación de datos y con maleabilidad [1] [2]. Aunque los estudios previos sobre Expand Ad-Hoc se han centrado en su escalabilidad y fiabilidad de almacenamiento, sigue existiendo una brecha crítica: la protección del estado integrado de la aplicación en ejecución junto con los datos que genera.

La contribución principal de este trabajo es la implementación de un mecanismo de checkpointing transparente integrado con el sistema de ficheros paralelos Expand Ad-Hoc. El desarrollo de este mecanismo se logra mediante la utilización y extensión de **MANA** (MPI-Agnostic Network-Agnostic

checkpointing) [3] [4] [5]. MANA es una librería que opera sobre **DMTCP** (Distributed Multi-Threaded Checkpointing) [6], proporcionando una base sólida para guardar de forma transparente el estado de aplicaciones MPI y no MPI. Se ha desarrollado una extensión interna especializada para MANA para cerrar la brecha entre el estado de ejecución de la aplicación y la capa de almacenamiento de Expand Ad-Hoc.

Esta integración permite un flujo de trabajo de checkpointing unificado en el que MANA captura la memoria del proceso y el estado de MPI, mientras que nuestra extensión garantiza que los datos de los ficheros y los descriptores asociados se sincronicen de forma consistente y se almacenen dentro de Expand Ad-Hoc. Al adoptar este enfoque, presentamos una solución que no solo protege la aplicación contra fallos de nodos, sino que también garantiza la integridad del estado del sistema de ficheros con respecto al punto de ejecución restaurado. Además, esta solución se logra sin necesidad de realizar alteraciones en el código fuente de las aplicaciones.

También se presenta una evaluación exhaustiva de esta solución integrada, centrada en la sobrecarga introducida por el proceso de checkpointing basado en MANA. Además, se presenta una comparación de rendimiento entre un sistema de ficheros paralelos backend tradicional, como Lustre [7], y la solución propuesta específicamente para almacenar los checkpoints de las aplicaciones. Estos experimentos se realizaron en el supercomputador C3 [8], utilizando el benchmark IOR [9] junto con Quantum ESPRESSO [10], una aplicación paralela del mundo real. Se adoptó este enfoque para demostrar la escalabilidad, eficiencia y fiabilidad del diseño.

Las secciones restantes del documento se estructuran de la siguiente manera: la Sección II analiza los trabajos relacionados sobre checkpointing transparente y E/S paralela. La Sección III detalla la integración y diseño de la extensión de MANA con Expand Ad-Hoc. La Sección IV presenta los resultados experimentales y el análisis de rendimiento. Por último, la Sección V resume las principales conclusiones y presenta las futuras direcciones de investigación.

II. Trabajos relacionados

El Checkpoint/Restart (C/R) sigue siendo la principal estrategia de tolerancia a fallos en la computación de alto rendimiento (HPC). Funciona guardando periódicamente el estado del proceso en un almacenamiento estable, permitiendo así la recuperación tras un fallo. La eficiencia de esta técnica está determinada por dos factores: la latencia del checkpoint, definida como el tiempo necesario para guardar un estado al almacenamiento, y el intervalo de checkpoint, que debe optimizarse para equilibrar la protección frente a la sobrecarga de ejecución. La implementación de los servicios de C/R puede dividirse en tres clasificaciones distintas según su nivel de transparencia e integración. La primera categoría es el C/R a nivel de aplicación, en el que el desarrollador se encarga manualmente de guardar el estado para minimizar el tamaño del archivo. La segunda categoría es el C/R a nivel de usuario, que utiliza librerías externas para automatizar el proceso con una optimización menos específica de la aplicación. La tercera categoría es el C/R a nivel de sistema, el enfoque más transparente, que captura el estado completo del proceso a través del sistema operativo o por software sin requerir modificaciones en el código fuente de la aplicación.

A. Checkpoint/Restart a nivel de usuario

En el dominio de las soluciones a nivel de usuario, SCR [11] destaca por su capacidad de aprovechar el almacenamiento local de los nodos para abordar los cuellos de botella de los sistemas de ficheros paralelos, aunque se enfrenta a desafíos debido a la volatilidad de los datos en caso de fallos totales de los nodos. Del mismo modo, VeloC [12], un backend de alto rendimiento preparado para la exaescala, ofrece un volcado (flushing) asíncrono eficiente. Aunque se ha demostrado que CRUISE [13] logra una alta eficiencia de E/S mediante la integración de un espacio de nombres sensible al checkpoint en los burst buffers, a menudo se ve limitado por su reducida portabilidad en arquitecturas de almacenamiento heterogéneas. Además, FTI [14] emplea un robusto enfoque de fiabilidad multinivel mediante la codificación Reed-Solomon, aunque esta resiliencia introduce una sobrecarga computacional adicional y requiere una integración más compleja en comparación con sus homólogos.

B. Checkpoint/Restart a nivel de sistema

En cuanto a las soluciones a nivel de sistema, CRIU [15] es ampliamente reconocido por su capacidad para congelar contenedores y procesos individuales aprovechando el sistema de ficheros `/proc` de Linux. Sin embargo, su principal limitación en HPC es la falta de un soporte nativo y fluido para aplicaciones MPI complejas y multinodo. Por otro lado, DMTCP [6] opera exclusivamente en el espacio de usuario sin requerir modificaciones del kernel, ofreciendo un marco altamente portátil y transparente para aplicaciones distribuidas. Sin embargo, a menudo se enfrenta a problemas de escalabilidad y a una sobrecarga significativa al gestionar metadatos a gran escala en redes de alto rendimiento. Para abordar estos desafíos específicos de MPI, MANA [3] [4] [5] introduce un enfoque de proceso dividido (split-process) que desacopla la aplicación de la librería MPI, logrando una estabilidad notable a través de diferentes capas de red. No obstante, esta sofisticada arquitectura introduce pequeñas penalizaciones de rendimiento durante la ejecución normal.

C. Checkpoint en sistemas de ficheros HPC

Tradicionalmente, las aplicaciones HPC han dependido de sistemas de ficheros paralelos globales como GPFS [16], Lustre [7] o BeeGFS [17] para almacenar los checkpoints. Aunque estos sistemas ofre-

cen almacenamiento compartido y alta capacidad, frecuentemente experimentan cuellos de botella de E/S significativos durante operaciones intensivas de checkpointing, donde las escrituras concurrentes desde numerosos nodos saturan el ancho de banda de almacenamiento.

Para abordar este desafío, la investigación reciente se ha desplazado hacia la implementación de estrategias de almacenamiento jerárquico y el uso del almacenamiento local en los nodos para funcionar como capas intermedias de alta velocidad. La aparición de sistemas de ficheros paralelos ad-hoc representa un avance sustancial en este dominio, aprovechando el almacenamiento local de los nodos para establecer entornos virtuales de almacenamiento de datos. Se han desarrollado diversas implementaciones, incluyendo BurstFS [18], GekkoFS [19], BeeOND [20] y Expand Ad-Hoc [2], para proporcionar una E/S localizada de alto ancho de banda que escala con el número de nodos. Al aprovechar estos recursos locales, es posible aislar el tráfico de checkpoint del sistema de ficheros backend. Esto, a su vez, reduce la latencia y evita que las operaciones de tolerancia a fallos degraden el rendimiento de la infraestructura compartida.

III. Diseño

A. Diseño de Expand Ad-Hoc

Expand Ad-Hoc [2] es un sistema de ficheros paralelo que ha sido optimizado para aplicaciones intensivas en datos en entornos HPC. Sigue un modelo cliente-servidor que utiliza sockets TCP POSIX, LFI[1] o MPI para la comunicación interna. La arquitectura se distingue por su notable flexibilidad, permitiendo el despliegue de servidores ad-hoc tanto en nodos dedicados como en conjunto con la aplicación. En este último caso, el sistema optimiza el rendimiento mediante la localidad de datos, permitiendo a los clientes omitir la comunicación de red y acceder al almacenamiento local directamente cuando los datos requeridos están presentes en el mismo nodo, como se muestra en la Figura 1.

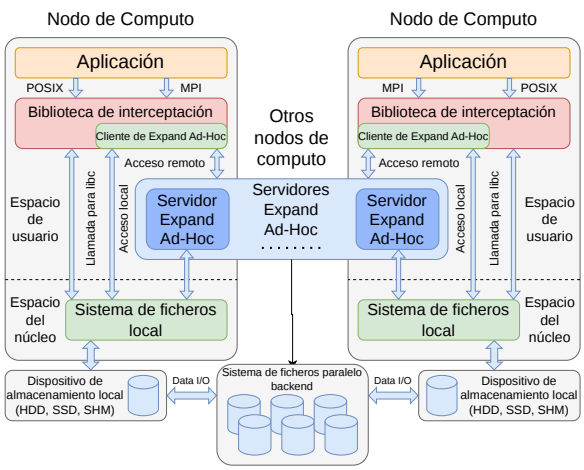


Fig. 1: Diseño de la arquitectura de Expand Ad-Hoc.

[1] https://github.com/dariomnz/lfi

La gestión de datos y metadatos en Expand Ad-Hoc está totalmente distribuida para evitar cuellos de botella. Los ficheros se distribuyen en todos los servidores disponibles en subficheros, que permanecen transparentes para el usuario. Para garantizar un uso eficiente de los recursos de almacenamiento, los subficheros se crean bajo demanda solo en los servidores necesarios para almacenar los bloques de datos. En consecuencia, si un servidor no aloja ninguna parte de un archivo, no se crea el subarchivo correspondiente, como se puede ver en la Figura 2. En lugar de utilizar servidores de metadatos dedicados, cada servidor ad-hoc gestiona de forma autónoma los metadatos en una cabecera reservada al principio de sus subficheros. Se utiliza un hash para establecer un servidor "maestro" para cada archivo y directorio. Esto garantiza búsquedas de metadatos eficientes y evita cuellos de botella.

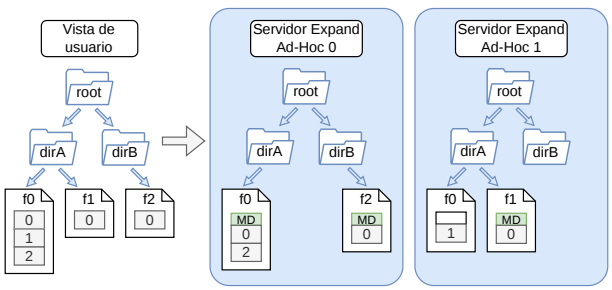


Fig. 2: Gestión de datos y metadatos en Expand Ad-Hoc.

Para garantizar la compatibilidad con el software existente, Expand Ad-Hoc incorpora una librería de interceptación de llamadas al sistema. El mecanismo `LD_PRELOAD` permite a la librería interceptar las llamadas de E/S POSIX estándar y redirigir aquellas destinadas a la partición de Expand al cliente ad-hoc, mientras pasa todas las demás llamadas a la librería estándar. Este enfoque garantiza una transparencia total, permitiendo que cualquier aplicación se beneficie del sistema de ficheros Expand Ad-Hoc sin ninguna modificación del código fuente.

La tolerancia a fallos se logra mediante una estrategia de replicación basada en bloques, en la que el sistema mantiene un número de copias definido por el usuario tanto para los datos como para los metadatos en diferentes servidores en un patrón de distribución round-robin, como se ilustra en la Figura 3. Este diseño garantiza que el sistema continúe funcionando en caso de fallos de nodos, siempre que al menos una réplica permanezca accesible. Para detectar tales fallos, el sistema puede utilizar LFI o sockets TCP POSIX, que proporcionan detección de errores durante la ejecución.

El mecanismo de replicación también se explota para mejorar el rendimiento de lectura. El cliente ad-hoc puede identificar la réplica disponible más cercana de un bloque de datos solicitado. Además, si una réplica está presente en el nodo de cómputo local, el sistema inicia una lectura local. Esta estrategia conduce a una reducción sustancial del tráfico de red y, en consecuencia, a una mejora en el rendimiento. Co-

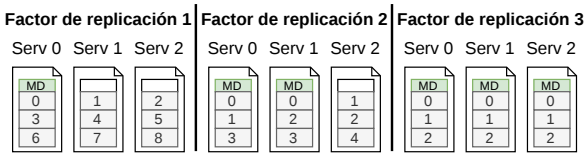


Fig. 3: Distribución de bloques de datos con factor de replicación 1, 2 y 3 en Expand Ad-Hoc.

mo resultado, se mejora la eficiencia del sistema de ficheros durante las operaciones intensivas de E/S.

B. DMTCP

Distributed MultiThreaded Checkpointing (DMTCP) [6] es una librería de checkpointing transparente diseñada para operar completamente en el espacio de usuario sin requerir modificaciones en el binario de la aplicación o en el kernel del sistema operativo subyacente. Para lograr este objetivo, el sistema emplea una librería diseñada para interceptar llamadas al sistema, facilitando la gestión del estado de las aplicaciones distribuidas en múltiples nodos. Esta transparencia es particularmente beneficiosa para trabajos de HPC de larga duración que requieren resiliencia frente a fallos de hardware o límites de tiempo de ejecución en clústeres de supercomputación.

La arquitectura de DMTCP se centra en un coordinador centralizado. Este coordinador sirve como mecanismo de control central, asegurando la sincronización de los checkpoints en todos los procesos de un cómputo distribuido. Cuando se activa un checkpoint, el coordinador se comunica con los clientes de DMTCP embebidos en cada proceso de la aplicación para pausar la ejecución, vaciar las colas de red y asegurar que se capture un estado global consistente, como se ilustra en la Figura 4. Este protocolo garantiza que la comunicación entre procesos siga siendo consistente tras la restauración.

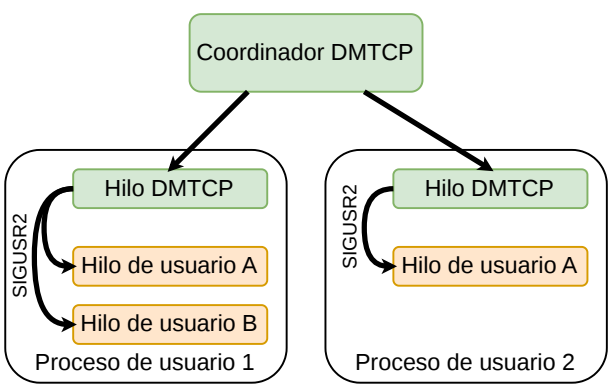


Fig. 4: Arquitectura de DMTCP.

El proceso de recuperación se ve facilitado por la generación de ficheros de checkpoint independientes para cada proceso. Estos ficheros contienen la huella de memoria completa y el contexto de ejecución, lo que permite que la aplicación se reinicie desde el punto exacto en el que se produjo el último checkpoint exitoso. Además, el diseño modular de DMTCP admite varios plugins, lo que permite a los investigadores ampliar su funcionalidad para manejar recursos especializados como la memoria de la GPU (vía CRAC [21]) o interconexiones de red de alta velocidad como InfiniBand [22], asegurando la compatibilidad con los entornos de computación heterogéneos modernos.

C. MANA

MPI-Agnostic Network-Agnostic (MANA) [3] es un marco de checkpointing especializado construido sobre la infraestructura de DMTCP. Fue diseñado para abordar la complejidad de la computación paralela basada en MPI. El objetivo principal de este enfoque es superar la dependencia rígida entre una aplicación y la implementación específica de MPI o la interconexión de red utilizada durante el tiempo de ejecución. Al lograr un estado agnóstico de la red, MANA garantiza que las aplicaciones de computación de alto rendimiento puedan guardarse y restaurarse en diferentes generaciones de hardware o de software sin perder el progreso.

La innovación fundamental de esta tecnología es el enfoque de proceso dividido, como se muestra en la Figura 5. En consecuencia, el espacio de direcciones de un proceso se divide en dos capas distintas: la "mitad superior", que contiene la aplicación del usuario y su memoria, y la "mitad inferior", que gestiona la librería MPI y los controladores de red específicos del sistema. Durante una operación de checkpoint, MANA preserva solo el estado relevante de la aplicación de la mitad superior. Este aislamiento evita que el checkpoint se vea afectado por estados de red temporales o identificadores específicos de hardware que, de otro modo, comprometerían el proceso de reinicio en recursos diferentes, haciéndolo imposible.

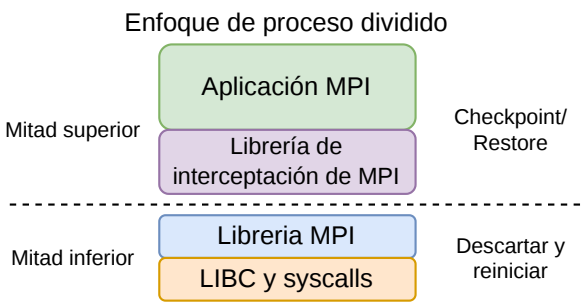


Fig. 5: Enfoque de proceso dividido de MANA.

Durante la fase de restauración, MANA inicializa una nueva mitad inferior, compuesta por una nueva librería MPI y pila de red, y vuelve a acoplar sin problemas la mitad superior guardada. Para mantener la consistencia, el marco utiliza una capa que intercepta las llamadas MPI, asegurando que todas las comunicaciones alcancen un "punto seguro" conocido y bien definido antes de que se inicie el checkpoint. Este mecanismo permite a MANA gestionar mensajes en tránsito y comunicaciones colectivas de forma transparente, proporcionando una solución robusta

para la resiliencia de trabajos a largo plazo en entornos de clúster heterogéneos.

D. Integración de Expand Ad-Hoc con MANA

La integración de Expand Ad-Hoc (XPN) con MANA está motivada principalmente por la necesidad de optimizar el flujo de checkpointing en entornos HPC a gran escala. Sin embargo, los métodos tradicionales de checkpointing suelen encontrar una sobrecarga de E/S y una contención sustancial cuando múltiples nodos escriben datos simultáneamente en un sistema de ficheros paralelo backend compartido. Al utilizar Expand Ad-Hoc para el almacenamiento de checkpoints, el sistema redistribuye esta carga. Sin embargo, utilizar almacenamiento ad-hoc para datos de recuperación críticos sería contraproducente en ausencia de garantías de fiabilidad robustas. En este sentido, XPN proporciona tolerancia a fallos integrada basada en la replicación de datos [2], lo que permite al sistema soportar fallos de nodos manteniendo la integridad de los datos. Esto garantiza la disponibilidad continua de los checkpoints, incluso en entornos inestables. En consecuencia, el enfoque utiliza los recursos de los nodos locales para optimizar el proceso de escritura, reduciendo así el tiempo total requerido para guardar el estado de la aplicación sin comprometer la fiabilidad.

Como se ilustra en la Figura 6, esta integración requiere una modificación estratégica de la arquitectura de proceso dividido de MANA. Para empezar, se desarrolló e integró una librería de interceptación dedicada a la interfaz Lightweight Fabric Interface (LFI) en la mitad inferior de MANA. LFI sirve como una librería de comunicación que permite el acceso a redes de alto rendimiento y baja latencia a través de la Open Fabric Interface (OFI). La integración de LFI dentro de la mitad inferior del sistema facilita la extensión de MANA, permitiendo así el reconocimiento y la gestión de esta capa de comunicación de alta velocidad. Esto garantiza que la capa de comunicación esté aislada de la mitad superior de la aplicación durante la ejecución del checkpoint.

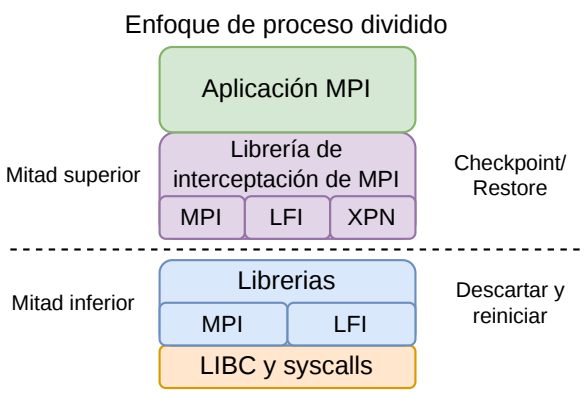


Fig. 6: Enfoque de proceso dividido de MANA con XPN.

Además, la arquitectura incorpora una librería de interceptación especializada para XPN, diseñada para interceptar las llamadas relevantes al sistema de ficheros. Esto permite redirigir las operaciones de E/S estándar al almacenamiento ad-hoc gestionado por Expand Ad-Hoc. Esta redirección de llamadas es transparente para la aplicación del usuario, lo que mantiene la filosofía no intrusiva de DMTCP y MANA al tiempo que se obtienen los beneficios de rendimiento de un checkpointing localizado de alta velocidad.

Un desafío crítico abordado durante esta integración es la estricta gestión de memoria requerida durante la fase de checkpointing. Para simplificar la explicación, DMTCP sigue un modelo en el que el estado del proceso se captura iterando secuencialmente sobre las regiones de memoria y escribiéndolas en el archivo de checkpoint. En este estado, cualquier modificación del heap del proceso, como llamadas a `malloc()` o `free()` dentro de las operaciones de `write` de XPN, podría alterar el mapa de memoria. Tales alteraciones darían lugar a un archivo de checkpoint corrupto, lo que imposibilitaría la restauración del proceso.

Para mitigar el riesgo mencionado anteriormente, las operaciones de escritura realizadas por XPN fueron reescritas meticulosamente para eliminar las asignaciones dinámicas de memoria. Adicionalmente, se implementó un mecanismo de seguridad mediante wrappers que monitorizan las llamadas a `malloc` y `free` específicamente durante la ventana de checkpointing. Si se detecta una asignación durante esta fase de una llamada a XPN, el programa falla intencionadamente y muestra un mensaje de error. Esto garantiza la integridad del estado capturado y asegura que el proceso de escritura secuencial permanezca determinista y seguro para futuras restauraciones al utilizar XPN en esta fase de checkpoint.

Además, la integración incorpora un mecanismo inteligente de preservación de estado para los datos que residen dentro del sistema de ficheros Ad-Hoc. Con el fin de reducir la redundancia de E/S, el sistema captura selectivamente los ficheros que han sufrido modificaciones desde el checkpoint anterior o el despliegue inicial. Los ficheros de solo lectura se excluyen del proceso de checkpointing, ya que se asume que estos se precargan durante la fase de inicio de Expand Ad-Hoc. En escenarios que implican intervalos de checkpointing frecuentes, si no se detectan cambios en el almacenamiento ad-hoc, el coste operativo se reduce efectivamente a cero. Esto se debe a que solo se incurre en una sobrecarga insignificante al realizar llamadas `stat()` para verificar los metadatos de los ficheros. Esta sincronización selectiva optimiza la eficiencia del checkpointing, evitando el movimiento innecesario de datos mientras mantiene el estado del almacenamiento consistentemente alineado con el progreso de la aplicación.

Normalmente, un trabajo que utiliza el sistema de ficheros Expand Ad-Hoc sigue un ciclo de vida estructurado que consta de las siguientes fases: precarga de datos, inicialización del servidor, ejecución de la aplicación y descarga de datos persistentes. La integración de nuestra implementación en este flujo

de trabajo implica un script de lanzamiento personalizado que diferencia automáticamente entre una ejecución nueva y un reinicio. Al monitorizar el directorio de checkpoints, el script detecta la presencia de imágenes existentes, lo que permite restaurar la aplicación. Adicionalmente, el sistema incorpora un mecanismo de seguridad de recuperación. En caso de que se detecte un fallo en los clientes de la aplicación, los servidores inician el respaldo de los datos de checkpoint en el sistema de ficheros paralelo backend. Esto garantiza que el usuario pueda volver a enviar el mismo trabajo y reanudar la ejecución sin problemas desde el último estado válido, manteniendo así la continuidad incluso a través de diferentes asignaciones de trabajos.

IV. Evaluación

Esta sección evalúa el rendimiento y la eficiencia del checkpointing transparente propuesto, integrado con el sistema de ficheros paralelos Expand Ad-Hoc. La evaluación se centra en dos aspectos críticos: la sobrecarga y la velocidad del propio proceso de checkpointing, y el impacto global en el tiempo de ejecución. Para este propósito, la solución propuesta se compara con el sistema de ficheros backend tradicional, Lustre.

Con el fin de evaluar las capacidades fundamentales de E/S de la integración, empleamos primero el **benchmark IOR** [9]. En este caso, IOR se utiliza como la aplicación de destino, y se mide el tiempo específico requerido para capturar y almacenar su estado durante una ejecución activa.

Además, para demostrar la eficacia de nuestra solución en un entorno de nivel de producción, utilizamos **Quantum ESPRESSO (PWscf v.7.5)** [10], una suite para cálculos mecánico-cuánticos de campo autoconsistente de ondas planas (PWscf) basada en la teoría funcional de la densidad. Se evalúa el tiempo total de ejecución de una simulación PWscf mientras se realizan checkpoints periódicos en intervalos de tiempo definidos. Esto permite evaluar la reducción en la sobrecarga de tiempo de ejecución proporcionada por la naturaleza de baja latencia del almacenamiento ad-hoc en comparación con los métodos convencionales.

En todas las evaluaciones realizadas, Expand Ad-Hoc se configuró por defecto y se utilizó el comunicador LFI.

La evaluación se lleva a cabo en el supercomputador C3 [8]. Las especificaciones técnicas de este supercomputador se presentan en la Tabla 1.

Tabla 1: Características del supercomputador C3.

Atributo	**Supercomputador C3** [8]
Número de nodos	90 nodos
Red	100Gbps HDR Infiniband
CPUs por nodo	2 x AMD EPYC 7713 128 núcleos totales, a 2.8 GHz
SSD por nodo	Dell Ent NVMe PM1733a RI 3.84TB
RAM por nodo	1024GB DDR4-3200
Sistema Operativo	Rocky Linux 8.10
Versión del Kernel	4.18.0-553
Distribución MPI	MPICH 4.3.0-ofi
Versión de GCC	12.2.0
Versión de SLURM	24.11.4
Sistema de ficheros	Lustre 2.15.7 16 OSTs en 2 nodos

A. Evaluación del tiempo de checkpoint

Para evaluar el rendimiento del checkpoint, se eligió el benchmark IOR. El objetivo de esta evaluación es determinar el impacto en el tiempo de checkpoint al comparar el sistema de ficheros backend (Lustre) frente al sistema de ficheros Expand Ad-Hoc (XPN).

Para realizar estas evaluaciones, se inicia el benchmark IOR y se solicita un checkpoint al coordinador después de 5 segundos de ejecución, y se mide el tiempo que se tarda en hacer el checkpoint. Esta evaluación se probó utilizando 8, 16 y 32 nodos de cómputo. Se distribuyeron 64, 128, 256, 512 y 1024 procesos IOR entre los nodos de cada evaluación. Los resultados se muestran en la Figura 7.

Para interpretar los resultados de la Figura 7 con precisión, es esencial señalar que cada proceso genera una imagen de checkpoint de 205 MiB. En consecuencia, el total de datos de checkpoint guardados es de aproximadamente 13 GiB, 26 GiB, 52 GiB, 105 GiB y 210 GiB para los 64, 128, 256, 512 y 1024 procesos, respectivamente.

Como se muestra en los resultados, Expand Ad-Hoc logra tiempos de checkpoint más bajos en todos los escenarios evaluados en comparación con el sistema de ficheros backend Lustre. Esto representa una reducción de tiempo sustancial que varía entre

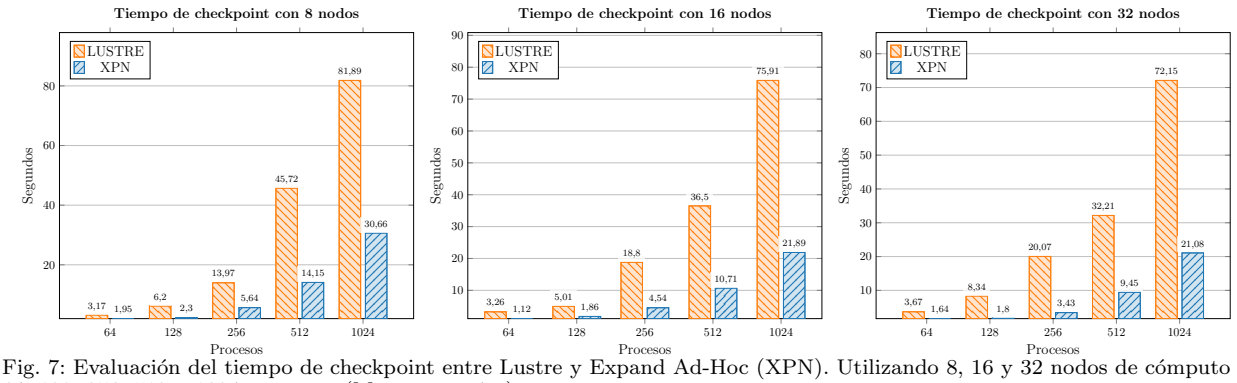


Fig. 7: Evaluación del tiempo de checkpoint entre Lustre y Expand Ad-Hoc (XPN). Utilizando 8, 16 y 32 nodos de cómputo y 64, 128, 256, 512 y 1024 procesos. (Menor es mejor)

el 44 % y el 83 %, demostrando la eficacia del sistema de ficheros Expand Ad-Hoc en comparación con Lustre.

B. Evaluación del tiempo de ejecución

Para evaluar el tiempo total de ejecución de la aplicación, utilizamos el algoritmo PWscf de la aplicación Quantum ESPRESSO. PWscf es un motor de teoría funcional de la densidad que emplea conjuntos de bases de ondas planas y pseudopotenciales para calcular propiedades del estado fundamental, optimizaciones estructurales y dinámica molecular para sistemas tanto metálicos como aislantes. La ejecución interna es iterativa, procediendo a través de múltiples ciclos para completar la simulación.

Esta evaluación consta de ocho casos distintos, con cuatro casos designados para cada sistema de ficheros, para comparar el rendimiento en diferentes escenarios. A continuación se detallan los casos, organizados secuencialmente desde la ejecución base hasta el escenario de recuperación completa:

- **Normal Lustre / XPN:** Representa la ejecución base de la aplicación, excluyendo el marco de checkpointing. Este caso se utiliza para medir el rendimiento nativo utilizando cada sistema de ficheros.
- **Checkpoint Lustre / XPN:** Evalúa el tiempo de ejecución, incluyendo la sobrecarga introducida por el marco de checkpointing. La aplicación guarda periódicamente su estado, pero no ocurren fallos durante la ejecución.
- **Restart Lustre / XPN:** Simula un escenario de fallo donde no existe un mecanismo de checkpointing. Tras el fallo, la aplicación debe reiniciarse desde su estado inicial, lo que representa la máxima pérdida de tiempo.
- **Restore Lustre / XPN:** Mide la eficiencia de la recuperación. Tras un fallo, la aplicación restaura su estado desde el checkpoint más reciente y reanuda la ejecución, destacando los beneficios del marco de tolerancia a fallos.

Se presentan dos enfoques: primero, un análisis detallado de una ejecución corta para resaltar las diferentes fases de la aplicación (Figura 8), y segundo, una ejecución de larga duración para evaluar el impacto de los errores simulados en el tiempo total de ejecución (Figuras 9, 10 y 11). Para estas evaluaciones, seleccionamos la configuración de rango medio de los experimentos anteriores: 16 nodos de cómputo y 256 procesos.

Para interpretar estos resultados con precisión, es esencial notar que cada proceso PWscf genera una imagen de checkpoint de 301 MiB. En consecuencia, el total de datos de checkpoint guardados para 256 procesos es de aproximadamente 77 GiB. También debe tenerse en cuenta que estos resultados reflejan el tiempo de ejecución del trabajo (job) y no incluyen el tiempo de espera en cola requerido para que el planificador asigne un nuevo trabajo después de un fallo.

En el primer enfoque (Figura 8), se simula un error en la marca de los 225 segundos. Debido a la naturaleza de los sistemas de ficheros Ad-Hoc, los datos deben ser cargados (stage-in) y descargados (stage-out). En consecuencia, la figura incluye una sección etiquetada como “XPN data I/O”, que representa la carga inicial de datos. En el caso de la restauración con XPN, se observa una duración mayor porque los datos del checkpoint deben guardarse en el sistema backend para asegurar que estén disponibles para el trabajo posterior. En este escenario, el intervalo de checkpoint se establece en 60 segundos. La figura también distingue entre el tiempo de ejecución de la aplicación, la sobrecarga de checkpointing y la fase de ejecución precedente al error.

Como se muestra en la Figura 8, los tiempos de ejecución base para ambos sistemas de ficheros son casi idénticos, al igual que los escenarios de reini-

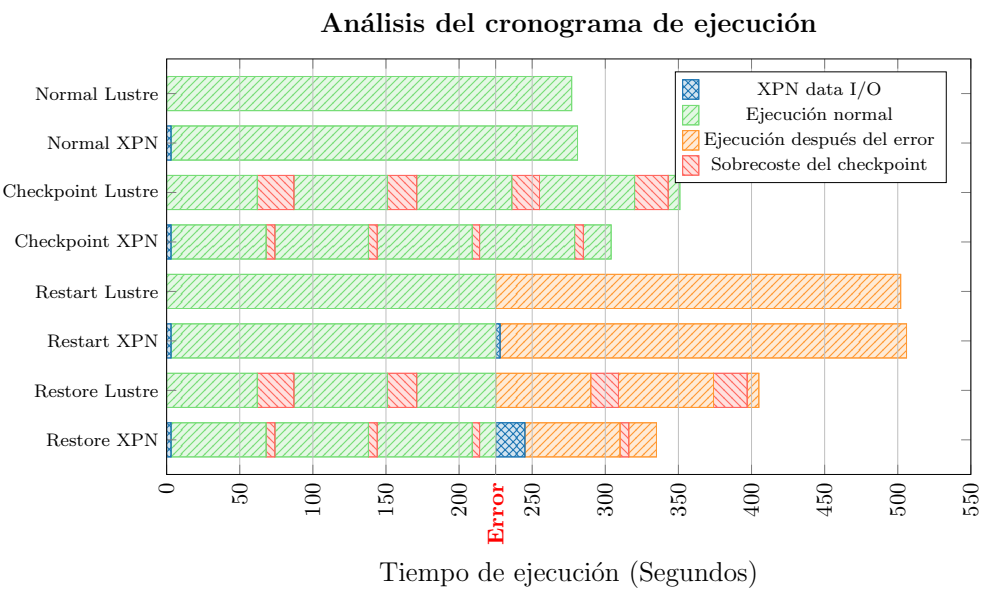


Fig. 8: Evaluación del tiempo de ejecución entre Lustre y Expand Ad-Hoc (XPN). Utilizando 16 nodos de cómputo y 256 procesos. Error a los 225 segundos. (Menor es mejor)

cio desde cero. Sin embargo, la diferencia se vuelve notable cuando el checkpointing está activo. Debido a la menor sobrecarga de checkpointing de XPN, el tiempo total de ejecución se reduce en un 13 % en comparación con Lustre. A medida que el tiempo de ejecución de la aplicación aumenta y se toman más checkpoints, se espera que esta brecha de rendimiento se amplíe, como se demuestra en el siguiente enfoque. Además, aunque XPN requiere tiempo extra para guardar el checkpoint en el backend durante un fallo para asegurar la persistencia de los datos, el tiempo total sigue siendo un 17 % menor que en el caso de Lustre. Esto se debe a que el tiempo acumulado ahorrado durante las fases periódicas de checkpointing compensa la sobrecarga del paso final de persistencia.

En el segundo enfoque, que evalúa el tiempo total de ejecución, se presentan tres escenarios en las Figuras 9, 10 y 11. Se simulan fallos al 25 %, 50 % y 75 % del tiempo total de ejecución base, que se establece en 10 horas. En consecuencia, los errores ocurren en las marcas de 2.5, 5 y 7.5 horas, respectivamente. Para esta evaluación, el intervalo de checkpoint se establece en 5 minutos, lo que resulta en 120 checkpoints a lo largo de todo el período de ejecución.

Como se observa en los tres casos, el tiempo total de ejecución sin errores se mantiene consistente, logrando Expand Ad-Hoc un ahorro de tiempo de 34 minutos en comparación con el sistema de ficheros backend Lustre.

Al comparar el ahorro de tiempo en escenarios de fallo, XPN supera a Lustre. Como se muestra en la Figura 9, con un error en la marca del 25 %, XPN ahorra 2.3 horas mientras que Lustre ahorra 1.7 horas. En el caso de fallo al 50 % (Figura 10), XPN ahorra 4.8 horas frente a las 4.1 horas de Lustre. Finalmente, para un error en la marca del 75 % (Figura 11), XPN ahorra 7.3 horas, mientras que Lustre ahorra 6.6 horas. Estos resultados confirman que, a medida que el error ocurre más tarde en la ejecución, el tiempo ahorrado por el enfoque de checkpointing aumenta como se esperaba, manteniendo XPN una ventaja significativa.

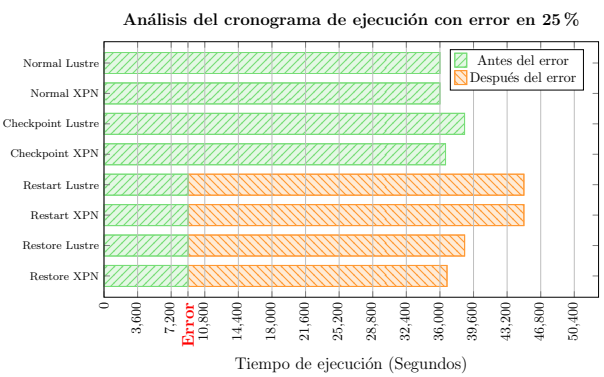


Fig. 9: Evaluación del tiempo de ejecución entre Lustre y Expand Ad-Hoc (XPN). Utilizando 16 nodos de cómputo y 256 procesos. Error al 25 %. (Menor es mejor)

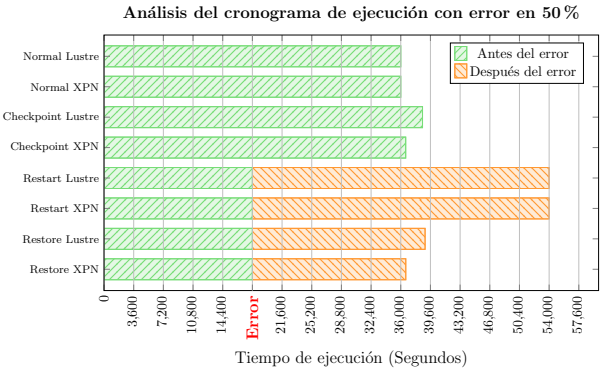


Fig. 10: Evaluación del tiempo de ejecución entre Lustre y Expand Ad-Hoc (XPN). Utilizando 16 nodos de cómputo y 256 procesos. Error al 50 %. (Menor es mejor)

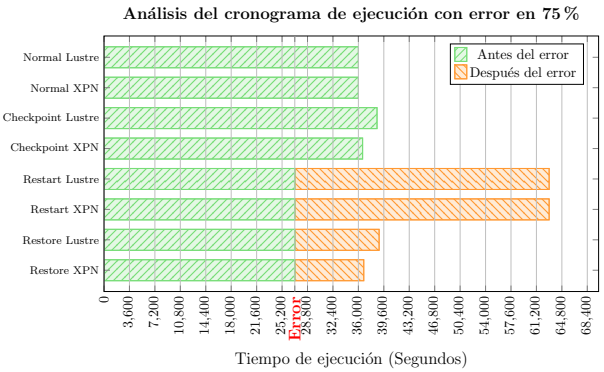


Fig. 11: Evaluación del tiempo de ejecución entre Lustre y Expand Ad-Hoc (XPN). Utilizando 16 nodos de cómputo y 256 procesos. Error al 75 %. (Menor es mejor)

V. Conclusión y trabajo futuro

Este trabajo elimina la brecha entre el almacenamiento ad-hoc de alto rendimiento y la tolerancia a fallos a nivel de aplicación. Aunque los sistemas de ficheros ad-hoc como Expand Ad-Hoc ofrecen un rendimiento de E/S superior, su naturaleza efímera ha planteado tradicionalmente un riesgo para las simulaciones de larga duración. La integración de Expand Ad-Hoc con la librería de checkpointing transparente MANA ha permitido el desarrollo de una solución unificada que asegura tanto el estado de ejecución como los datos de los ficheros. Esta integración proporciona un mecanismo de seguridad robusto y no intrusivo para las aplicaciones de HPC, garantizando que las ganancias de rendimiento del almacenamiento ad-hoc no se produzcan a expensas de la fiabilidad.

Las evaluaciones experimentales realizadas en el supercomputador C3 demuestran que la solución presentada supera significativamente al almacenamiento backend tradicional. Utilizando el benchmark IOR, observamos una reducción en el tiempo de checkpointing de entre el 44 % y el 83 % en comparación con el sistema de ficheros Lustre. Además, las pruebas con la aplicación del mundo real Quantum ESPRESSO mostraron que la menor sobrecarga de E/S de XPN resulta en una reducción del 13 % en el tiempo total de ejecución para ejecuciones cortas. Esta brecha se amplía a medida que disminuyen los intervalos de checkpoint. Incluso en escenarios de fallo donde los datos deben persistir en el backend, el tiempo acumulado ahorrado durante el checkpointing periódico

garantiza que el tiempo total de ejecución siga siendo inferior al de los métodos tradicionales.

Como trabajo futuro, pretendemos explorar mecanismos de guardado asíncrono para ocultar aún más la latencia de la persistencia de datos en el sistema de ficheros backend durante la recuperación de fallos. También proponemos evaluar la escalabilidad de esta integración en entornos de escala aún mayor con interconexiones heterogéneas. Por último, proponemos estudiar la integración de algoritmos de compresión adaptativos en el flujo de trabajo de checkpointing para reducir aún más la huella de almacenamiento y la presión de E/S en la capa ad-hoc, evaluando la comparación entre la sobrecarga de CPU de la compresión en tiempo real y la reducción en el tiempo de transferencia de datos.

Agradecimientos

Este trabajo ha sido financiado parcialmente por el proyecto de la Agencia Estatal de Investigación "New scalable I/O techniques for hybrid HPC and data-intensive workloads (SCIOT)" (Ref. PID2022-138050NB-I00) financiado por MCIN/AEI/10.13039/501100011033 y por el "FERDF A way of making Europe", y financiado parcialmente por el proyecto "Expand - High performance storage system for HPC and Big Data environments" (Ref. TED2021-131798B-I00) financiado por MICIU/AEI/10.13039/501100011033 y por la "European Union NextGenerationEU/PRTR".

Los experimentos de este trabajo se han realizado con el clúster de computación del "Centro para análisis y modelado de sistemas complejos en ingeniería y biomedicina" de la Universidad Carlos III de Madrid, financiado por el "Programa Estatal para Impulsar la Investigación Científico-Técnica y su Transferencia" con referencia EQC2021-007184-P.

Referencias

[1] Diego Camarmas-Alonso, Felix Garcia-Carballeira, Alejandro Calderon-Mateos, and Jesus Carretero, "Improving i/o performance in hpc environments using the expand ad-hoc file system," *The Journal of Supercomputing*, vol. 81, no. 16, pp. 1541, Nov 2025.

[2] Dario Muñoz-Muñoz, Felix Garcia-Carballeira, Diego Camarmas-Alonso, Alejandro Calderon-Mateos, and Jesus Carretero, "Malleability and fault tolerance in ad-hoc parallel file systems," *Cluster Computing*, vol. 28, no. 13, pp. 860, Sep 2025.

[3] Rohan Garg, Gregory Price, and Gene Cooperman, "Mana for mpi: Mpi-agnostic network-agnostic transparent checkpointing," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, New York, NY, USA, 2019, HPDC '19, p. 49–60, Association for Computing Machinery.

[4] Yao Xu, Leonid Belyaev, Twinkle Jain, Derek Schafer, Anthony Skjellum, and Gene Cooperman, "Implementation-oblivious transparent checkpoint-restart for mpi," in *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, New York, NY, USA, 2023, SC-W '23, p. 1738–1747, Association for Computing Machinery.

[5] Yao Xu and Gene Cooperman, "Enabling practical transparent checkpointing for mpi: A topological sort approach," 2024.

[6] Jason Ansel, Kapil Arya, and Gene Cooperman, "DMTCP: Transparent checkpointing for cluster computations and the desktop," in *2009 IEEE International Symposium on Parallel & Distributed Processing (IPDPS'09)*, Rome, Italy, 2009, IEEE, pp. 1–12, IEEE.

[7] Peter Braam, "The lustre storage architecture," *CoRR*, vol. abs/1903.01955, 2019.

[8] uc3m, "C3 supercomputer web," 2026, Accessed January. 9, 2026. [Online].

[9] HPC, "Github repository," 2024.

[10] Paolo Giannozzi et al., "QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials," *Journal of Physics: Condensed Matter*, vol. 21, no. 39, pp. 395502, sep 2009.

[11] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R. de Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, USA, 2010, SC '10, p. 1–11, IEEE Computer Society.

[12] Bogdan Nicolae, Adam Moody, Elsa Gonsiorowski, Kathryn Mohror, and Franck Cappello, "Veloc: Towards high performance adaptive asynchronous checkpointing at large scale," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Rio de Janeiro, Brazil, May 2019, pp. 911–920, IEEE.

[13] Raghunath Rajachandrasekar, Adam Moody, Kathryn Mohror, and Dhabaleswar K. (DK) Panda, "A 1 pb/s file system to checkpoint three million mpi tasks," in *Proceedings of the 22nd International Symposium on High-Performance Parallel and Distributed Computing*, New York, NY, USA, 2018, HPDC '13, p. 143–154, Association for Computing Machinery.

[14] Leonardo Bautista-Gomez, Seiji Tsuboi, Dimitri Komatitsch, Franck Cappello, Naoya Maruyama, and Satoshi Matsuoka, "Fti: high performance fault tolerance interface for hybrid systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2011, SC '11, Association for Computing Machinery.

[15] CRIU, "Checkpoint/restore in userspace (criu)," 2026, Accessed January. 9, 2026. [Online].

[16] Frank Schmuck and Roger Haskin, "Gpfs: a shared-disk file system for large computing clusters," in *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, USA, 2002, FAST'02, p. 16, USENIX Association.

[17] Frank Herold, Sven Breuner, and Jan Heichler, "An introduction to beegfs," 2014.

[18] Teng Wang, Kathryn Mohror, Adam Moody, Kento Sato, and Weikuan Yu, "An ephemeral burst-buffer file system for scientific applications," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, UT, USA, Nov 2016, pp. 807–818, IEEE.

[19] Marc-André Vef, Nafiseh Moti, Tim Süß, Markus Tacke, Tommaso Tocci, Ramon Nou, Alberto Miranda, Toni Cortes, and André Brinkmann, "Gekkofs—a temporary burst buffer file system for hpc applications," *Journal of Computer Science and Technology*, vol. 35, pp. 72–91, 2020.

[20] Zoya Masih, "On demand file systems with beegfs," Tech. Rep. 19034321, Georg-August-Universität Göttingen, 2023.

[21] Twinkle Jain and Gene Cooperman, "CRAC: checkpoint-restart architecture for CUDA with streams and UVM," *CoRR*, vol. abs/2008.10596, 2020.

[22] Jiajun Cao, Kapil Arya, and Gene Cooperman, "Transparent checkpoint-restart over infiniband," *CoRR*, vol. abs/1312.3938, 2013.

Simulación de procesamiento en memoria de GeMV y GeMM con LPDDR5

Jaime Martínez-Fernández[1], Juan Gómez-Luna[2], José L. Abellán[1] y Manuel E. Acacio[1]

Resumen— **El rendimiento de las aplicaciones limitadas por memoria está determinado principalmente por el tiempo empleado en el movimiento de datos entre la memoria principal y la unidad de procesamiento (CPU o GPU). Para conseguir ejecutar este tipo de cargas de trabajo de manera eficiente, se ha propuesto el paradigma de procesamiento en memoria (*Processing-In-Memory* o PIM). Más concretamente, se trata de dotar a la memoria DRAM de cierta capacidad de cómputo, pudiendo operar directamente sobre los datos en memoria y evitando así su transferencia al procesador.**

En particular, la multiplicación matriz-vector (GeMV) es una operación inherentemente limitada por memoria debido a la escasa reutilización de datos. Existen varias arquitecturas PIM capaces de ejecutar este tipo de cargas de trabajo. En concreto, Newton ejecuta eficientemente cargas de trabajo GeMV sobre memorias HBM. Sobre esta base, PIMFlow emplea dicha arquitectura para procesar redes neuronales convolucionales (*Convolutional Neural Networks* o CNNs) en un sistema GPU+PIM utilizando GDDR6.

En este contexto, tomamos como base la herramienta utilizada para evaluar PIMFlow e implementamos PIM en memoria LPDDR5, ampliamente utilizada en dispositivos con restricciones energéticas. Además, añadimos soporte para operaciones GeMM junto con un módulo de validación funcional. Este módulo no solo verifica la corrección de los resultados, sino que también permite obtener estadísticas detalladas del comportamiento de la arquitectura. Nuestros resultados muestran que PIM supera a una GPU orientada al *edge* en operaciones GeMV y también ofrece mejoras de rendimiento en GeMMs de pequeño tamaño.

Palabras clave— **GeMV, GeMM, PIM, LPDDR5, Newton, PIMFlow**

I. Introducción

En la actualidad existen multitud de cargas de trabajo incapaces de aprovechar la capacidad de cómputo de las distintas unidades de procesamiento disponibles (CPU y GPU). Esto se debe a que invierten un mayor tiempo moviendo datos entre la memoria principal y la unidad de procesamiento correspondiente, que propiamente computando con dichos datos. La operación de multiplicación matriz-vector (GeMV) es un ejemplo de carga de trabajo limitada por memoria, pues cada elemento de la matriz es leído de memoria y operado una única vez, sin presentar ningún tipo de reuso.

La operación GeMV forma una parte fundamental de aplicaciones como los modelos de lenguaje a gran escala (*Large Language Models* o LLMs). Otro ejemplo de operación limitada por memoria es la multiplicación de matrices donde una de las matrices es dispersa (SpMM). En este caso, el hecho de que la matriz dispersa presente un gran número de ceros hace que el acceso a sus datos sea muy irregular.

Para acelerar el procesamiento de este tipo de cargas de trabajo limitadas por memoria, se suele hacer uso de la técnica de procesamiento en memoria (*Processing-In-Memory* o PIM) [1]. Esta consiste en dotar a la memoria DRAM del sistema con una cierta capacidad de cómputo, evitando así tener que mover datos de DRAM a la unidad de procesamiento, al poder ser estos operados directamente en memoria. De esta manera, se consigue eficiencia tanto a nivel de tiempo de ejecución como a nivel de consumo energético.

Si bien existen en la actualidad propuestas que utilizan PIM para acelerar diversos ámbitos de aplicación, como por ejemplo, LLMs [2], [3] o redes neuronales de grafos (*Graph Neural Networks* o GNNs) [4], [5], [6], la eficiencia energética que ofrece PIM resulta especialmente útil para dar servicio a aplicaciones que demandan un bajo consumo energético, como la inferencia en el propio dispositivo para LLMs [7]. Este último tipo de aplicaciones suele estar ligado a la tecnología de memoria *Low Power Double Data Rate 5* (LPDDR5), que ofrece un bajo consumo energético a costa de un menor ancho de banda, y será al que prestemos atención en este trabajo.

Por otro lado, aunque podemos encontrar a día de hoy varias herramientas de simulación de arquitecturas DRAM [8], [9], hasta donde sabemos, ninguna ofrece soporte PIM para cargas de trabajo GeMV o similares sobre la tecnología de memoria LPDDR5. Es por ello que, en este trabajo, nos centramos en extender el simulador de memoria Ramulator [10], de manera que seamos capaces de ejecutar cargas de trabajo GeMV y GeMM (esta última, si bien no es limitada por memoria, resulta de gran interés como punto de partida para posteriormente soportar SpMM) en LPDDR5 utilizando la técnica de procesamiento en memoria.

Para conseguirlo, hemos partido de la herramienta utilizada para evaluar PIMFlow [11]. En PIMFlow aceleran la ejecución de redes neuronales convolucionales (*Convolutional Neural Networks* o CNNs) con PIM en un sistema GPU, ofreciendo diferentes estrategias de paralelismo entre la GPU y PIM. En cuanto a PIM, se considera una implementación en Ramulator para GDDR6. En concreto, implementan la arquitectura Newton [12], orientada a acelerar cargas GeMV en tecnologías de memoria HBM. Es importante remarcar que la implementación de Newton en PIMFlow está basada en un modelo analítico, es decir, únicamente modelan las latencias de los diferentes comandos PIM. Para la simulación de GPU

[1]Dpto. de Ingeniería y Tecnología de Computadores, Universidad de Murcia, e-mail: {jaime.m.f1,jlabellan,meacacio}@um.es.
[2]NVIDIA, e-mail: jgomezluna@nvidia.com.

utilizan Accel-Sim [13], una reconocida herramienta de simulación que modela arquitecturas GPU de NVIDIA.

Por lo tanto, las contribuciones de este trabajo son las siguientes:

- Sobre la herramienta de evaluación de PIMFlow, en esencia, Ramulator, hemos añadido soporte para la tecnología de memoria LPDDR5, ya que esta herramienta de simulación únicamente soporta hasta la generación 4 de LPDDR.
- Hemos añadido soporte para GeMV en LPDDR5. Además, aprovechando la representación de capas convolucionales mediante multiplicación de matrices (im2col), somos capaces de reutilizar la arquitectura Newton para ejecutar cargas de trabajo GeMM.
- Hemos desarrollado desde cero un módulo de validación funcional que asegura que el resultado del cómputo realizado con PIM es correcto. Además, esto nos permite generar estadísticas sobre el uso de la arquitectura PIM (uso de los bancos PIM, balanceo de carga entre bancos de un canal,...).

II. Estado del arte

A. Newton

Newton [12] es una arquitectura PIM desarrollada por el fabricante de semiconductores de chips de memoria SK Hynix, en colaboración con investigadores de Purdue University. Su objetivo es acelerar operaciones GeMV sobre la tecnología de memoria HBM. Para ello, sitúan la lógica de procesamiento a nivel de banco de memoria. La Figura 1 muestra el diseño de la arquitectura correspondiente a un banco de memoria. Para simplificar la explicación, asumimos un canal y un *rank* con 16 bancos. Las unidades de cómputo consisten en 16 multiplicadores y un árbol de reducción. Tras el árbol de reducción, existe un registro de salida en el que se acumulan los resultados obtenidos de las operaciones de multiplicación-acumulación (*Multiply–ACcumulates* o MACs). La fuente que nutre a estas operaciones es, por un lado, los datos almacenados en el banco de memoria y, por otro, los datos contenidos en el *buffer* global. El primero almacena elementos de la matriz, mientras que el segundo almacena elementos del vector y es compartido por todos los bancos de un canal.

De esta forma, el flujo de multiplicación matriz-vector es el expuesto en la Figura 2. En este caso, suponemos un *buffer* global con capacidad para 1 KB (512 elementos FP16), una matriz con 512 filas y 1536 columnas, y un vector con 1536 elementos. Puesto que las dimensiones del vector son mayores que la capacidad del *buffer* global, este se opera en *tiles* de 1 KB. Así, se mantiene estacionario un *tile* del vector en el *buffer* global y se realizan todas las multiplicaciones y acumulaciones con los *tiles* correspondientes de la matriz. Una vez completadas las operaciones con el *tile* actual, se copia al *buffer* global el siguiente *tile* del vector.

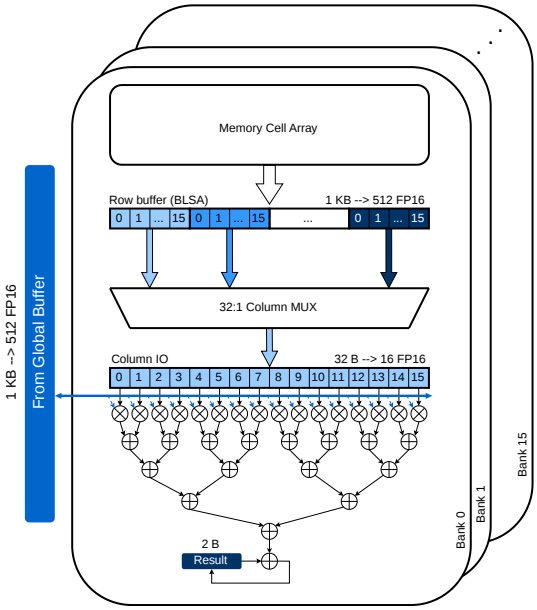


Fig. 1: Arquitectura de Newton. Se muestra con detalle la organización de uno de los bancos.

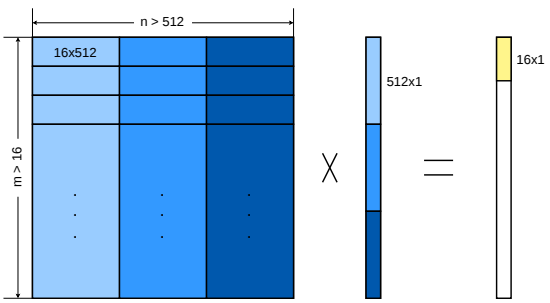


Fig. 2: *Data tiling* con 16 bancos y un tamaño de fila de DRAM de 1 KB.

El hecho de que el vector se mantenga estacionario y se opere completamente con cada *tile* consigue que los elementos del vector sean leídos una única vez. Sin embargo, esta manera de operar matriz-vector hace que las MACs de cada banco produzcan resultados parciales que deben ser acumulados para obtener el resultado final. Este procesamiento se hace en el *host*.

Newton añade 4 comandos PIM (Tabla I). *GWRITE* se encarga de cargar en el *buffer* global un *tile* del vector. Esta operación es realizada por el *host*. Con *G_ACT* se activa una fila en 4 bancos (debido a la restricción temporal tFAW[1]). En concreto, la fila de cada banco activado es escrita en el *row buffer*, también conocido como *Bit Line Sense Amplifier* o BLSA (Figura 1). Una vez los datos están preparados, el comando *COMP* se encarga de computarlos. Debido a la existencia de 16 MACs, cada comando *COMP* opera con 16 elementos (16 del *buffer* global y 16 del *row buffer*). Una característica importante de Newton es que un único comando *COMP* afecta a todos los bancos del canal. Por tanto, suponiendo FP16, serán necesarios 32 comandos *COMP* para

[1]Por motivos de potencia, las activaciones de filas DRAM se verían limitadas por una sincronización de cuatro activaciones (fFAW). Estas restricciones implican que las activaciones de filas se prolongarían durante un largo periodo de tiempo. Como solución, Newton agrupa 4 activaciones en un único comando.

Tabla I: Comandos PIM en Newton.

Comando	Funcionamiento
GWRITE	El *host* escribe en el *buffer* global un *tile* del vector.
G_ACT	Activa una fila en 4 bancos.
COMP	Opera en todos los bancos con 16 elementos.
READRES	Lee el valor del registro resultado de todos los bancos.

operar completamente con los datos del *row buffer*. Finalmente, el comando *READRES* se encarga de leer el resultado parcial almacenado en el registro resultado (*Result* en la Figura 1) y enviarlo al *host* para que realice la acumulación final.

Como hemos comentado, la explicación expuesta supone un canal. La configuración final de Newton consiste en 8 *stacks* con 8 canales, 2 pseudocanales (equivalente a 1 canal tradicional dividido en 2 para mayor paralelismo) y 16 bancos por canal. Esto se consigue replicando las operaciones explicadas en cada canal de forma paralela. La configuración concreta que nosotros utilizamos en los experimentos está incluida en la Sección IV.

B. PIMFlow

Hasta donde sabemos, PIMFlow [11] es el único artículo que añade soporte PIM para ejecutar operaciones GeMV en memoria y que tiene la herramienta de evaluación disponible. Es por ello que este artículo es de gran valor para nuestro trabajo, pues sirve de punto de partida para alcanzar nuestro objetivo.

PIMFlow propone una manera de acelerar la inferencia de CNNs permitiendo su ejecución en una memoria de GPU compatible con PIM. De manera muy general, su funcionamiento consiste en transformar los grafos de los modelos CNN para crear paralelismo entre nodos en GPU y PIM, considerando diferentes técnicas de paralelismo (de tareas y de datos). Puesto que el ámbito de nuestro trabajo se centra en procesamiento en memoria, vamos a centrar la explicación de la propuesta de PIMFlow en este aspecto.

Para ejecutar parte del procesamiento de modelos CNN, la herramienta usada para modelar y evaluar PIMFlow contiene una versión modificada de Ramulator. Como tecnología de memoria, se asume GDDR6. Las modificaciones que realizan sobre Ramulator consisten en, en primer lugar, añadir soporte a GDDR6 (Ramulator no implementa esta tecnología de memoria). Tras ello, implementan las operaciones de Newton (Tabla I) según un modelo analítico que les permite conocer la latencia de cada una de ellas. Para nutrir a Ramulator con estos comandos, PIMFlow genera una traza PIM a partir de los parámetros de una CNN (tamaño de la entrada, de los filtros, *padding*...). Esto, junto con las latencias de los comandos, permite a PIMFlow generar un tiempo de ejecución en PIM de una operación convolucional. Remarcar que, al simular PIM, PIMFlow no utiliza datos reales ni realiza ninguna comprobación sobre la corrección funcional de la salida. En su lugar, se basan exclusivamente en un modelo analítico con el que ofrecen un tiempo de ejecución en PIM.

El hecho de que una operación convolucional pueda ser expresada como multiplicación de matrices (GeMM) (a través de la operación *im2col* [14]) hace posible que la arquitectura Newton, que ofrece la posibilidad de ejecutar la operación GeMV en PIM, pueda ser utilizada también en el ámbito de las CNNs. Así, una operación GeMM no es más que multitud de operaciones GeMV, donde los vectores son columnas de la segunda matriz. Los filtros se almacenan en los bancos de memoria (*Memory Cell Array* en la Figura 1), mientras que las matrices de entrada se hacen estacionarias por vectores en el *buffer* global.

Una característica importante de PIMFlow es que, para permitir paralelismo real entre GPU y PIM, realizan una división de los canales de memoria de la GPU, de manera que dedican 16 canales exclusivos a PIM, quedando los 16 restantes para uso exclusivo de GPU. Esto implica que la GPU tiene a su disposición la mitad del ancho de banda de memoria. Sin embargo, esto permite ejecutar *kernels* en la GPU (haciendo uso de su subconjunto de canales), al mismo tiempo que la parte PIM del sistema puede ejecutar operaciones GeMV.

C. Accel-Sim

Accel-Sim [13] es un *framework* que, entre otros factores, extiende el simulador GPGPU-Sim 3.x [15], dotándolo de diversas mejoras y funcionalidades, en lo que sería su versión 4.0. GPGPU-Sim está caracterizado por su arquitectura modular y su flexibilidad para modelar distintas generaciones de GPUs de NVIDIA. Permite lanzar simulaciones tanto en modo PTX—una representación intermedia entre el código de un *kernel* y el ISA que el ejecuta el *hardware* de la GPU—como en modo SASS—el ISA nativo y específico de cada modelo GPU.

Una de las características más relevantes de Accel-Sim es la inclusión de un módulo de generación de trazas SASS (*Accel-Sim Tracer*). Esto permite generar trazas de la ejecución de *kernels* en una GPU, para posteriormente simularlas en Accel-Sim.

III. Simulación de PIM en LPDDR5

Como hemos comentado en la Sección I, la operación GeMV es ampliamente utilizada en aplicaciones como LLMs y Modelos de Lenguaje Pequeños (*Small Language Models* o SLMs), haciendo que su optimización sea de gran interés. Por otro lado, hemos visto que la técnica de procesamiento en memoria consigue acelerar este tipo de operación. Es por ello que una de nuestras contribuciones ha sido el ampliar Ramulator con la tecnología de memoria LPDDR5, comúnmente utilizada en dispositivos con altos requisitos de eficiencia energética (dispositivos alimentados por baterías o sistemas del *edge*), añadiéndole soporte PIM para GeMV. Para conseguirlo, hemos utilizado como punto de partida la herramienta utilizada para evaluar PIMFlow, donde implementan GDDR6 con

PIM en Ramulator.

Además, como hemos comentado en la Sección II, PIMFlow no realiza una validación funcional del resultado del cómputo con PIM, pues no implementa el funcionamiento de las operaciones PIM (simplemente asocia una latencia a cada una de ellas). Como consecuencia, otra de nuestras contribuciones consiste en la implementación de un módulo de validación funcional.

A. LPDDR5 en Ramulator

Ramulator no implementa LPDDR5. Sin embargo, su sucesor Ramulator2 [8] sí que lo hace. Por tanto, hemos portado LPDDR5 de Ramulator2 a Ramulator. Si bien ambos simuladores presentan, en esencia, la misma estructura, durante el proceso de portado ha sido necesario realizar diferentes adaptaciones sobre la tecnología de memoria simulada.

Para el portado de LPDDR5 de Ramulator2 a Ramulator, hemos modificado la implementación de LPDDR4 de Ramulator, adaptándola según el comportamiento definido en LPDDR5 de Ramulator2. Las principales diferencias entre LPDDR4 y LPDDR5 son, por un lado, un aumento en la tasa de transferencia de datos, que pasa a ser 6400 MT/s, y, por otro, el número de bancos se dobla (pasa de 8 a 16). Teniendo en cuenta esto, junto con la tabla de latencias de LPDDR5 incluida en Ramulator2, hemos sido capaces de implementar LPDDR5 en Ramulator.

Puesto que nuestro objetivo es ejecutar GeMV en LPDDR5, el siguiente paso consiste en dotar a esta tecnología de memoria de dicha capacidad. Para ello, como ya hemos comentado, partimos de la implementación que hace PIMFlow de GeMV (la arquitectura Newton) en GDDR6. En primer lugar, y con el objetivo de poder validar nuestra implementación (ver Sección V-A), añadimos soporte PIM a LPDDR4. Para ello, es necesario incluir la definición de los comandos de Newton (Tabla I) y las latencias asociadas. Una vez implementado PIM en LPDDR4, lo hacemos también para LPDDR5.

B. GeMM en Newton

Como hemos comentado en la Sección II-B, PIMFlow consigue reutilizar la arquitectura Newton para ejecutar capas convolucionales como operaciones de multiplicación matriz-matriz. Esto lo consiguen haciendo uso de la operación *im2col*. Por tanto, en el proceso de generación de trazas PIM de PIMFlow, la operación de multiplicación de matrices está implícita. Sin embargo, la herramienta utilizada para evaluar PIMFlow ofrece facilidades para generar una operación de multiplicación de matrices correspondiente a una capa convolucional. Es decir, dados los parámetros de una convolución (tamaño de la imagen de entrada, tamaño del filtro, canales de entrada, canales de salida, *padding*, *stride*), PIMFlow genera la traza PIM correspondiente a una multiplicación de matrices equivalente a dicha convolución.

Para reutilizar este proceso de generación de trazas, de manera que podamos generar trazas PIM para una operación GeMM según el tamaño de las matrices, es necesario recurrir a la fórmula que representa la multiplicación de matrices de una convolución en función de sus parámetros. La fórmula es:

Tabla II: Valor de los parámetros de una convolución para generar trazas PIM de una GeMM de tamaño $M \times K \times N$.

Parámetro	Valor
O_c	M
I_c	K
H	N
W	1
K_h	1
K_w	1
Padding	0
Stride	1

$$\text{Filtro}[O_c, I_c, K_h, K_w] \times \text{Imagen}[I_c, H, W] = \\ = \text{Salida}[O_c, H_{out}, W_{out}]$$

donde H y W son la altura y anchura de la imagen, K_h y K_w la altura y anchura del filtro, I_c y O_c los canales de entrada y salida, y H_{out} y W_{out} la altura y anchura de la matriz de salida.

El primer paso es reducir el número de dimensiones de todas las matrices implicadas a 2. Para ello, es necesario fusionar las últimas 3 dimensiones de la matriz filtro y las últimas 2 dimensiones de la matriz imagen y la matriz de salida. De esta forma, la fórmula queda de la siguiente manera:

$$\text{Filtro}[O_c, I_c \times K_h \times K_w] \times \text{Imagen}[I_c, H \times W] = \\ = \text{Salida}[O_c, H_{out} \times W_{out}]$$

A continuación, es necesario fijar el valor de los parámetros de una convolución de manera que esta se corresponda con una operación GeMM. Suponiendo una multiplicación de matrices $A[M, K] \times B[K, N] = C[M, N]$, los valores necesarios de cada parámetro de la convolución son los mostrados en la Tabla II. Es importante notar que $I_c \times K_h \times K_w == I_c$, por lo que fijamos $K_h = 1$ y $K_w = 1$, de forma que $K_h \times K_w = 1$. Con estos valores conseguimos generar trazas PIM para una GeMM de tamaño $M \times K \times N$, que posteriormente pueden ser ejecutadas con nuestra implementación de LPDDR5+PIM.

C. Validación funcional de PIM

Para añadir una validación funcional de los datos para GeMV y GeMM, es necesario implementar el comportamiento de los 4 comandos de Newton (Tabla I). Así, a partir de una traza PIM, y siguiendo el flujo de ejecución de Newton, podemos obtener el vector (matriz) resultante de la operación GeMV (GeMM) y compararlo con el vector (matriz) resultado esperado, verificando así su corrección. Si bien en la Sección II-A hemos explicado brevemente el funcionamiento de cada uno de los comandos, vamos a ver a continuación los detalles de cada uno de ellos, con el objetivo de mostrar cómo ha sido realizada la implementación del módulo de validación funcional.

El comando *GWRITE* escribe en el *buffer* global 1 KB, es decir, 512 elementos FP16 procedentes de la segunda matriz. De esto se encarga el *host*, pues dicha matriz se encuentra almacenada en los canales de memoria asignados al *host* (recordemos que PIM-Flow asigna 16 canales exclusivos al *host*—la GPU—y 16 canales exclusivos a PIM).

Por su parte, cada *G_ACT* activa 1 fila de DRAM de 4 bancos. Como ya se ha mencionado, esto se debe a una restricción temporal que añade una cierta latencia (*tFAW*) entre activaciones de filas de 4 bancos. En este caso, dado que los bancos almacenan la primera matriz, y que las lecturas de DRAM (tanto en GDDR6 como en LPDDR5) son de 1 KB, en cada banco se cargan 512 elementos FP16. Para el correcto funcionamiento de este comando, es necesario que la matriz esté almacenada por *tiles* de 512 elementos de manera intercalada entre los bancos, tal y como muestra la Figura 2. Así, *G_ACT* simplemente activa filas de bancos de manera secuencial.

COMP se encarga de realizar la operación de multiplicación y acumulación. Dado que la arquitectura Newton presenta 16 MACs, la granularidad con la que se opera con los 512 elementos cargados en el *buffer* global y en el BLSA es de 16 elementos. Así, para computar los 512 elementos serán necesarios 32 *COMPs*. Es importante destacar que un comando *COMP* actúa sobre todos los bancos, trabajando todos de manera síncrona. El resultado de cada *COMP* se almacena en un registro resultado, local a cada banco. Sobre este registro se acumula el resultado de sucesivos *COMPs*.

Finalmente, *READRES* se encarga de completar la operación GeMV (o GeMM), leyendo el resultado parcial presente en el registro resultado de cada banco y enviándolo al *host*. Es el *host* el que se encarga de realizar la acumulación final, obteniendo el vector (o matriz) resultado. Al igual que *COMP*, *READRES* actúa sobre todos los bancos del canal.

Como ya hemos comentado en la Sección II-B, PIMFlow genera una traza con los comandos PIM, para posteriormente extraer el tiempo de ejecución según un modelo analítico. Esta generación de comandos mapea la multiplicación matriz-vector como se muestra en la Figura 3, de forma que, en primer lugar, opera con una submatriz y subvector múltiplos de 512 (azul claro), dejando para el final la submatriz y subvector no múltiplo de 512 (azul oscuro). Esto es algo a tener en cuenta a la hora de implementar la validación funcional de los datos, ya que afecta al orden en el que los comandos aparecerán en la traza PIM y, por tanto, al orden en el que debemos realizar la multiplicación matriz-vector.

Como hemos visto, PIMFlow carga datos de 512 en 512 elementos y opera con datos de 16 en 16 elementos. Es por ello que, para la multiplicación de la segunda submatriz y subvector (azul oscuro en la Figura 3), y dado que la arquitectura Newton opera con todos los bancos y MACs simultáneamente, deberemos asegurarnos de añadir padding a los datos cargados con ceros hasta completar los 512 elementos. Además, debemos considerar el caso en el que el número de filas de la matriz no sea múltiplo del número de bancos. En ese caso, algunos bancos operarán con ceros (de nuevo, añadiendo ceros a las filas de los bancos correspondientes). Con todo esto conseguimos que los bancos no operen con datos residuales y el resultado sea el esperado.

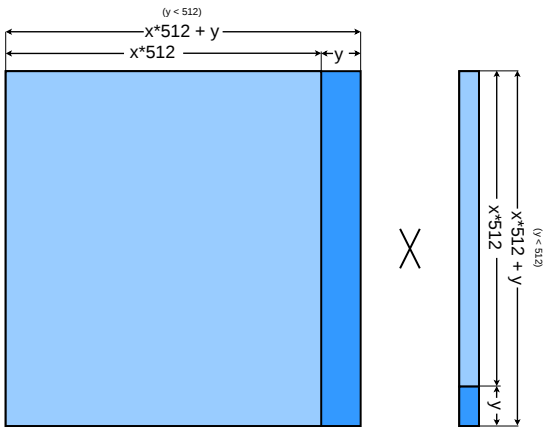


Fig. 3: Forma de operar en PIMFlow con matriz-vector no múltiplo de 512.

Este módulo de validación funcional es de gran valor, no solo por el hecho de que realiza una comprobación funcional del resultado generado por PIM, sino que también nos permite conocer con gran detalle el uso que se hace de la arquitectura. En concreto, tenemos a nuestra disposición el espacio de memoria utilizado en cada banco, el número de veces que las MACs operan con ceros y el balanceo de carga entre los bancos en función del número de operaciones realizadas por cada banco. Este tipo de estadísticas son evaluadas en la Sección V-D.

IV. Metodología

Para el desarrollo de los experimentos hemos empleado dos entornos: un entorno PIM y un entorno GPU. El primero se trata de tecnología de memoria LPDDR5 (Tabla III) que funciona a 6400 MT/s con una frecuencia de 800 MHz, y presenta 1 *rank* y 32 canales, 16 de ellos de uso exclusivo para PIM, con una densidad total de 8 Gb. Dado que el reparto de la carga de trabajo entre canales es equilibrado, funcionando todos ellos en paralelo, los experimentos realizados consisten en la simulación de un único canal (mismo procedimiento que en PIMFlow). Cada canal dispone de 16 bancos, cada uno de ellos equipado con 16 unidades de multiplicación y un árbol de reducción. Hemos utilizado un *buffer* global de 1 KB. Además, hemos considerado la optimización realizada en PIMFlow con la que la latencia del comando *GWRITE* puede ser ocultada lanzando asíncronamente los comandos *G_ACT*.

Por otro lado, para el entorno GPU hemos modificado la arquitectura RTX 3070 de Accel-Sim para acercarla a la GPU de un dispositivo SoC de NVIDIA para el *edge*. De esta forma, podemos comparar la implementación de LPDDR5+PIM con un sistema GPU cuyo propósito es similar (eficiencia energéti-

Tabla III: Configuración LPDDR5.

Característica	Valor
Velocidad de transferencia	6400 MT/s
Ranks	1
Canales PIM	16
Canales *host*	16
Bancos/canal	16
MACs/banco	16
Buffer global	1 KB

Tabla IV: Configuración GPU.

Característica	Valor
SMs	8
Frecuencia interna	800 MHz
Frecuencia memoria	800 MHz
Capacidad L2	1.5 MB

Tabla V: Aplicaciones GeMV.

Dataset	GeMV
DLRMs1 [20]	512x256
BERTs1 [19]	1024x1024
BERTs3 [19]	4096x1024
AlexNetL7 [21]	2048x2048
GNMT LSTMs2 [18]	4096x2048
AlexNetL6 [21]	21632x2048
BERTs2 [19]	1024x4096

Tabla VI: Aplicaciones GeMM.

Dataset	GeMM
Transformer [22]	256x256x256
Transformer [22]	512x512x512
Cora [24]	2708x2708x1433
CiteSeer [24]	3327x3327x3703
Wiki [25]	2405x2405x4973
Facebook [25]	4039x4039x1283
AmazonPhoto [26]	7650x7650x745

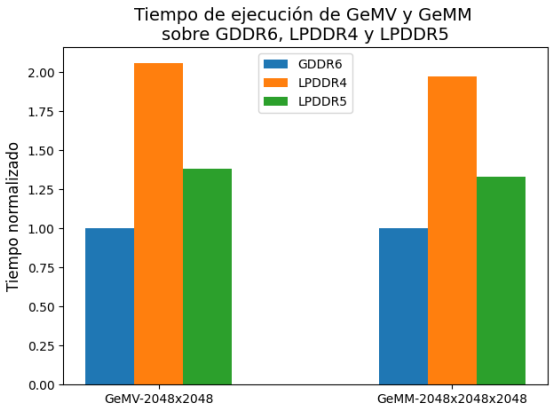


Fig. 4: Validación de la implementación de LPDDR5+PIM mediante la ejecución de una operación GeMV 2048 × 2048 y una operación GeMM 2048 × 2048 × 2048.

ca). En concreto, hemos utilizado como referencia la GPU contenida en el modelo Jetson Orin Nano [16]. Las características más relevantes a tener en cuenta a la hora de adaptar la RTX 3070 son la reducción en el número de SMs (8), una menor frecuencia de funcionamiento (800 MHz), una caché L2 de menor capacidad (1.5 MB) y una tecnología de memoria LPDDR5. Para esto último, hemos definido la frecuencia del sistema de memoria de Accel-Sim a 800 MHz. El hecho de que tanto la RTX3070 como la Jetson Orin Nano se basen en la arquitectura Ampere de NVIDIA hace que esta adaptación sea realizable. La Tabla IV muestra un resumen de la configuración de GPU empleada.

Para la generación de trazas SASS, hemos utilizado una GPU NVIDIA RTX A4500, basada, también, en la arquitectura Ampere.

En las Tablas V y VI encontramos las aplicaciones empleadas para las operaciones GeMV y GeMM. Ambas operaciones son ejecutadas en una RTX A4500 utilizando la biblioteca cuBLAS [17] con valores generados aleatoriamente. En cuanto a GeMV, incluimos tamaños que podemos encontrar en dominios de *machine learning* como procesamiento natural del lenguaje [18], [19] y sistemas de recomendación [20], además de capas *fully connected* (FC) de AlexNet [21], una CNN de sobra conocida. Para GeMM, consideramos tamaños obtenidos en modelos Transformer [22], junto con las dimensiones de las matrices implicadas en la multiplicación de matrices de la fase de agregación de diferentes *datasets* al procesarlos con el modelo de red neuronal de grafos (GNN) GraphSAGE [23].

V. Resultados experimentales

En esta sección vamos a evaluar la implementación de LPDDR5 con soporte PIM. Para ello, en primer lugar, mostramos una validación de dicha implementación. Tras ello, realizamos una comparación del rendimiento ofrecido al ejecutar cargas GeMV y GeMM en PIM frente al modelo simulado de la GPU del sistema Jetson Orin Nano. Finalmente, mostraremos la distribución de los distintos comandos PIM ejecutados, además de mostrar la utilidad del módulo de validación funcional analizando el comportamiento de programas PIM con mayor detalle.

A. Validación de LPDDR5+PIM

La validación de la implementación de LPDDR5+PIM consiste en comparar el tiempo de ejecución en PIM sobre las distintas tecnologías de memoria que actualmente incluyen PIM: GDDR6, ofrecido en PIMFlow base; LPDDR4, al que nosotros le hemos añadido soporte PIM; y LPDDR5, implementado por nosotros tanto la propia tecnología como el soporte PIM. Entre estas 3 tecnologías, las versiones de LPDDR sacrifican rendimiento a costa de una mayor eficiencia energética. Para comparar el rendimiento de manera experimental, hemos ejecutado una operación GeMV de tamaño 2048 × 2048 y una GeMM de 2048 × 2048 × 2048. Si vemos los resultados en la Figura 4, observamos que, efectivamente, GDDR6 es capaz de ejecutar en menos tiempo ambas cargas de trabajo. En cuanto a LPDDR4 y LPDDR5, vemos que LPDDR4 ofrece menor rendimiento que LPDDR5, quedando este último entre GDDR6 y LPDDR4. Como vemos, estos resultados coinciden con lo esperado.

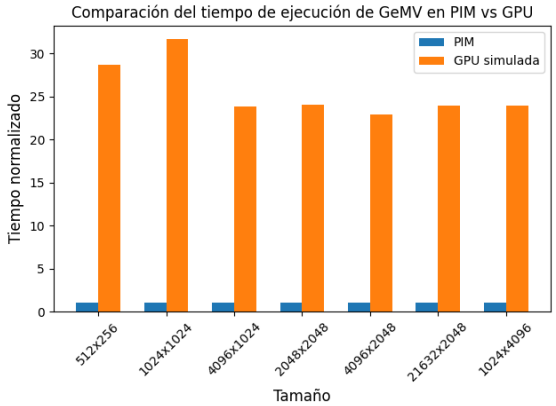


Fig. 5: Comparación del tiempo de ejecución de GeMV en PIM y en el modelo de GPU simulado en Accel-Sim.

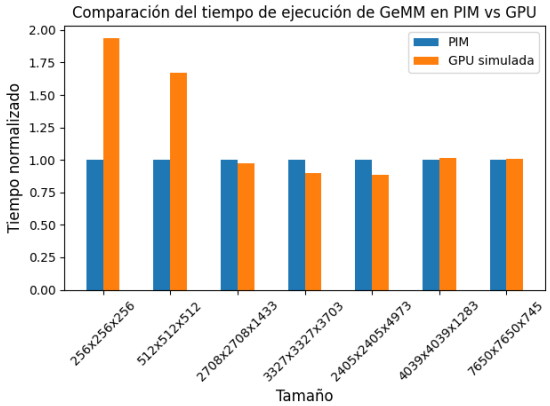


Fig. 6: Comparación del tiempo de ejecución de GeMM en PIM y en el modelo de GPU simulado en Accel-Sim.

B. LPDDR5+PIM vs. GPU simulada

Para la evaluación de nuestra propuesta LPDDR5+PIM comparamos el rendimiento ofrecido con el modelo de GPU Jetson Orin Nano, simulado en Accel-Sim. En la Figura 5 se muestran los tiempos obtenidos para la operación de multiplicación matriz-vector. En este caso, PIM siempre ofrece un mejor rendimiento. Esto se debe a que, como ya hemos comentado en la Sección I, esta operación emplea un mayor tiempo moviendo datos de la memoria a la GPU que propiamente procesándolos, pues no experimentan ningún tipo de reuso. Así, al acercar el cómputo a la memoria, evitamos realizar esa cantidad masiva de movimiento de datos, ahorrando tanto energía como tiempo.

Por otro lado, la Figura 6 compara el tiempo de ejecución de diferentes operaciones GeMM en PIM y en el modelo de GPU simulado Jetson Orin Nano. Vemos que en algunos casos PIM es más eficiente que la GPU, mientras que en otros es justo al contrario. Para entender esto, es apropiado recordar que la arquitectura PIM modelada (Newton) está diseñada para realizar GeMV. Nosotros reutilizamos dicha arquitectura para ejecutar GeMMs considerando la segunda matriz como múltiples vectores. De esta forma, una GeMM no es más que múltiples GeMVs. En concreto, el número de GeMVs dependerá del tamaño de la segunda matriz. Así, a mayor número de GeMVs, menor será el rendimiento ofrecido por PIM. Esto se debe a que la operación GeMM, en general, no está limitada por memoria, pues al ejecutarlas sobre la GPU, existe reuso de los datos. Esto es algo que PIM no permite aprovechar.

Por tanto, los casos en los que PIM ofrece mejor rendimiento que la GPU son aquellos en los que la operación GeMM se puede mapear mediante un número reducido de GeMVs, es decir, la segunda matriz es de tamaño reducido. Solo en los casos en los que la segunda matriz presenta un mayor tamaño ($2708 \times 2708 \times 1433$, $3327 \times 3327 \times 3703$, $2405 \times 2405 \times 4973$) la GPU rinde mejor, pues es capaz de beneficiarse del reuso de los datos.

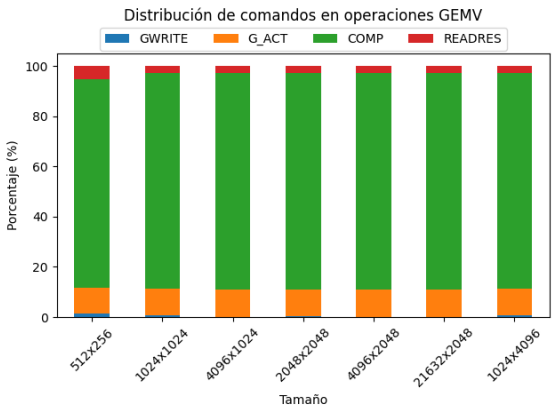


Fig. 7: Distribución de comandos PIM ejecutados en distintas operaciones GeMV.

C. Distribución de comandos PIM

Vamos a estudiar ahora el número de instancias ejecutadas de cada comando PIM. La Figura 7 muestra la distribución de comandos PIM en operaciones GeMV. Como vemos, el comando que domina es el que realiza el cómputo entre la matriz y el vector (*COMP*). Si bien es cierto que cada *COMP* actúa sobre todos los bancos de manera simultánea, el hecho de que opere únicamente sobre 16 elementos hace que sea el comando PIM que más aparece, suponiendo el 85 % de los comandos. Por su parte, el comando *G_ACT* supone un 10 % del total. Finalmente, vemos que *GWRITE* y *READRES* son los menos comunes, representando menos de un 1 % y 3 %, respectivamente.

En la Figura 8 podemos ver que la operación GeMM muestra una misma distribución de comandos PIM que GeMV. Esto se debe a que la implementación de GeMM consiste en repetidas GeMVs.

De estos resultados es importante remarcar el número de instancias del comando *READRES*. Con este comando, el *host* lee el resultado parcial generado por PIM y lo acumula, suponiendo un importante consumo energético al mover datos entre PIM y *host*. Es por ello que resulta de interés conocer el número de este tipo de comandos. Como hemos visto, tanto en GeMV como en GeMM, el 3 % de los comandos son *READRES*. Si bien no es un porcentaje alto, los

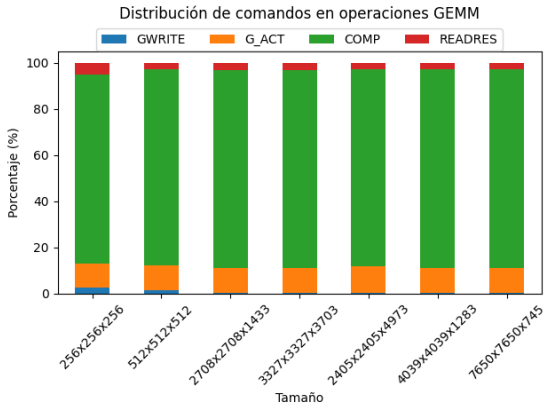


Fig. 8: Distribución de comandos PIM ejecutados en distintas operaciones GeMM.

Tabla VII: Reparto de la primera matriz de GeMM de tamaño $2405 \times 2405 \times 4973$ en los 16 bancos de un canal de memoria.

Bancos	Elementos cargados	Elementos cero
0-4	386560	6.05 %
5-15	384000	6.05 %

comandos *READRES* son algo a tener en cuenta a la hora de proponer optimizaciones de la arquitectura Newton.

D. Estadísticas sobre uso de la arquitectura PIM

La implementación de un módulo de validación funcional que, además de proporcionar una estimación del tiempo de ejecución de una traza PIM, define el flujo de datos según el comportamiento de la arquitectura, no solo permite comprobar que la salida del programa es correcta, sino que también habilita la posibilidad de extraer información acerca del uso de la arquitectura.

Con este módulo podemos conocer la cantidad de datos que ha sido cargada en cada banco. Esto permite conocer la existencia de un desbalanceo al repartir la carga de trabajo entre los distintos bancos de cada canal de memoria. A modo de ejemplo, la Tabla VII muestra cómo se ha distribuido la primera matriz de una GeMM de tamaño $2405 \times 2405 \times 4973$ a lo largo de un canal (16 bancos). Como vemos, los bancos 5-15 almacenan menos elementos. Siguiendo el reparto intercalado realizado de las filas de la matriz entre los bancos, las filas 2401-2405 se asignan a los bancos 0-4, quedando los bancos 5-15 sin más filas de la matriz que asignar.

También podemos extraer información sobre el valor de los datos almacenados en los bancos. En concreto, algo interesante a conocer es el número de elementos cero almacenados. Siguiendo con el mismo ejemplo, en la Tabla VII vemos que el 6.05 % de los elementos almacenados en cada banco son cero. Esto se debe a la necesidad de rellenar con ceros cuando las matrices no son múltiplos de 512 y 16 (ver Sección III-C). Relacionado con esto, también podemos conocer la cantidad de operaciones PIM realizadas con operandos cero. Sin embargo, dado que estamos trabajando con matrices generadas aleatoriamente y que, como vemos, el número de elementos cero es muy bajo (principalmente debido a *padding*), el porcentaje de operaciones donde ambos operandos son cero es muy bajo: 0.46 % por banco.

VI. Conclusiones

Con este trabajo hemos constatado que, por un lado, el procesamiento en memoria es la solución a la ejecución eficiente de aplicaciones limitadas por memoria. Hemos partido de PIMFlow, una herramienta que acelera el procesamiento de CNNs mediante su ejecución paralela en un sistema GPU+PIM, donde la parte PIM implementa la arquitectura Newton sobre GDDR6. Hemos extendido dicha herramienta con la tecnología de memoria LPDDR5, a la que también le hemos añadido soporte PIM. PIMFlow estaba orientado a acelerar CNNs en un sistema GPU+PIM. Aprovechando la representación de una capa convolucional como multiplicación de matrices (a través de la operación *im2col*), hemos sido capaces de modelar la operación de GeMM como una composición de múltiples GeMVs. Con todo ello, hemos visto que PIM es capaz de acelerar operaciones GeMV. Con operaciones GeMM, comparando PIM con una GPU del *edge*, PIM ofrece mejor rendimiento en aquellos casos en los que la segunda matriz es de un tamaño reducido.

Por último, dado que PIMFlow implementa la arquitectura Newton según un modelo analítico, hemos implementado desde cero un mecanismo de validación funcional. Este módulo, además de comprobar que la salida de PIM es correcta a nivel funcional, nos permite conocer con detalle la manera en la que se utiliza la arquitectura PIM, pudiendo extraer estadísticas sobre ello (balanceo de carga entre bancos, operaciones con elementos cero) y siendo de gran utilidad para poder analizar su efectividad.

Agradecimientos

Trabajo realizado en el contexto del proyecto PID2022-136315OB-I00 financiado por MICIU/AEI/10.13039/501100011033/ y "FEDER Una manera de hacer Europa", UE. También por MICIU/AEI/10.13039/501100011033 y la "Unión Europea NextGenerationEU/PRTR", bajo la subvención RYC2021-031966-I. Jaime Martínez-Fernández ha sido financiado mediante la ayuda 22931/FPI/25 de Fundación Séneca, Agencia de Ciencia y Tecnología de la Región de Murcia.

Referencias

[1] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun, *A Modern Primer on Processing in Memory*, pp. 171–243, Springer, 01 2023.

[2] Seungjae Moon, Junseo Cha, Hyunjun Park, and Joo-Young Kim, "Hybe: Gpu-npu hybrid system for efficient llm inference with million-token context window," in *Proceedings of the ACM International Symposium on Computer Architecture (ISCA)*, 06 2025, pp. 808–820.

[3] Hyojung Lee, Daehyeon Baek, Jimyoung Son, Jieun Choi, Kihyo Moon, and Minsung Jang, "Paise: Pim-accelerated inference scheduling engine for transformer-based llm," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025, pp. 1707–1719.

[4] Sungmin Yun, Hwayong Nam, Jaehyun Park, Byeongho Kim, Jung Ho Ahn, and Eojin Lee, "Grande: Efficient near-data processing architecture for graph neural networks," *IEEE Transactions on Computers*, vol. 73, no. 10, pp. 2391–2404, 2024.

[5] Christina Giannoula, Peiming Yang, Ivan Fernandez, Jiacheng Yang, Sankeerth Durvasula, Yu Xin Li, Mohammad Sadrosadati, Juan Gomez Luna, Onur Mutlu, and Gennady Pekhimenko, "Pygim: An efficient graph neural network library for real processing-in-memory architectures," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 8, no. 3, Dec. 2024.

[6] Taehwan Kim, Yunki Han, Seohye Ha, Jiwan Kim, and Lee-Sup Kim, "Eod: Enabling low latency gnn inference via near-memory concatenate aggregation," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, New York, NY, USA, 2025, ISCA '25, p. 1125–1139, Association for Computing Machinery.

[7] Seong Hoon Seo, Junghoon Kim, Donghyun Lee, Seonah Yoo, Seokwon Moon, Yeonhong Park, and Jae W. Lee, "Facil: Flexible dram address mapping for soc-pim cooperative on-device llm inference," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025, pp. 1720–1733.

[8] Haocong Luo, Yahya Tu˘grul, F. Bostancı, Ataberk Olgun, A. Ya˘glıkc¸ı, and Onur Mutlu, "Ramulator 2.0: A modern, modular, and extensible dram simulator," *IEEE Computer Architecture Letters*, vol. PP, pp. 1–4, 01 2023.

[9] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob, "Dramsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020.

[10] Yoongu Kim, Weikun Yang, and Onur Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, 2016.

[11] Yongwon Shin, Juseong Park, Sungjun Cho, and Hyojin Sung, "Pimflow: Compiler and runtime support for cnn models on processing-in-memory dram," in *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization*, New York, NY, USA, 2023, CGO '23, p. 249–262, Association for Computing Machinery.

[12] Mingxuan He, Choungki Song, Ilkon Kim, Chunseok Jeong, Seho Kim, Il Park, Mithuna Thottethodi, and T. N. Vijaykumar, "Newton: A dram-maker's accelerator-in-memory (aim) architecture for machine learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 372–385.

[13] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers, "Accel-sim: An extensible simulation framework for validated gpu modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 473–486.

[14] Jordi Fornt, Pau Fontova-Musté, Martí Caro, Jaume Abella, Francesc Moll, Josep Altet, and Christoph Studer, "An energy-efficient gemm-based convolution accelerator with on-the-fly im2col," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 11, pp. 1874–1878, 2023.

[15] Tor M. Aamodt et al., "Gpgpu-sim 3.x manual," junio 2017.

[16] NVIDIA, "Jetson orin," `https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-orin/`, 2023, Accessed: 2026-04-09.

[17] NVIDIA, "cublas library," `https://docs.nvidia.com/cuda/cublas/index.html`, 2024, Accessed: 2026-04-09.

[18] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, and Jeffrey Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[20] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, and Misha Smelyanskiy, "Deep learning recommendation model for personalization and recommendation systems," *arXiv preprint arXiv:1906.00091*, 2019.

[21] Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Yoesoep Rachmad, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, pp. 1097–1105, 01 2012.

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. 2017, vol. 30, Curran Associates, Inc.

[23] William L. Hamilton, Rex Ying, and Jure Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2017, NIPS'17, p. 1025–1035, Curran Associates Inc.

[24] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93, Sep. 2008.

[25] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, Sourav S. Bhowmick, and Juncheng Liu, "Pane: scalable and effective attributed network embedding," *The VLDB Journal*, vol. 32, no. 6, pp. 1237–1262, Mar. 2023.

[26] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann, "Pitfalls of graph neural network evaluation," *arXiv preprint arXiv:1811.05868*, 2018.

[27] Benedek Rozemberczki, Carl Allen, and Rik Sarkar, "Multi-scale attributed node embedding," *Journal of Complex Networks*, vol. 9, no. 2, pp. cnab014, 05 2021.

Redes de comunicaciones (2/2)

Análisis mediante TCPN de los efectos del error de tiempo en TSN

Alitzel Galilea Torres Macías[1,2], Antonio Ramírez Treviño[1] José Luis Briz[2], Juan Segarra[2]

Resumen— **La planificación de flujos en Redes Sensibles al Tiempo (TSN) es muy sensible a las incertidumbres de sincronización y a los errores temporales (TE) dependientes del *hardware*. La estimación conservadora del TE lleva a planificar infrautilizando recursos, mientras que la optimista puede generar planificaciones óptimas, en teoría, pero que incumplen plazos al ser desplegadas. El correcto ajuste paramétrico del planificador complica a menudo el despliegue y puesta a punto del sistema. Este trabajo presenta un banco de pruebas modular basado en Redes de Petri Continuas Temporizadas (TCPN) que evalúa el rendimiento de planificaciones basadas en el Time-Aware Shaper (IEEE 802.1Qbv TAS) ante variaciones del TE, emitiendo recomendaciones del intervalo de sincronización para la TSN real y contribuyendo al acortamiento del ciclo de ajuste y despliegue del sistema. Dos casos de estudio muestran la capacidad del enfoque propuesto para caracterizar la tolerancia de planificaciones más allá de la comprobación de factibilidad convencional.**

Palabras clave— **Simulador, Redes Sensibles al Tiempo, IEEE 802.1Qbv, IEEE 802.1AS, Redes de Petri Continuas Temporizadas.**

I. Introducción

LAS Redes Sensibles al Tiempo (*Time-Sensitive Networks*, TSN) cumplen con una serie de estándares de capa de Enlace de Datos que permiten una comunicación determinista y de criticidad mixta a través de Ethernet y redes inalámbricas, facilitando la consolidación de redes en distintos sectores (automovilístico, aeroespacial, subestaciones eléctricas) e impulsando la convergencia de IT (*Information Technology*) y OT (*Operational Technology*) en los sistemas de fabricación de la Industria 4.0/5.0.

En TSN se diferencian dos tipos de nodos: estaciones terminales (*end-stations* o *hosts*) y *bridges*. Las primeras actúan como emisores (*talkers*) o receptores (*listeners*) de flujos de datos, mientras que los *bridges* son dispositivos de interconexión con capacidades de regulación de tráfico (e.g., IEEE 802.1Qav, IEEE 802.1Qbv [1]) y sincronización temporal (IEEE 802.1AS) [2].

La regulación de tráfico sensible al tiempo en TSN se realiza mediante *time shapers*. El *shaper* más flexible en el estándar, implementado en los *bridges*, es el *Time-Aware Shaper* (TAS) [1], (Fig. 1). El TAS asigna distintas ventanas de transmisión a distintos tipos de tráfico, evitando las esperas no acotadas propias de la transmisión en switches Ethernet.

La planificación de los TAS considera las especificaciones de los flujos (periodo, *deadline*, tamaño de trama e identificador de tipo tráfico) junto con la topología de la red. En el TAS, un conmutador interno dirige las tramas de cada flujo hacia una cola dependiente de su identificador de tráfico. La transmisión se regula mediante la apertura y cierre de cada compuerta (*gate*) por cola, creando ventanas de transmisión. Los tiempos de apertura o cierre son el resultado de la planificación y están codificados en una lista de control (*Gate Control List*, GCL) por TAS, inicializada en tiempo de despliegue y operativa a partir de un ciclo de red preciso.

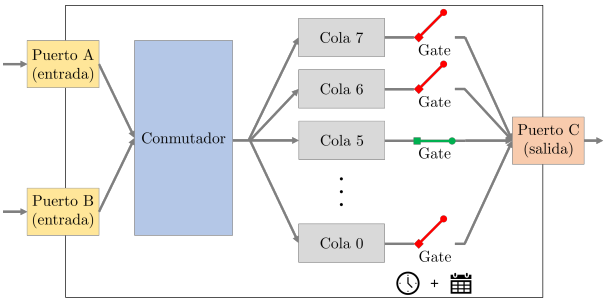


Fig. 1: *Bridge* simplificado tal como se define en el estándar IEEE 802.1Q; obtenido de [3].

Cada nodo cuenta con un oscilador local que le permite establecer su base de tiempo. Todos los nodos están sincronizados con precisión de nanosegundos; sin embargo, debido a la precisión finita de los osciladores, la tecnología de fabricación y las variaciones en el sellado temporal inherentes al protocolo de sincronización, el tiempo local en los distintos dispositivos tiende a divergir gradualmente.

Los mecanismos de sincronización temporal de la TSN ajustan periódicamente los relojes de los nodos, pero los desvíos entre intervalos de sincronización pueden ser suficientemente significativos como para que los algoritmos de planificación tengan que estimarlos e incorporarlos en el cálculo de las ventanas de transmisión para evitar incumplimientos temporales. En la primera puesta en marcha o en replanificaciones por modificaciones en el sistema TSN, la estimación y el ajuste de estos desvíos pueden suponer varios ciclos de replanificación y despliegue.

Los planificadores TSN no conocen *a priori* las desviaciones entre tiempos locales de los nodos en una plataforma y deben estimarlas para garantizar el cumplimiento de plazos temporales. La sobreestimación lleva a infrautilizar recursos, lo que puede reducir el espacio de soluciones de planificación, considerando no planificables escenarios que lo son. La subestimación conlleva el riesgo de incumplimiento de plazos, puesta la planificación en marcha. Más aún, las planificaciones con similar holgura total (huecos

[1]CINVESTAV Unidad Guadalajara, México, e-mail: {alitzel.torres,antonio.ramirezt}@cinvestav.mx
[2]Departamento de Informática e Ingeniería de Sistemas / I3A, Universidad de Zaragoza, e-mail: {a.torres,briz,jsegarra,alex.gracia}@unizar.es

temporales sin transmisión) toleran de forma muy distinta las desviaciones temporales entre nodos, dependiendo de la distribución de dichos huecos temporales entre ventanas de transmisión consecutivas.

Los trabajos que abordan la evaluación de planificadores en TSN obvian estas variaciones temporales, críticas en muchos casos de uso. Por este motivo, proponemos un banco de pruebas basado en un modelo de Redes de Petri Continuas Temporizadas (*Timed Continuous Petri Nets*, TCPN) que simula el comportamiento temporal de la TSN, incluyendo un controlador que implementa el TAS a partir de la planificación proporcionada. Esta herramienta permite la variación supervisada de los parámetros que afectan a las desviaciones de los relojes, mide la resiliencia de la planificación a dichas variaciones y la utilización de los recursos del sistema, y recomienda un intervalo de sincronización para la TSN real a partir de los resultados obtenidos, orientando al diseñador sin necesidad de rediseñar la planificación.

En [4] se estudia la robustez de dos planificaciones basadas en TAS ante un desfase de frecuencia constante en los relojes locales de cada nodo de la red respecto a un reloj global. El presente trabajo extiende dicho análisis considerando el ruido en el modelo de los relojes y se estudian sus efectos en el rendimiento de las planificaciones, mostrando así que el modelo TCPN es aplicable a variaciones no lineales de las derivas de reloj.

La comunicación se organiza de la siguiente manera. La Sec. II revisa los fundamentos de la sincronización temporal en TSN, describiendo el modelo de reloj utilizado en este estudio en el contexto del protocolo gPTP. La Sec. III presenta un ejemplo motivador que ilustra la importancia del estudio de las desviaciones de relojes en las TSN. La Sec. IV resume el modelo formal en TCPN, empleado para el desarrollo del banco de pruebas en [4], y detalla el modelado del comportamiento de los relojes en la red. La Sec. V describe el estado actual de la herramienta de simulación desarrollada, su arquitectura y las métricas que produce. La Sec. VI presenta dos casos de estudio, detallando los experimentos realizados y el análisis de los resultados obtenidos. La última sección recoge conclusiones y perspectivas.

II. Relojes, TE y Sincronización Temporal

La sincronización de los relojes de nodos en TSN se realiza mediante el protocolo gPTP (*generalized Precision Time Protocol* [2]). Con precisión de nanosegundos, se basa en el estándar IEEE 1588 [5] y especifica perfiles según campo de aplicación (e.g., IEEE/IEC 60802 para automatización industrial, IEEE P802.1DP para aerospacial) [6].

Como en protocolos similares, gPTP establece un nodo como *timeTransmitter* (tT), respecto a cuyo reloj ajusta periódicamente los relojes del resto de nodos (*timeReceivers*, tR). Dicho ajuste considera tanto la diferencia entre tiempos (*time offset*) como entre frecuencias (*frequency offset*), mediante *sincronización* de tiempo (fase; *phase alignment*) en el primer caso y *sintonización* de frecuencia en el segundo. La sincronización utiliza un mecanismo de latencia entre pares (*peer-to-peer delay mechanism*), calculando la diferencia entre el tiempo de cada tR y el del tT mediante diferencias entre el sello temporal de los mensajes. La sintonización se ocupa de la diferencia entre la frecuencia de cada tR y la del reloj del tT.

Los errores de precisión en el sellado temporal de los mensajes utilizados para sincronizar, por un lado, y las derivas de sintonización de las frecuencias de reloj de los nodos, por otro, contribuyen a desviaciones temporales en la llegada y la transmisión de tramas en un sistema real, que los planificadores deben tener en cuenta. Para mostrar las ventajas y la viabilidad del modelo propuesto, este estudio se centra específicamente en los errores de sintonización. En lo que sigue, se definen el *error de tiempo* (TE), sus componentes y su modelado en el sistema.

A. Modelo de reloj y error de tiempo

Los estudios sobre metrología de tiempo y frecuencia [7–9] definen el tiempo de un reloj como la medida de tiempo ideal τ que proporciona. Así, la función de tiempo $Z(\tau)$ generada por un reloj es

$$Z(\tau) = \frac{\Theta(\tau)}{2\pi\nu_{nom}}, \tag{1}$$

donde ν_{nom} es la frecuencia nominal del reloj y $\Theta(\tau)$ es la fase instantánea total, modelada como:

$$\Theta(\tau) = \Theta_0 + 2\pi\nu_{nom}(1+y_0)\tau + D\pi\nu_{nom}\tau^2 + \vartheta(\tau), \tag{2}$$

Θ_0 es la desviación de fase inicial, D la tasa de deriva de frecuencia relativa lineal, que representa esencialmente los efectos de envejecimiento del oscilador, y $\vartheta(\tau)$ la componente de desviación de fase aleatoria. y_0 (*frequency offset*) es una métrica de precisión del oscilador, medida como error de la frecuencia relativo al valor nominal ν_{nom}; si se expresa en ppm (partes por millón), se calcula como $y_0 = \frac{f(D,\vartheta(\tau))}{\nu_{nom}} \times 10^{-6}$. Así, si un nodo se adelanta 1 s respecto al tT cada 24 h, el y_0 respecto al tT es de 11.6 ppm. En la práctica, y_0 se estima a partir del NNR y del RR (*Neighbor Rate Ratio*, *Rate Ratio*) [2].

El TE $x(\tau)$ es la diferencia entre el tiempo de un reloj $Z(\tau)$ y el tiempo indicado por un estándar de tiempo de referencia acordado $Z_{ref}(\tau)$ [7]. Suponiendo que dicho tiempo de referencia es el correspondiente al tT y éste es ideal (i.e., $Z_{ref}(\tau) = \tau$), sustituyendo (2) en (1), se obtiene el siguiente modelo

$$\begin{aligned} x(\tau) &= Z(\tau) - Z_{ref}(\tau) = Z(\tau) - \tau \\ &= x_0 + y_0\tau + \frac{D}{2}\tau^2 + \frac{\vartheta(\tau)}{2\pi\nu_{nom}}, \end{aligned} \tag{3}$$

donde x_0 es la desviación inicial de tiempo y $\frac{\vartheta(\tau)}{2\pi\nu_{nom}}$ representa la componente de ruido aleatorio. $Z(\tau)$ en (3) corresponde al *modo de funcionamiento libre* definido en [8]. Dicho de otra forma, el reloj del tR no está sincronizado con el del tT.

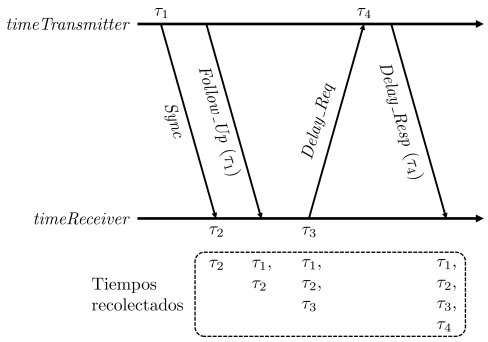


Fig. 2: Secuencia de intercambio de mensajes en PTP.

De acuerdo con [10], en requisitos industriales, como es el caso de las TSN, la tasa de deriva de frecuencia D se ignora, pues el sistema debe estar en funcionamiento durante más de 300 años para que el envejecimiento sea perceptible frente a la desviación de frecuencia aceptable establecida en [2] ($y_0 = \pm 100$ ppm). Si, además, se omite el componente de ruido aleatorio por simplicidad de modelado, se obtiene la representación afín del tiempo del reloj que se utiliza generalmente [11–16].

No obstante, los relojes reales presentan fluctuaciones aleatorias debidas a ruido físico, efectos de implementación e imperfecciones del oscilador [17], que estos modelos lineales y deterministas no capturan. Aunque el origen físico del ruido suele manifestarse como ruido de frecuencia [9, 18, 19], su impacto en sistemas TSN se observa principalmente como variaciones en el TE, dado que los eventos planificados (aperturas y cierres de *gates*) se ejecutan en función del tiempo local de cada nodo. Como se verá a continuación, la sincronización temporal no elimina estas fluctuaciones, por lo que conforman el TE residual.

B. Sincronización y protocolo gPTP

Cuando el *timeReceiver* ($Z(\tau)$) está sincronizado con el *timeTransmitter* ($Z_{ref}(\tau)$), se dice que opera en *modo enganchado* [8], lo que lleva a reescribir el modelo de TE como:

$$x(\tau) = x_0 + \frac{\vartheta(\tau)}{2\pi\nu_{nom}}. \tag{4}$$

Aquí, la desviación inicial x_0 corresponde al componente constante (*constant Time Error*, cTE), mientras que el componente de ruido $\frac{\vartheta(\tau)}{2\pi\nu_{nom}}$ representa el componente dinámico (*dynamic Time Error*, dTE).

El mecanismo de sincronización por latencia entre pares (*peer-to-peer delay mechanism*) utiliza los mensajes de solicitud (*Pdelay_Req*), respuesta (*Pdelay_Resp*) y seguimiento de respuesta (*Pdelay_Resp_Follow_Up*). A grandes rasgos, el protocolo funciona de la siguiente manera: el tT envía mensajes de sincronización (*Sync*) cada *syncInterval* (por defecto, 125 ms [2]) y cada dispositivo tR los propaga a lo largo de la red. La llegada y transmisión de estos mensajes se sellan temporalmente en *hardware* con los relojes locales en modo de funcionamiento libre (ver Fig. 2).

Estos sellos de tiempo permiten medir la latencia de procesamiento (el tiempo de residencia) en un *bridge*, la latencia del enlace entre dos dispositivos (*meanLinkDelay*) y la desviación de frecuencia de reloj entre dichos dispositivos (NRR), así como su relación con la frecuencia del *timeTransmitter* (RR) [20]. Cada tR emplea estos cálculos para determinar el TE respecto al reloj del tT y así sincronizar sus relojes.

Dado que los sellos de tiempo se obtienen con el reloj local del tR, los tiempos de residencia medidos y los valores de latencia promedio del enlace deben escalarse para compensar la desviación de frecuencia del reloj local respecto al tT [20]. Sin esta compensación, un error de frecuencia de ±100 ppm puede traducirse en un TE de hasta 1 μs con un tiempo de residencia de 10 ms, excediendo los límites establecidos en [2].

En definitiva, la precisión de la sincronización depende fundamentalmente de la calidad en la estimación del retardo entre el tT y cada tR: cuanto mejor sea dicha estimación, menor será el TE residual. El estudio de los errores asociados a los sellos temporales y al proceso de sincronización se deja como trabajo futuro.

C. Efecto de las desviaciones de reloj en la planificación TSN

El análisis del TE revela que la magnitud de la desviación de los relojes locales respecto al reloj global no es el único factor de riesgo; la dirección geométrica de dicha variación (su signo) también juega un papel determinante. La divergencia acumulada entre sus tiempos locales es máxima cuando los osciladores de dos nodos tR adyacentes presentan derivas con signos opuestos, es decir, cuando el reloj de uno avanza más rápido que el del tT mientras que el del otro se retrasa. Esta combinación asimétrica de signos acelera drásticamente el desalineamiento entre las ventanas planificadas de ambos nodos antes del siguiente *syncInterval*, comprometiendo severamente las garantías de calidad de servicio (QoS). La Tabla I muestra las combinaciones de signo entre dos nodos adyacentes a y b (i.e., $(a, b) \in \mathcal{E}$), sus efectos físicos y su nivel de impacto.

Se denomina *violación de ventana* al evento en que una trama no logra transmitirse dentro de su ventana de transmisión planificada. Esto ocurre en dos situaciones: 1) cuando la trama llega mientras la *gate* ya está abierta y el tiempo restante de la ventana es insuficiente para completar su transmisión o 2) cuando la trama llega una vez que la *gate* ya se ha cerrado. En ambos casos, la trama es retenida en cola hasta la siguiente apertura de la *gate* correspondiente o descartada, según la configuración del sistema, comprometiendo el cumplimiento del *deadline*. En ausencia de aislamiento adecuado entre ventanas, una trama que llega anticipadamente puede transmitirse en una ventana ajena, introduciendo interferencias entre flujos que constituyen una categoría distinta de fallo de planificación.

La Tabla I considera el signo de la desviación de

Tabla I: Impacto de la combinación direccional de las desviaciones de reloj de dos nodos adyacentes a y b respecto al tiempo global τ.

Nodo		
a	b	**Efecto físico e impacto**
+	+	Moderado: ambos relojes se adelantan. La desalineación relativa se mantiene acotada, retrasando la aparición de violaciones.
−	−	Moderado: ambos relojes se retrasan. Mismo comportamiento de desalineación relativa que en el caso anterior.
+	−	Crítico: el nodo a transmite antes y el nodo b abre su ventana tarde. Maximiza posibilidad de descarte de tramas por acumulación en cola.
−	+	Crítico: el nodo a transmite tarde y el nodo b cierra su ventana antes. Provoca violaciones de ventana.

cada nodo de forma estática. En la práctica, un mismo nodo puede presentar desviación inicial (x_0) y de frecuencia (y_0) con signos opuestos, lo que da lugar a un comportamiento dinámico más complejo. Así, si $x_0 > 0$ e $y_0 < 0$, el reloj del nodo comienza adelantado respecto al tiempo global pero se retrasa progresivamente, de modo que su signo efectivo cambia a lo largo del tiempo. En tal caso, el nivel de impacto sobre la planificación no es constante: puede transitar de un régimen moderado a uno crítico (o viceversa) dependiendo del instante en que se evalúe el sistema respecto al *syncInterval*. Esta naturaleza dinámica de las desviaciones refuerza la necesidad de un análisis de robustez que vaya más allá de los casos estáticos de peor caso, motivando el enfoque de simulación presentado en las secciones siguientes.

III. Motivación

Con relojes ideales, el *talker* ES_1 y el *bridge* posterior BR_1 de la Fig. 3 operan con desviación de reloj cero y siguen estrictamente la GCL en tiempo global $Z_{ref} = \tau$. ES_1 realiza la transmisión de tramas en el instante planificado, posiblemente incluyendo un desfase de transmisión inicial. En BR_1, la transmisión se realiza según lo previsto, ya que la trama ya ha llegado y la *gate* está abierta, cerrándose de acuerdo con la entrada activa de la GCL.

Con relojes reales, las diferentes desviaciones de frecuencia acumulan desviaciones progresivas de reloj de signo opuesto en ambos nodos. La transmisión de tramas en ES_1 se retrasa en tiempo global, mientras que la ventana de transmisión correspondiente en BR_1 se adelanta. Como resultado, la trama llega a BR_1 demasiado tarde para completar su transmisión antes de que se cierre la *gate*: hay violación de ventana y la trama queda en cola hasta la siguiente apertura, incumpliendo sus restricciones temporales. La desviación entre el inicio de transmisión planificado y el real se captura mediante el *Transmission Offset* (TO), cuya raíz cuadrada se indica en la figura

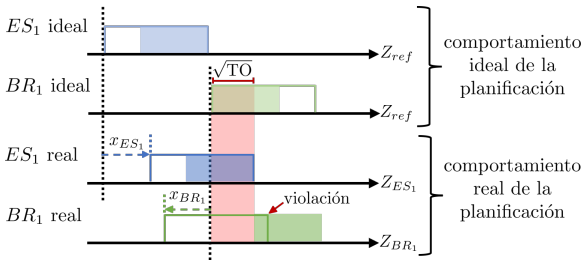


Fig. 3: Violación de ventana debido a desviaciones de reloj en el *talker* ES_1 y el *bridge* BR_1; adaptada de [4].

y se define formalmente en la Sec. V.

IV. Modelado de Redes Sensibles al Tiempo

Utilizamos una metodología de modelado modular de TSN basada en Redes de Petri Continuas Temporizadas (TCPN), desarrollada a partir de [3, 4]. *Talkers*, *listeners* y *bridges* se representan como módulos TCPN. Su interconexión según la topología de la TSN permite generar el modelo del sistema completo. Los conceptos básicos de TCPN pueden ampliarse en [21–23].

A. Redes de Petri Continuas Temporizadas

Definición 1: Una *estructura de Red de Petri* es un digrafo bipartito $N = (\mathcal{P}, \mathcal{T}, Pre, Post)$, donde $\mathcal{P}$ y $\mathcal{T}$ son conjuntos finitos no vacíos y disjuntos de lugares y transiciones, respectivamente, y $Pre, Post : \mathcal{P} \times \mathcal{T} \to \mathbb{N} \cup \{0\}$ son funciones que representan el peso de los arcos que existen de lugares a transiciones y de transiciones a lugares.

La matriz de incidencia C de N de dimensión $|\mathcal{P}| \times |\mathcal{T}|$ se define como $C[i,j] = C^+[i,j] - C^-[i,j]$, donde $C^+[i,j] = Post(i,j)$ y $C^-[i,j] = Pre(i,j)$. Un marcado continuo es una función $m : \mathcal{P} \to \mathbb{R}^+$ que asigna un valor real no negativo a cada lugar. Esta función es representada por el vector $\vec{m}[j] = m(p_j)$.

Definición 2: Una *Red de Petri Continua* se define como $CPN = (N, \vec{m}_0)$, donde N es una estructura de Red de Petri y $\vec{m}_0 \in \{\mathbb{R}^+\}^{|\mathcal{P}|}$ es el marcado inicial.

Una transición t_i está *habilitada* en un marcado $\vec{m}$ si y solo si su *grado de habilitación*,

$$enab(t_i, \vec{m}) = \min_{p_j \in {}^\bullet t_i} \left(\frac{\vec{m}[j]}{C^-[j,i]} \right), \tag{5}$$

es mayor que cero. El disparo de t_i en cualquier cantidad real entre el intervalo $0 \le \eta \le enab(t_i, \vec{m})$ lleva a un nuevo marcado $\vec{m}' = \vec{m} + \eta C[\cdot, i]$. El lugar p_j restringe a t_i cuando $enab(t_i, \vec{m}) = \frac{\vec{m}[j]}{C^-[j,i]}$.

El marcado $\vec{m}$ es *alcanzable* si puede ser alcanzado desde $\vec{m}_0$ al disparar una secuencia σ de transiciones habilitadas. El conjunto de alcanzabilidad contiene todos los marcados alcanzables desde $\vec{m}_0$ que satisfacen la ecuación fundamental $\vec{m} = \vec{m}_0 + C\vec{\sigma}$, donde $\vec{\sigma} \in \{\mathbb{R}^+\}^{|\mathcal{T}|}$ es el vector de disparos de una secuencia de transiciones habilitadas.

Definición 3: Una *Red de Petri Continua Temporizada* se define como $TCPN = (N, \vec{\lambda}, \vec{m}_0)$, donde $(N, \vec{m}_0)$ es una CPN y el vector $\vec{\lambda} \in \{\mathbb{R}^+\}^{|\mathcal{T}|}$ re-

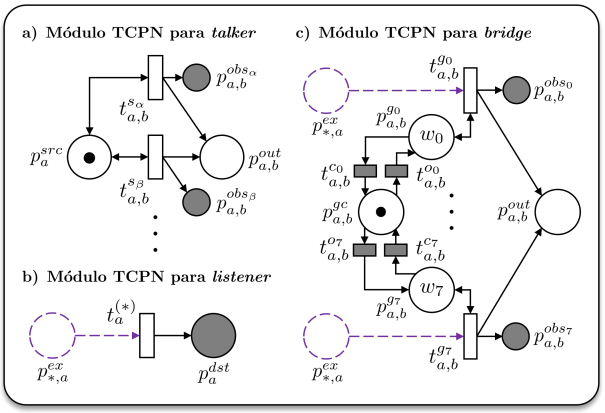


Fig. 4: Módulos TCPN para cada nodo TSN. Los lugares (transiciones) en gris son medibles (controlables); los lugares con contorno punteados son los que se utilizan para hacer las interconexiones entre módulos según la topología $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

presenta la tasa de disparo de las transiciones que determina la evolución temporal del sistema.

Bajo la semántica de servidores infinitos, el *flujo de disparo* $\varphi_i(\vec{m})$ de una transición t_i es proporcional a su grado de habilitación y está definido como $\varphi_i(\vec{m}) = \lambda_i enab(t_i, \vec{m})$. Una *configuración* $\mathcal{E}$ de una *TCPN* es un conjunto de arcos (p, t), uno por transición. La matriz de configuración de $\mathcal{E}$ es

$$\Pi[j,i] = \begin{cases} \frac{1}{C^-[j,i]} & \text{si } (p_i, t_j) \text{ pertenece a } \mathcal{E}, \\ 0 & \text{cualquier otro caso.} \end{cases}$$

Una configuración $\mathcal{E}$ está activa en un marcado $\vec{m}$ si $\forall (p,t) \in \mathcal{E}$, p restringe a t; aquí, $\Pi(\vec{m})$ es la matriz de la configuración activa y el flujo que atraviesa todas las transiciones se escribe en forma vectorial como $\varphi(\vec{m}) = \Lambda\Pi(\vec{m})\vec{m}$, donde $\Lambda = diag(\lambda_1, \ldots, \lambda_{|\mathcal{T}|})$ es la matriz de tasas de disparo.

La ecuación de estado que describe el comportamiento dinámico de una *TCPN* es

$$\dot{\vec{m}} = C\varphi(\vec{m}) = C\Lambda\Pi(\vec{m})\vec{m}. \tag{6}$$

Puede aplicarse una acción de control a (6) añadiendo un término $\vec{u}$ a cada transición t_i, tal que $0 \leq u_i \leq \varphi_i(\vec{m})$ (usaremos φ_i en lugar de $\varphi_i(\vec{m})$ por simplicidad), lo que indica que su flujo puede ser reducido o detenido. De esta forma, el flujo controlado de t_i es $\omega_i = \varphi_i - u_i$ y la ecuación de estado queda reescrita como $\dot{\vec{m}} = C[\varphi - \vec{u}] = C\Lambda\Pi(\vec{m})\vec{m} - C\vec{u}$, donde $0 \leq \vec{u} \leq \Lambda\Pi(\vec{m})\vec{m}$.

B. Modelo del sistema TSN

La Fig. 4 presenta los módulos TCPN para cada tipo de nodo. Éstos son modelos unitarios que pueden interconectarse siguiendo la topología de la TSN, formando así un único modelo integrado. Específicamente, la Fig. 4a ilustra el puerto de salida del *talker* a al *bridge* b, la Fig. 4b muestra un puerto de entrada del a-ésimo *listener* y la Fig. 4c representa el puerto de salida con ocho colas/*gates* $q = \{0, \ldots, 7\}$ de un *bridge* a hacia el nodo b (que puede ser otro *bridge* o un *listener*), basado en el *bridge* simplificado descrito en la Sec. I. Supongamos que existe un enlace entre el nodo a y el nodo b. En este caso, la salida $p^{out}_{a,b}$ del módulo *talker* o *bridge* sustituye a la salida $p^{ex}_{a,b}$ del módulo *bridge* o *listener*. La Fig. 5 muestra un ejemplo de interconexión de nodos TSN.

La ecuación de estado para el modelo integrado está dada por:

$$\begin{aligned} \dot{\vec{m}} &= \begin{bmatrix} C_T & 0 & 0 \\ 0 & C_L & C_{BL} \\ C_{TB} & 0 & C_B \end{bmatrix} \begin{bmatrix} \Lambda_T & 0 & 0 \\ 0 & \Lambda_L & 0 \\ 0 & 0 & \Lambda_B \end{bmatrix} \Pi\, \vec{m} \\ &\quad - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \hat{C}_B \end{bmatrix} \vec{u}, \\ y &= \mathcal{O}\vec{m}. \end{aligned} \tag{7}$$

Las matrices C_T, C_L y C_B corresponden a los módulos *talkers*, *listeners* y *bridges*, respectivamente, siendo Λ_T, Λ_L y Λ_B sus matrices de tasas de disparo asociadas. La matriz C_{TB} captura los arcos desde los lugares *talker* hasta las transiciones *bridge*, mientras que la matriz C_{BL} captura los arcos desde los lugares *bridge* hasta las transiciones *listener*. En $\hat{C}_B$, todas las columnas son cero excepto las de las transiciones $t^{o_q}_{a,b}$ y $t^{c_q}_{a,b}$, que representan la apertura y el cierre de la q-ésima *gate* del puerto de salida b del *bridge* a y son controlables (gris en Fig. 4). Los controladores generan las señales en $\vec{u}$ para abrir y cerrar las *gates* según lo requerido. La matriz $\mathcal{O}$ identifica los lugares medibles (gris en Fig. 4).

Se asume que el planificador ha tenido en cuenta el número y tamaño de colas en los TAS, por lo que este modelo no las limita, centrándose en el caso crítico de combinación de signos $(-,+)$ de la Tabla I.

C. Incorporación del tiempo local al modelo

La estructura modular permite incorporar explícitamente los tiempos locales de cada nodo en la dinámica del sistema mediante la regla de la cadena. Así, por ejemplo, la expresión siguiente describe el subsistema correspondiente al módulo *talker* en tiempo ideal τ:

$$\frac{d\vec{m}}{d\tau} = C_T\Lambda_T\Pi_T(\vec{m}, \tau)\vec{m}(\tau) = \xi(\vec{m}, \tau). \tag{8}$$

Sin embargo, la activación de las *gates* depende del tiempo local $\tau' = Z(\tau)$ (1) y no de τ, por lo que la dinámica correcta es:

$$\frac{d\vec{m}}{d\tau'} = \xi(\vec{m}, \tau'). \tag{9}$$

Reconciliando ambas parametrizaciones se tiene:

$$\frac{d\vec{m}}{d\tau} = \xi(\vec{m}, \tau')\frac{d\tau'}{d\tau}. \tag{10}$$

Para el caso afín $(\tau' = x_0 + (1 + y_0)\tau)$,

$$\frac{d\tau'}{d\tau} = 1 + y_0; \tag{11}$$

para el caso general $(\tau' = x(\tau) + \tau)$,

$$\frac{d\tau'}{d\tau} = \frac{dx}{d\tau} + 1. \tag{12}$$

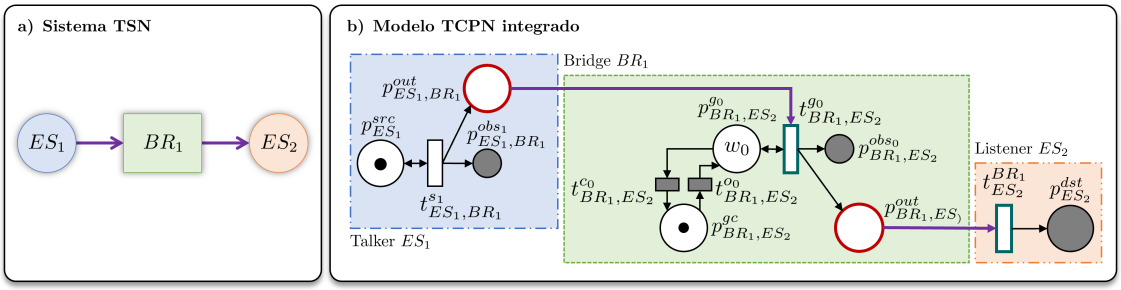


Fig. 5: Un sistema TSN sencillo que consiste de un *talker* (ES_1), un *listener* (ES_2) y un *bridge* (BR_1). Los módulos TCPN son interconectados por un arco (morado) desde los lugares de salida (rojo) $p^{out}_{ES_1,BR_1}$ en el *talker* y $p^{out}_{BR_1,ES_2}$ en el *bridge* hacia las transiciones de entrada (azul) $t^{g_0}_{BR_1,ES_2}$ en el *bridge* y $t^{BR_1}_{ES_2}$ en el *listener*, respectivamente.

En (10) no es necesario modificar estructuralmente la ecuación de estado, pues basta con multiplicar la dinámica por la derivada del tiempo local de cada nodo respecto al tiempo global.

Finalmente, los componentes de fase pueden convertirse en componentes de frecuencia dividiendo y viceversa integrando [9]. En consecuencia, por simplicidad y a efectos de simulación, el componente de ruido de fase implícito en (12) se construye integrando ruido de frecuencia de tipo caminata aleatoria, generado mediante el algoritmo de diferenciación fraccional de Hosking [24], según lo establecido en [18].

V. Banco de Pruebas

El banco de pruebas consiste en un entorno controlado, que integra un simulador basado en el modelo TCPN de la Sec. IV-B y un conjunto de métodos para pruebas de estrés. Se omiten las pilas de protocolos completas y el comportamiento específico del *hardware*, para evitar una complejidad innecesaria a nivel del sistema. El diseño es modular: a partir de la descripción de la TSN y la planificación de sus flujos, esta herramienta construye automáticamente el modelo dinámico, ejecuta las pruebas y presenta los resultados mediante visualizaciones interactivas y un registro reproducible. Una interfaz gráfica actualmente en desarrollo facilitará la interacción del usuario con el banco de pruebas.

A. Entradas y parámetros de simulación

El banco de pruebas recibe como entradas la topología $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ de la TSN, las especificaciones de los flujos y la planificación a analizar, calculada sobre la base de un TE estimado y generada por algoritmos externos. A partir de ellas, el simulador deriva automáticamente las ecuaciones diferenciales del modelo y adopta el hiperperiodo H (definido como el mínimo común múltiplo de los periodos individuales de todos los flujos) como ciclo de red.

Para permitir la exploración sistemática del comportamiento de la TSN, el banco de pruebas expone varios parámetros de simulación que se pueden ajustar de forma independiente, como es el caso del tiempo de inicio de transmisión, el reloj local y el intervalo de sincronización.

A.1 Tiempo de inicio de transmisión

El inicio efectivo de una transmisión dentro de las ventanas planificadas puede variar según el tipo de nodo. En el *talker*, el momento de la transmisión del flujo es configurable, lo que permite que las transmisiones comiencen después de lo previsto. Los *bridges* pueden introducir retrasos de procesamiento que desplazan el inicio de la transmisión dentro de la ventana. El modelo TCPN captura estos efectos aplicando compensaciones configurables a los tiempos de activación de las transiciones correspondientes. El TAS se implementa conforme a [1], con la particularidad de que opera en función de la disponibilidad real de la trama, es decir, el controlador abre una *gate* únicamente cuando la trama está lista para su transmisión —nunca antes del tiempo de apertura especificado en la entrada GCL correspondiente— y la cierra al completarse la misma. Esto permite al controlador absorber desviaciones temporales moderadas sin violar las restricciones de la planificación.

A.2 Parámetros de reloj local

El usuario especifica x_0 e y_0 por nodo, conforme al modelo de la Sec. II-A, con un componente de ruido de frecuencia de tipo caminata aleatoria. Recordemos que los relojes locales se capturan en la TCPN escalando las tasas de disparo de las transiciones correspondientes (ver Sec. IV-C).

A.3 Intervalo de sincronización

El parámetro $logSyncInterval \in [-24, 24]$, definido en [2], determina el intervalo de sincronización como $syncInterval = 2^{logSyncInterval}$ segundos y, a partir de este último, el número de ciclos simulados

$$\mathcal{H} = \left\lfloor \frac{syncInterval}{\mathcal{H}} \right\rfloor . \tag{13}$$

Por defecto, $logSyncInterval = -3$ (i.e., 125 ms).

A.4 Parámetros para pruebas de estrés

El módulo de pruebas de estrés incorpora tres métodos complementarios, descritos en la Sec. V-C: a) *adversario*, b) *por pares* —ambos propuestos en este trabajo— y c) *Monte Carlo* [25]. El usuario configura los rangos máximos de x_0 e y_0 (denotados como $\bar{x}_0$ e $\bar{y}_0$) comunes a todos los métodos. Para

el método por pares, se especifica la resolución de la cuadrícula $X \times Y$ de puntos en el espacio de desviaciones relativas (x_0^{rel}, y_0^{rel}), donde $x_0^{rel} = x_0^b - x_0^a$ e $y_0^{rel} = y_0^b - y_0^a$ para cada par $(a,b) \in \mathcal{E}$. Para el Monte Carlo, se especifica el número de semillas K.

B. Métricas de rendimiento

El banco de pruebas proporciona diferentes métricas de rendimiento que caracterizan la calidad de las planificaciones en condiciones operativas variables y con perturbaciones controladas.

La simulación principal calcula dos métricas por transmisión. Sea q la *gate* y $(a,b) \in \mathcal{E}$ el enlace correspondiente:

- El *Transmission Offset* (TO) cuantifica la desviación cuadrática en tiempo global entre el inicio de transmisión real y el planificado,

$$\mathrm{TO}^q_{(a,b)} = (Tx^{real}_{start} - Tx^{sched}_{start})^2. \tag{14}$$

- El *Remaining Delay Tolerance* (RDT) mide el margen de tiempo desde la finalización de la transmisión hasta el cierre de la ventana planificada, evaluado en el reloj local,

$$\mathrm{RDT}^q_{(a,b)} = window^{sched}_{end} - Tx^{real}_{end}. \tag{15}$$

  Un valor $\mathrm{RDT}^q_{(a,b)} < 0$ corresponde a una violación de ventana (Sec. II-C).

A partir del ciclo del primer fallo h^*, el banco de pruebas emite automáticamente la recomendación

$$logSyncInterval^* = \lceil \log_2(h^* \cdot H) \rceil, \tag{16}$$

que permite al usuario disminuir el intervalo de sincronización con el tT sin replanificación. Este criterio es conservador, pues garantiza que la red se sincronice antes de alcanzar el ciclo de primer fallo observado.

Cuando el primer fallo ocurre en el primer ciclo, h^* no está definido y la fórmula anterior no es aplicable. En este caso, la recomendación se calcula a partir de la duración de la ventana del enlace más crítico y las desviaciones relativas entre sus nodos:

$$\tau^*_{sync} = \frac{w^q_{(a,b)} - |x_0^{rel}|}{|y_0^{rel}|}, \tag{17}$$

$$logSyncInterval^* = \left\lfloor \log_2(\tau^*_{sync}) \right\rfloor, \tag{18}$$

donde $w^q_{(a,b)} = window^{sched}_{end} - window^{sched}_{start}$.

La violación de ventana es, además, un indicador común a todos los métodos para pruebas de estrés, cuyas métricas específicas se resumen en la Tabla II y se formalizan en la Sec. V-C.

En versiones futuras se prevé incorporar métricas adicionales, como el *jitter* de llegada y la tasa de violación por flujo y la utilización del enlace crítico.

C. Pruebas de estrés

El módulo evalúa la robustez mediante tres métodos complementarios. El método *adversario* identifica el peor caso absoluto, el método *por pares* caracteriza la región de operación segura de cada enlace de forma aislada y el método *Monte Carlo* estima la probabilidad global de fallo bajo condiciones aleatorias correlacionadas entre todos los nodos. Los tres métodos emplean el modelo afín de TE en modo libre (3) para preservar la interpretabilidad y reproducibilidad de los resultados. Su uso combinado permite identificar si la planificación es vulnerable, dónde y bajo qué condiciones.

Tabla II: Métricas por método para pruebas de estrés. γ: intensidad del estrés; $\vec{\zeta}$: vector de signos de los relojes; $h(\gamma, \vec{\zeta})$: ciclo del primer fallo; $\mathcal{C}_{\mathrm{mín}}(\gamma)$, $\mathcal{C}_{\mathrm{máx}}(\gamma)$: envolvente de peor y mejor caso; γ_c: umbral crítico; P(fallo): probabilidad global de fallo.

Método	**Métricas**
Adversario	▪ Ciclo del primer fallo $h(\gamma, \vec{\zeta})$ ▪ Envolvente $\mathcal{C}_{\mathrm{mín}}(\gamma)$, $\mathcal{C}_{\mathrm{máx}}(\gamma)$ ▪ Umbral crítico γ_c
Por pares	▪ Zona segura en (x_0^{rel}, y_0^{rel}) ▪ Porcentaje de puntos con fallo ▪ Par crítico (cuello de botella)
Monte Carlo	▪ P(fallo) ▪ Mediana y P10 del primer fallo ▪ Nodos adyacentes causales

C.1 Adversario

Evalúa el peor caso asignando signos opuestos a los parámetros de reloj de pares adyacentes y barriendo una intensidad $\gamma \in [0,1]$. El modelo del TE del nodo a queda parametrizado como

$$x_a(t) = \gamma \cdot \zeta_a \cdot (\bar{x}_0 + \bar{y}_0 \tau), \tag{19}$$

donde $\gamma = 0$ corresponde a relojes perfectos y $\gamma = 1$ al máximo estrés configurado. En modo *crítico* se analiza cada par $(a,b) \in \mathcal{E}$ con $\zeta_a = -1$, $\zeta_b = +1$ y los demás nodos en cero. En modo *completo* se barren todas las $2^{|\mathcal{V}|}$ combinaciones $\vec{\zeta} \in \{-1,+1\}^{|\mathcal{V}|}$, obteniendo la envolvente

$$\mathcal{C}_{\mathrm{mín}}(\gamma) = \min_{\vec{\zeta}} h(\gamma, \vec{\zeta}), \quad \mathcal{C}_{\mathrm{máx}}(\gamma) = \max_{\vec{\zeta}} h(\gamma, \vec{\zeta}) \tag{20}$$

donde $h(\gamma, \vec{\zeta}) \in \{1, \ldots, \mathcal{H}+1\}$ es el ciclo del primer fallo bajo la asignación $\vec{\zeta}$ e intensidad γ, tomando el valor $\mathcal{H}+1$ si no se produce ninguna violación. Adicionalmente, escalando los valores x_0 e y_0 del usuario por γ se obtiene el umbral crítico

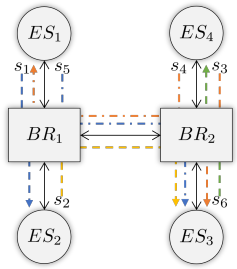
$$\gamma_c = \min \{\gamma \in [0,1] : h(\gamma) < \mathcal{H}+1\}. \tag{21}$$

C.2 Por pares

Caracteriza la región de operación segura de cada enlace $(a,b) \in \mathcal{E}$ mediante un barrido sobre (x_0^{rel}, y_0^{rel}) dentro de los rangos $[-\bar{x}_0, \bar{x}_0]$ y $[-\bar{y}_0, \bar{y}_0]$, en una cuadrícula de $X \times Y$ puntos, manteniendo los demás nodos en condición ideal. Permite identificar el par responsable del cuello de botella, aunque no captura efectos combinados entre múltiples nodos.

C.3 Monte Carlo

Muestrea aleatoriamente K configuraciones de (x_0, y_0) para todos los nodos simultáneamente dentro



Flujo	Tx (ns)	P = D (μs)
s_1	4000	100
s_2	12000	100
s_3	6400	100
s_4	4000	100
s_5	4000	100
s_6	11200	50

Fig. 6: Un sistema TSN y las especificaciones de sus flujos.

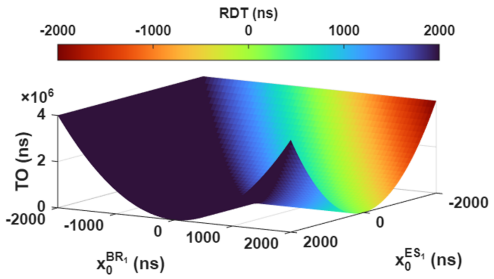


Fig. 7: TO y RDT resultantes de una planificación sin esperas.

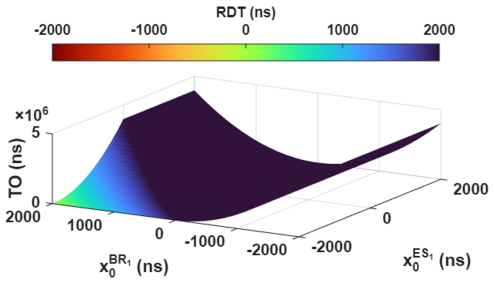


Fig. 8: TO y RDT resultantes de una planificación que permite espera en cola de 2 μs en BR_1.

de los rangos $\bar{x}_0$ y $\bar{y}_0$, incorporando el componente de ruido conforme al modelo de la Sec. II-A. Estima la probabilidad global de fallo y captura los efectos combinados de las desviaciones de los relojes de todos los dispositivos de la TSN. La semilla es fija para garantizar la reproducibilidad.

VI. Casos de Estudio

Evaluamos el banco de pruebas mediante dos casos de estudio sobre un sistema TSN de 1 Gb/s con intervalo de sincronización de 125 ms [2] y tT externo ideal, empleando las métricas y métodos de la Sec. V. El primero valida la consistencia del banco de pruebas con resultados previos [4], usando el modelo afín del reloj; el segundo muestra la aportación principal de este trabajo, aplicando el modelo no lineal y las tres pruebas de estrés a una red de mayor escala.

Para el primer caso, consideramos una TSN con un único flujo (tiempo de transmisión $Tx = 3$ μs, periodo y *deadline* $P = D = 20$ μs) sobre la topología de la Fig. 5. Las desviaciones de frecuencia se fijan en sus valores extremos $y_0^{ES_1} = +100$ ppm e $y_0^{BR_1} = -100$ ppm (combinación crítica según la Tabla I), mientras que $x_0^{ES_1}$ y $x_0^{BR_1}$ se barren en $[-2\ \mu\text{s}, 2\ \mu\text{s}]$. Se evalúan dos planificaciones calculadas con el algoritmo de [26] suponiendo una cota del TE estimado de 2 μs. La primera impone una restricción de no espera entre nodos consecutivos. La segunda permite una espera en cola de 2 μs en BR_1.

El segundo caso considera la TSN de la Fig. 6 con seis flujos e hiperperiodo $H = 100$ μs. Los parámetros de reloj de cada nodo, consistentes con [2], son:

ES_1: $x_0^{ES_1} = +32$ ns, $y_0^{ES_1} = +37$ ppm;
ES_2: $x_0^{ES_2} = -18$ ns, $y_0^{ES_2} = -03$ ppm;
ES_3: $x_0^{ES_3} = +54$ ns, $y_0^{ES_3} = +12$ ppm;
ES_4: $x_0^{ES_4} = -41$ ns, $y_0^{ES_4} = -24$ ppm;
BR_1: $x_0^{BR_1} = +23$ ns, $y_0^{BR_1} = -71$ ppm;
BR_2: $x_0^{BR_2} = -67$ ns, $y_0^{BR_2} = +55$ ppm.

El ruido de fase se obtiene integrando realizaciones independientes de ruido de frecuencia tipo caminata aleatoria con densidad espectral $S(f) \propto f^{-2}$ y varianza $Q_d = (10\ \text{ns})^2$, generadas con semilla distinta por nodo mediante el método de diferenciación fraccional [24] siguiendo [18]. Las pruebas de estrés utilizan $|x_0| \leq \bar{x}_0 = 1$ μs (conforme a la precisión de sincronización garantizada por [2] en estado estacionario), $|y_0| \leq \bar{y}_0 = 100$ ppm [2] y $K = 50$ configuraciones para el Monte Carlo. Se considera una única planificación con ventanas esparcidas a lo largo de H, que permite esperas en cola, calculada con el algoritmo de [27], implementado en la herramienta TSNKit [28]; ninguno de los algoritmos disponibles en TSNKit contempla una cota del TE distinta de cero, práctica habitual en este tipo de estudios.

A. Primer caso: modelo afín de reloj

Las Fig. 7 y 8 muestran el TO resultante en BR_1 y el RDT codificado en el mapa de calor: el azul oscuro indica que se conserva todo el margen, el verde su agotamiento completo (RDT = 0) y los tonos naranja y rojo oscuro denotan violaciones de ventana (RDT < 0). En la Fig. 7, el TO crece considerablemente con las desviaciones de reloj y la región factible se limita a una zona estrecha en torno a valores pequeños de x_0, lo que pone de relieve la sensibilidad de las planificaciones ajustadas ante desviaciones moderadas. La Fig. 8 muestra la misma tendencia cualitativa, pero con una región de operación factible sustancialmente ampliada: la espera en cola de 2 μs introduce holgura adicional que amortigua las violaciones observadas en el cuadrante $(-,+)$ de la Fig. 7, donde $x_0^{BR_1} - x_0^{ES_1} > 2000$ ns, aunque a costa de aumentar la latencia extremo a extremo.

Ambas planificaciones son factibles según un verificador convencional; sin embargo, aunque fueron calculadas con la misma cota del TE, el TO y el RDT revelan que difieren significativamente en su tolerancia a desviaciones de reloj reales, información que un verificador convencional no puede proporcionar. Este resultado ilustra que incluso bajo el modelo afín, que es la aproximación más simple del reloj, las desviaciones de reloj pueden comprometer la planificación. El segundo caso aborda qué ocurre cuando el comportamiento del reloj es no lineal, como sucede en el mundo real, sobre una red de mayor escala.

B. Segundo caso: modelo no lineal de reloj

Por un lado, vamos a analizar la simulación principal, que evalúa el escenario real bajo la configuración de relojes descrita. Por el otro, las pruebas de estrés, que generalizan las conclusiones a un rango más amplio de condiciones. En el primer caso de estudio, los indicadores TO y el RDT caracterizan el margen disponible en la ventana según la holgura considerada en la planificación. En este segundo caso, la holgura de la planificación es cero, por lo que se indica directamente la violación de ventana y el ciclo en que ocurre.

La simulación principal revela violaciones de ventana desde el primer ciclo de red en los flujos s_2 y s_4, en los enlaces (BR_1, BR_2) y (BR_1, ES_1), respectivamente, con llegadas tardías de 35 ns y 78 ns. El fallo en (BR_1, BR_2), además, ocasiona violación de ventana en el enlace (BR_2, ES_3). El flujo s_5 falla en el ciclo 5 en (BR_2, ES_3), mientras que el flujo s_6 lo hace en el ciclo 29 en (BR_2, ES_4), donde la primera trama llega 2 ns tarde. Los flujos s_1 y s_3 no presentan violaciones de ventana. Los fallos se deben a la conjunción de tres factores: el planificador no contempla holgura para el TE, algunas de las ventanas de los flujos afectados son consecutivas (sin espera en cola) y la combinación de signos de los parámetros de reloj es crítica, generando una desviación relativa entre nodos adyacentes que, sumada al ruido acumulado, provoca violaciones de ventana. Estos resultados sólo caracterizan el comportamiento bajo una configuración específica de relojes; las pruebas de estrés permiten generalizar las conclusiones.

El método adversario completo (Fig. 9) muestra que 47 de las 64 combinaciones de signos producen fallo; las 17 restantes son no críticas (Tabla I). De ellas, 46 fallan en $h = 0$ para todo $\gamma > 0$, pues la desviación inicial $\gamma \cdot \zeta_i \cdot x_0$ es suficiente para causar violación inmediata. La excepción es $\vec{\zeta} = [-1, +1, +1, +1, +1, +1]$, que sólo falla para $\gamma \geq 0{,}3$, con primer fallo en $h = 867$ para $\gamma = 0{,}3$ y en $h = 190$ para $\gamma = 1$, explicando el perfil decreciente de $\mathcal{C}_{\text{máx}}$. El método por pares (Fig. 10) identifica los enlaces (BR_1, BR_2) y (BR_2, BR_1) como cuellos de botella (99 % de fallo), frente al 70 % del resto. La región segura se concentra en torno a desviaciones relativas nulas, mientras que los fallos tempranos dominan el cuadrante $(x_0^{rel} > 0, y_0^{rel} > 0)$, consistente con el caso crítico $(-,+)$ de la Tabla I. El método Monte Carlo estima $P(\text{fallo}) = 100\,\%$ con mediana y P10 en $h = 0$.

Los tres métodos se complementan entre sí: el adversario revela que la vulnerabilidad es prácticamente independiente de la intensidad del estrés; el método por pares localiza el cuello de botella en los enlaces entre *bridges*; y el método Monte Carlo confirma que no existe configuración de relojes dentro de los rangos estándar que evite el fallo. Los tres en conjunto señalan que la planificación analizada no es robusta a desviaciones de reloj dentro de los rangos establecidos por los estándares [2, 7]. Aplicando (17) y (18) al enlace más crítico (BR_1, BR_2), el banco de pruebas

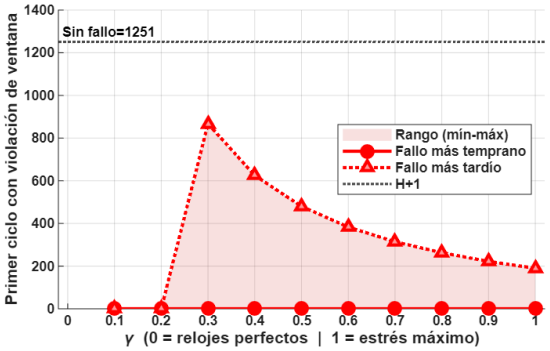


Fig. 9: Envolvente del adversario completo: fallo más temprano ($\mathcal{C}_{\text{mín}}$) y más tardío ($\mathcal{C}_{\text{máx}}$) sobre las combinaciones con fallo en función de γ. La línea punteada indica el total de ciclos simulados sin fallo ($\mathcal{H} + 1 = 1251$).

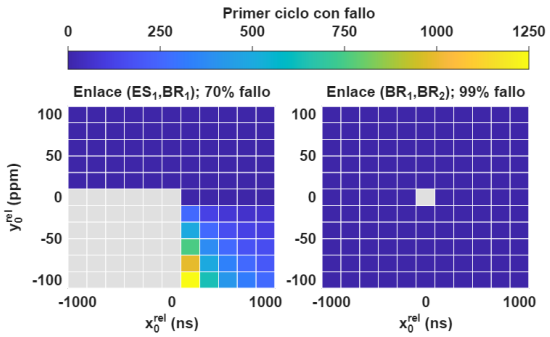


Fig. 10: Mapa de calor del método por pares: primer ciclo con fallo en el espacio (x_0^{rel}, y_0^{rel}). Gris: sin fallo. Izquierda: enlace (ES_1, BR_1) (70 % de fallo; los enlaces (ES_2, BR_1), (ES_3, BR_2) y (ES_4, BR_2) presentan el mismo comportamiento). Derecha: enlace crítico (BR_1, BR_2) (99 % de fallo; mismo comportamiento en el enlace (BR_2, BR_1)).

recomienda $logSyncInterval^* = -14$ ($2^{-14} \approx 61\ \mu$s). Este valor es menor que H: el intervalo de sincronización requerido generaría una sobrecarga de mensajes gPTP en la red. La solución más adecuada, por tanto, consiste en introducir holgura para el TE en la planificación de ventanas de transmisión.

VII. Conclusiones

Presentamos un banco de pruebas modular basado en un modelo TCPN para evaluar la validez de planificaciones IEEE 802.1Qbv ante variaciones del TE lineales y no lineales. A diferencia de los verificadores existentes, esta herramienta cuantifica la tolerancia de una planificación a desviaciones de reloj reales mediante pruebas de estrés complementarias: el adversario identifica el peor caso absoluto y el umbral crítico de la configuración actual; el método por pares localiza el enlace más vulnerable y caracteriza su región de operación segura; el método Monte Carlo estima la probabilidad global de fallo capturando los efectos combinados de todos los nodos. Adicionalmente, nuestra herramienta emite automáticamente una recomendación del intervalo de sincronización a partir del ciclo del primer fallo observado, orientando la configuración del sistema sin necesidad de replanificar y determinando los puntos y condiciones de vulnerabilidad de una planificación TSN.

Se abordan dos escenarios complementarios como casos de estudio. En el primero, los indicadores TO

y RDT del banco de pruebas cuantifican con precisión que dos planificaciones con la misma holgura total difieren significativamente en su tolerancia a desviaciones de reloj. En el segundo, los tres métodos coinciden en que una planificación sin margen para el TE no es robusta ante ninguna configuración de relojes dentro de los rangos estándar, identificando los enlaces entre *bridges* como cuello de botella y descartando el ajuste del intervalo de sincronización como mejor solución.

Extensiones en curso incorporarán métricas adicionales y extenderán el análisis a múltiples intervalos de sincronización, clarificando si el ajuste del reloj en el instante de resincronización introduce violaciones transitorias. Hasta donde sabemos, esta herramienta analiza aspectos no abordados en estudios previos. El modelo TCPN constituye una herramienta complementaria en verificación de planificaciones, que permitirá ajustes automatizados de las ventanas de transmisión subóptimas o no válidas, contribuyendo a acelerar el ciclo de diseño, despliegue y actualización de sistemas TSN.

Agradecimientos

Financiación MICIU/AEI/10.13039/501100011033 (PID2022-136454NB-C22) y Gobierno de Aragón (grupo T58_23R), España. Torres Macías cuenta con financiación SECIHTI (1141966), México.

Referencias

[1] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 25: Enhancements for Scheduled Traffic," *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pp. 1–57, 2016, https://doi.org/10.1109/IEEESTD.2016.8613095.

[2] "IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications," *IEEE Std 802.1AS-2025 (Revision of IEEE Std 802.1AS-2020)*, pp. 1–491, 2025, https://doi.org/10.1109/IEEESTD.2025.11302966.

[3] A.G. Torres-Macías et al., "Modeling Time-Sensitive Networking Using Timed Continuous Petri Nets," *IFAC-PapersOnLine*, vol. 58, no. 1, pp. 300–305, 2024, 17th IFAC WODES 2024, https://doi.org/10.1016/j.ifacol.2024.07.051.

[4] A. Torres et al., "TSN Schedule Verification with Parametric Variation Based on TCPN," in *2026 Int. Workshop Discrete Event Syst.*, 2026, Accepted for publication.

[5] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008)*, pp. 1–499, 2020, https://doi.org/10.1109/IEEESTD.2020.9120376.

[6] IEC and IEEE Standards Associations, "IEC/IEEE 60802: Time-Sensitive Networking Profile for Industrial Automation," https://1.ieee802.org/tsn/iec-ieee-60802/, 2025.

[7] International Telecommunication Union – Telecommunication Standardization Sector, "ITU-T G.8260: Definitions and terminology for synchronization in packet networks," 2022, https://www.itu.int/rec/T-REC-G.8260/en.

[8] International Telecommunication Union – Telecommunication Standardization Sector, "ITU-T G.810: Definitions and terminology for synchronization networks," 1996, https://www.itu.int/rec/T-REC-G.810/en.

[9] W. Riley and D. Howe, "Handbook of Frequency Stability Analysis," 2008, https://www.nist.gov/publications/handbook-frequency-stability-analysis.

[10] L. Thomas and J-Y. Le Boudec, "On Time Synchronization Issues in Time-Sensitive Networks with Regulators and Nonideal Clocks," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 2, 06 2020, https://doi.org/10.1145/3392145.

[11] N.M. Freris and P.R. Kumar, "Fundamental limits on synchronization of affine clocks in networks," in *2007 IEEE Conf. Decis. Control*, 2007, pp. 921–926, https://doi.org/10.1109/CDC.2007.4434546.

[12] R. Exel et al., "Asymmetry Mitigation in IEEE 802.3 Ethernet for High-Accuracy Clock Synchronization," *IEEE Trans. Instrum. Meas.*, vol. 63, no. 3, pp. 729–736, 2014, https://doi.org/10.1109/TIM.2013.2280489.

[13] A.K. Karthik and R.S. Blum, "Recent Advances in Clock Synchronization for Packet-Switched Networks," *Found. Trends Signal Process.*, vol. 13, no. 4, pp. 360–443, 09 2020, https://doi.org/10.1561/2000000108.

[14] F. Dang et al., "A Survey on Clock Synchronization in the Industrial Internet," *J. Comput. Sci. Technol.*, vol. 38, no. 1, pp. 146–165, 2023, https://doi.org/10.1007/s11390-023-2908-4.

[15] A. Ghosh et al., "Importance of Realistic Considerations of Time Synchronization for TAS in TSN Networks," in *2023 Int. Symp. Netw., Comput. Commun.*, 2023, pp. 1–6, https://doi.org/10.1109/ISNCC58260.2023.10323693.

[16] A. Ghosh et al., "Multi-Stream TSN Gate Control Scheduling in the Presence of Clock Synchronization," in *Procs. of the 26th Int. Conf. Distrib. Comput. Netw.*, New York, NY, USA, 2025, ICDCN '25, p. 11–20, Association for Computing Machinery, https://doi.org/10.1145/3700838.3700847.

[17] D.W. Allan, "Time and Frequency (Time-Domain) Characterization, Estimation, and Prediction of Precision Clocks and Oscillators," *IEEE Trans. Ultrason., Ferroelectr., Freq. Control*, vol. 34, no. 6, pp. 647–654, 1987, https://doi.org/10.1109/T-UFFC.1987.26997.

[18] N.J. Kasdin and T. Walter, "Discrete simulation of power law noise (for oscillator stability evaluation)," in *Procs. of the 1992 IEEE Freq. Control Symp.*, 1992, pp. 274–283, https://doi.org/10.1109/FREQ.1992.270003.

[19] S. Shinde, "Review of Oscillator Phase Noise Models," *Lect. Notes Eng. Comput. Sci.*, vol. 2210, 03 2014.

[20] K. Brunner et al., "Practical Aspects of Cross-Vendor TSN Time Synchronization Using IEEE 802.1AS," *J. Sens. Actuator Netw.*, vol. 14, no. 4, 2025, https://doi.org/10.3390/jsan14040067.

[21] T. Murata, "Petri nets: Properties, analysis and applications," *Procs. of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989, https://doi.org/10.1109/5.24143.

[22] L. Recalde and M. Silva, "Petri Nets fluidification revisited: Semantics and steady state," *J. Eur. Syst. Autom.*, vol. 35, 2001.

[23] M. Silva and L. Recalde, "On Fluidification of Petri Nets: From Discrete to Hybrid and Continuous Models," *Ann. Rev. in Control*, vol. 28, no. 2, pp. 253–266, 2004, https://doi.org/10.1016/j.arcontrol.2004.05.002.

[24] J.R.M. Hosking, "Fractional Differencing," *Biometrika*, vol. 68, no. 1, pp. 165–176, 04 1981, https://doi.org/10.1093/biomet/68.1.165.

[25] R.Y. Rubinstein and D.P. Kroese, *Simulation and the Monte Carlo Method*, John Wiley & Sons, Hoboken, NJ, 3 edition, 2016.

[26] A.G. Torres-Macías et al., "Optimal Fast IEEE802.1Qbv Incremental Scheduling," Submitted, under revision, 2025.

[27] R. Serna Oliver et al., "IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding," in *2018 IEEE Real-Time Embedded Technol. Appl. Symp.*, 2018, pp. 13–24, https://doi.org/10.1109/RTAS.2018.00008.

[28] C. Xue et al., "Real-Time Scheduling for 802.1Qbv Time-Sensitive Networking (TSN): A Systematic Review and Experimental Study," in *2024 IEEE Real-Time Embedded Technol. Appl. Symp.* IEEE, 2024, pp. 108–121.

Generación adaptativa de rutas de telemetría INT-MD en redes congestionadas

Pablo Gomariz Martínez[1], Francisco Manuel Delicado Martínez[1] y Gabriel Cebrián-Márquez[2]

Resumen— **El despliegue de redes 5G y la estrecha integración del borde de la red con la nube exigen una monitorización precisa y en tiempo real para satisfacer los estrictos requisitos de las aplicaciones modernas. La telemetría en banda (INT) surge como una solución eficiente frente a los métodos tradicionales, aunque plantea desafíos críticos de sobrecarga y obsolescencia de los datos. En este contexto, el presente trabajo analiza diversas estrategias de generación de rutas de telemetría basadas en algoritmos de búsqueda de caminos eulerianos. A diferencia de las aproximaciones estáticas de la literatura, este estudio propone un método de cálculo de rutas que tiene en cuenta la longitud de los segmentos y la edad de la información (AoI - *Age of Information*). Los resultados muestran que este enfoque permite obtener rutas de telemetría concentradas en segmentos de red según su nivel de congestión, facilitando la monitorización adaptativa y el control de la AoI, especialmente en entornos de computación en el borde, donde las aplicaciones tienen requisitos estrictos de latencia.**

Palabras clave— **Telemetría, telemetría en banda, INT, P4, redes definidas por software, SDN, monitorización.**

I. Introducción

LAS redes de comunicaciones se encuentran en constante evolución para satisfacer los requisitos de ultra baja latencia y conectividad masiva de aplicaciones como la realidad aumentada y virtual, el Internet de las cosas, la Industria 4.0 y las ciudades inteligentes, impulsadas por las redes 5G y de próxima generación. Este contexto exige infraestructuras escalables, dinámicas y resilientes, lo que ha consolidado la *softwarización* como pilar fundamental en el diseño de las arquitecturas de red.

En este sentido, las redes definidas por software (SDN, *Software-Defined Networking*) han transformado la gestión tradicional de red al desacoplar los planos de control y de datos, permitiendo su programabilidad y automatización, y dando lugar a una gestión más eficiente de los recursos. Sin embargo, estas capacidades dependen de una monitorización precisa y en tiempo real del estado de la red. La telemetría se convierte así en un componente esencial para la detección temprana de anomalías, la optimización del rendimiento y la toma de decisiones informadas en entornos altamente dinámicos, especialmente en un entorno donde la inteligencia artificial está cobrando un papel cada vez más relevante.

[1]Instituto de Investigación en Informática de Albacete (I3A), Universidad de Castilla-La Mancha, e-mail: {pablo.gomariz, francisco.delicado}@uclm.es.

[2]Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha, e-mail: gabriel.cebrian@uclm.es.

Los enfoques tradicionales de monitorización, tales como *Simple Network Management Protocol* (SNMP), se basan en modelos de sondeo que introducen sobrecarga de señalización y latencias de medición, especialmente en redes de gran escala. Estas limitaciones dificultan la obtención de información en tiempo real, lo que resulta insuficiente para aplicaciones con requisitos estrictos de latencia y fiabilidad.

Para abordar estas limitaciones, surge la telemetría en banda (INT, *In-Band Network Telemetry*), que permite recopilar información directamente en el plano de datos mediante dispositivos programables, típicamente utilizando P4 [1]. INT permite una monitorización de grano fino al insertar metadatos en los paquetes a medida que atraviesan la red, incluyendo métricas como la latencia por salto, la ocupación de colas o los identificadores de puertos, lo que reduce la dependencia del plano de control.

El modo *INT Metadata* (INT-MD) [2] constituye la implementación estándar de INT, e implementa este enfoque mediante la inserción progresiva de información en los paquetes. Un nodo origen (*INT Source*) selecciona los flujos a monitorizar e indica las métricas a recolectar, mientras que los nodos intermedios (*INT Transit*) añaden sus propios datos. Como resultado, el nodo destino (*INT Sink*) recibe una visión completa del recorrido del paquete sin generar tráfico adicional de control. Este esquema se puede emplear con sondas sintéticas generadas por los propios nodos origen para controlar de forma flexible y dinámica la recolección de telemetría.

Sin embargo, este enfoque introduce desafíos relevantes. La inclusión de metadatos incrementa el tamaño de los paquetes y el consumo de recursos en los dispositivos de red, por lo que su eficiencia depende de una planificación adecuada de las sondas y sus trayectorias. Además, factores como la edad de la información (AoI, *Age of Information*) y las limitaciones en el tamaño de los paquetes (MTU, *Maximum Transmission Unit*) condicionan el diseño de los mecanismos de recolección. Si bien los recorridos largos reducen el número de sondas, estos pueden generar información obsoleta. Por el contrario, recorridos más cortos incrementan la sobrecarga.

La literatura existente ha abordado este problema principalmente desde una perspectiva de grafos, centrándose en la generación de caminos que cubren toda la red en base a múltiples objetivos, como la longitud de los mismos o la repetición de enlaces. Sin embargo, estas aproximaciones suelen ignorar el estado dinámico de la red y de los dispositivos. En

escenarios reales, las zonas congestionadas requieren una monitorización más frecuente que aquellas con comportamiento estable, lo que pone de manifiesto la necesidad de estrategias conscientes del estado de la red y que sean capaces de adaptarse a ella.

En este contexto, este trabajo analiza distintas estrategias de generación de caminos de telemetría INT-MD mediante algoritmos de búsqueda de caminos eulerianos en grafos. Las rutas de telemetría se evaluarán en cuanto a la longitud en número de saltos, la cantidad de rutas necesarias para recorrer una red, y la localización de las rutas en zonas de congestión de la red. La contribución principal es avanzar hacia mecanismos adaptativos que permitan ajustar la frecuencia de sondas y controlar la AoI en función de las necesidades de las aplicaciones, especialmente en entornos de computación en el borde.

El resto del documento se organiza como sigue. La Sección II revisa el estado del arte en telemetría INT y generación de caminos. La Sección III describe los algoritmos considerados, mientras que la Sección IV presenta su evaluación. Por último, la Sección V concluye este trabajo.

II. Estado del arte

En la bibliografía ya existen trabajos que tratan el problema de la optimización de las rutas que han de seguir los mensajes INT. Esta optimización puede centrarse en varios objetivos: minimizar el retardo que un mensaje INT sufre desde que se introduce en la red hasta que es enviado al colector de telemetrías, la adaptación de las rutas al tamaño del máximo MTU que los mensajes INT pueden tener, minimizar el número de nodos *source* y *sink* (nodos que inyectan mensajería INT y que la extraen de la red respectivamente) o minimizar el número de rutas con las que somos capaces de recorrer todos los enlaces de nuestra red.

La base de todos estos trabajos es el problema del cartero chino [3] y, más concretamente, la búsqueda de un ciclo euleriano como solución óptima a dicho problema. Cuando existe el ciclo euleriano en un grafo se dice que dicho grafo es euleriano. Un grafo dirigido es euleriano y, por tanto, se puede obtener un ciclo euleriano si y solo si cumple alguna de estas propiedades [4]:

- Cada vértice tiene el mismo grado de entrada in_{deg} que de salida out_{deg}, y todos los vértices con grado distinto de cero pertenecen a un único componente fuertemente conexo. De manera equivalente, un grafo dirigido tiene un ciclo euleriano si y solo si puede descomponerse en ciclos dirigidos disjuntos en aristas y todos sus vértices con grado distinto de cero pertenecen a un único componente fuertemente conexo. Esto es lo que se denomina un grafo dirigido conexo de grado par.
- A lo sumo, un vértice cumple que $out_{deg} - in_{deg} = 1$; a lo sumo, un vértice cumple que $in_{deg} - out_{deg} = 1$; todos los demás vértices cumplen que $in_{deg} = out_{deg}$, y todos los vértices con grado distinto de cero pertenecen a un único componente conexo del grafo no dirigido subyacente.

Así, en [5] se presenta un algoritmo para minimizar el número de rutas que recorren todos los enlaces de una red, así como el de solapamientos, es decir, el número de rutas que recorren el mismo enlace. Dado que los autores consideran la red como un grafo no dirigido, ello implica que determinadas topologías pueden presentar nodos de grado impar. Esta premisa obliga a los autores a preprocesar el grafo y dividirlo en subgrafos ahora sí eulerianos; asegurando así la existencia de un camino cíclico que los recorre sin solapamiento alguno. Es en el proceso del cálculo de los subgrafos donde aparecen los enlaces que serán recorridos por varias rutas. Una modificación de este algoritmo se presenta en [6] para proporcionar rutas más balanceadas en cuanto a longitud.

En [7] también se parte de la premisa de que la red se representa como un grafo no dirigido. Además, supone que no todos los nodos de una red son dispositivos P4 capaces de actuar como nodos *source/sink*, lo que limita el número de nodos desde los que puede comenzar o hacia los que puede finalizar una ruta de un mensaje INT. De esta forma, el trabajo propone las transformaciones que hay que realizar en el grafo inicial para que este se comporte como un grafo euleriano y asegurar la existencia de un camino cíclico que recorra todos sus enlaces. Otros trabajos que parten de la premisa de representar una red como un grafo unidireccional son [8] y [9]. Los autores proponen mecanismos para eliminar los nodos de grado impar y convertir el grafo en uno euleriano, asegurando así la existencia de un camino que recorre todos los enlaces. Tras esto, proponen un mecanismo para limitar el tamaño máximo de los mensajes INT, segmentando el camino euleriano calculado en función del MTU.

Por su parte, en [10], los autores proponen un método de programación lineal para determinar el número mínimo de ciclos con los que se pueden recorrer todos los enlaces de una topología de red. Esto presupone que un mensaje INT solo puede partir y finalizar su trayecto en el mismo nodo.

En [11], los autores tratan de reducir el *overhead* de la telemetría y evitar el retraso en la información recogida. El cálculo de las rutas se realiza de manera voraz, evaluando todas las posibles rutas entre cada par origen-destino de *hosts* de la red y buscando aquellas que pasen por todos los enlaces de la red. Otro trabajo que también emplea un algoritmo voraz para obtener rutas es [12]. Una vez obtenido el conjunto de rutas, se le aplican restricciones relativas a la ocupación de los enlaces que la componen, al

MTU y al tiempo que tardarán los mensajes INT en recorrerlas, para determinar si son elegibles o no.

Un planteamiento basado en el problema del viajante de comercio, que calcula la ruta en un grafo en la que cada nodo se visita una sola vez, se presenta en [13]. Tras calcular la ruta, esta se fragmenta mediante un algoritmo voraz para que los mensajes INT que la recorran no excedan el MTU. Hay que tener en cuenta que el problema del viajante de comercio no garantiza que se recorran todas las aristas de un grafo, por lo que este trabajo no asegura que los mensajes INT recorran todos los enlaces de la red.

La mayoría de los trabajos estudiados se basan en la premisa de que una red de interconexión puede representarse como un grafo no dirigido, lo que puede provocar la aparición de nodos de grado impar y, por lo tanto, la inexistencia de un ciclo euleriano y la necesidad de realizar cambios en el grafo para generar un grafo euleriano alternativo. Pero desde nuestro punto de vista, al ser los enlaces de toda red de comunicaciones bidireccionales, estas pueden representarse como grafos dirigidos en los que, para cada enlace de la red, un nodo tendrá una arista entrante y otra saliente. De este modo, toda red es un grafo dirigido de grado par y, por ello, con un ciclo euleriano que se puede obtener mediante los algoritmos de Fleury [14] o Hierholzer [15].

Lo que tampoco plantean los trabajos estudiados es cómo calcular las rutas de los mensajes INT para que estén ubicadas en zonas de la red con determinadas características, como el grado de congestión, el retardo de los enlaces, etc. Esto puede ser útil a la hora de adaptar la frecuencia de muestreo de los mensajes INT, ya que contaremos con rutas que recorran enlaces con un nivel de ocupación determinado.

III. Nuestra propuesta

En nuestra propuesta tratamos de evaluar los caminos eulerianos obtenidos, representando la red como grafos dirigidos de grado par, con diferentes algoritmos en base a la latencia (que influye en el AoI) y el MTU, ya que son los dos factores que más impacto tienen en la obtención de telemetría. Para ello, abordamos el problema en tres fases: el cálculo de la latencia de un salto de la red, el cálculo de los caminos eulerianos y la división de los caminos teniendo en cuenta el MTU de la red y la latencia límite establecida.

A. Latencia de rutas

Para calcular la latencia, consideramos el tiempo que tarda un paquete en atravesar la red, desde que la cabeza del paquete llega al puerto de entrada del primer dispositivo de red hasta que la cola del paquete abandona el puerto de salida del último dispositivo. La latencia total, T, se calcula como la suma de T_h y T_s, donde T_h es la latencia de cabeza y T_s es la latencia de serialización. La latencia de cabeza se define como el tiempo que tarda la cabeza del mensaje en atravesar la red. La latencia de serialización, por su parte, es el tiempo que tarda la cola en alcanzar a la cabeza y se calcula como L/b, es decir, el tiempo que tarda un paquete de longitud L en cruzar un canal con ancho de banda b [16].

Tabla I: Notación de la ecuación de T_{INT}

Símbolo	**Descripción**
T_h	Latencia de cabeza
T_s	Latencia de serialización
T_r	Latencia de enrutamiento
T_w	Tiempo de vuelo
(u_i, v_i)	Enlace i entre nodos $u \rightarrow v$
tr_{u_i}	Lat. de enrutamiento en nodo u_i
tq_{u_i}	Lat. de contención en nodo u_i
D_i	Distancia de enlace i
v_i	Velocidad de enlace i
L_i	Tam. de paquete enviado en enlace i
b_i	Ancho de banda del enlace i
M	Número de enlaces

La latencia de cabeza T_h es la suma de dos factores: el retardo del enrutador T_r, y el tiempo de vuelo T_w. El retardo del enrutador, T_r, se calcula de forma general como $n * tr$ para una red con n nodos y un retraso por enrutador de tr. A este cálculo podemos añadir un tiempo de contención tq, es decir, un retardo adicional debido a la competencia por los recursos de la red, de modo que $T_r = n * (tq + tr)$. Consideramos tq como el tiempo que un paquete permanece en la cola de un buffer de entrada de un enrutador hasta que es procesado y se calcula como $(q_{occ} + L)/qr$, donde q_{occ} es la ocupación existente en un buffer de entrada en un momento dado y L es el tamaño del paquete entrante. La velocidad a la que se procesan los datos en la cola qr será, como mínimo, la del enlace.

El tiempo promedio de vuelo es $T_w = D_{min}/v$, donde D_{min} es la distancia promedio y v la velocidad de propagación.

Con estos parámetros, la expresión para calcular la latencia de una ruta de telemetría será la indicada en la Ecuación 1, encontrándose sus símbolos definidos en la Tabla I.

$$\begin{aligned} T_{INT} &= T_h + T_s = T_r + T_w + T_s \\ &= \sum_{i=0}^{M} [tq_{u_i} + tr_{u_i}] + \sum_{i=0}^{M} [\frac{D_i}{v_i}] + \sum_{i=0}^{M} \frac{L_i}{b_i} \end{aligned} \quad (1)$$

Para calcular la latencia de enrutamiento tr de un nodo, se utiliza un modelo de latencia de un switch programable con soporte para P4, basado en el estudio de latencias presentado en [17]. Este trabajo explora la latencia en el procesamiento de paquetes en la arquitectura Tofino, específicamente en el modelo BF2556X-1T, a velocidades de enlace de 10 Gbps y 100 Gbps. Tras analizar la latencia base del switch, las latencias en las fases de *parse* y *deparse*, y las latencias en la fase de control para cada tipo de

operación *match*, y la latencia que conlleva el paso de los paquetes por un número de tablas *match-action* determinado, los autores obtienen un modelo de latencia del switch que resulta útil para la predicción de latencias en la evaluación del rendimiento de una red sin necesidad de utilizar el hardware real.

Para cada tipo de nodo (*source, transit o sink*), las cabeceras del paquete y el tamaño de los datos varían, por lo que la latencia tr difiere entre ellos. Para calcular la latencia, el factor más importante es el número de saltos, ya que determina cuántos nodos han insertado sus datos en el paquete sonda.

- Un nodo *source* genera un paquete de telemetría de 58 bytes y añade 40 bytes de telemetría del puerto de salida al enviarlo. Su latencia se basa en las siguientes operaciones de comparación: 2 tablas con *match*, o comparaciones exactas de valores de 8 y 16 bits, y otras 4 tablas de comparaciones *longest prefix match* (LPM), 2 de ellas de 18 bits y las 2 últimas de 32 bits.
- Un nodo *transit* procesa $58 + 40 \cdot m$ bytes por paquete, donde m es el número de saltos que ha dado el paquete previamente, y añade 40 bytes de datos a cada paquete de telemetría recibido, incluyendo la información de los puertos de entrada y salida por los que pasa el paquete. La latencia de un nodo *transit* se compone de 2 tablas de comparación exacta de 16 bits.
- Un nodo *sink* procesa $58 + 40 \cdot M$ bytes, donde M es el número total de saltos que ha dado el paquete sonda. Añade 40 bytes de telemetría del puerto de entrada del paquete sonda y reporta $40 \cdot M + 1$ bytes al controlador. La latencia de un nodo *sink* está compuesta por dos operaciones de comparación en tablas exactas: la primera, de 8 bits, y la segunda, de 16 bits.

Teniendo en cuenta el tamaño de las cabeceras y de los datos de un paquete en cada salto y las tablas *match-action* utilizadas para procesar las cabeceras, según el modelo de latencia de [17], y sumando el tiempo de contención tq de cada nodo, se puede calcular la latencia de enrutamiento de un paquete de telemetría a lo largo de una ruta como:

$$\begin{aligned} T_r &= tq + tr \\ &= \sum_{i=1}^{M} \frac{q_{u_i} occ + (58 + 40i)}{qr} \\ &+ lat_{SO}(58 + 40) \\ &+ \sum_{i=1}^{M-1} lat_{TR}(58 + 40(i+1)) \\ &+ lat_{SI}(58 + 40(M)) \end{aligned} \tag{2}$$

El tiempo de vuelo Tw es la suma de los tiempos de vuelo de cada uno de los enlaces de la ruta:

$$T_w = \sum_{i=0}^{M} \frac{dist(i)}{v} \tag{3}$$

La latencia de serialización del paquete de telemetría, en este caso, con un tamaño variable en cada salto de la ruta, se corresponde con:

$$T_s = \sum_{i=0}^{M} \frac{58 + 40i}{b_i} \tag{4}$$

Con esto, podemos calcular el tiempo que tarda un paquete de telemetría en recorrer una ruta determinada, formada por M enlaces.

B. Cálculo de caminos eulerianos

Puesto que partimos de la premisa de considerar todos los grafos de una red como grafos dirigidos, en los que cada enlace de la red corresponde a una arista bidireccional entre un par de nodos, nuestro estudio consiste en utilizar diferentes algoritmos para calcular rutas eulerianas de un grafo, y evaluar las diferencias entre las rutas obtenidas. Estos algoritmos consisten en:

- Ruta euleriana recorriendo los nodos vecinos en anchura.
- Ruta euleriana recorriendo los nodos vecinos en profundidad mediante el algoritmo de Fleury.
- Ruta euleriana recorriendo los nodos con el algoritmo de NetworkX[18], basado en el algoritmo de Hierholzer.
- Variación del algoritmo de Hierholzer con agrupación de ciclos en un árbol.

B.1 Algoritmo de recorrido en anchura

En primer lugar, utilizamos un algoritmo de recorrido en anchura de grafos, que se muestra en el Algoritmo 1. Se comienza por un nodo inicial y se obtienen todos sus vecinos. Se recorren todos los enlaces entre ellos, generando un ciclo o ruta en anchura, es decir, una lista de enlaces dirigidos sin repetición que recorren todos los vecinos del nodo elegido, como muestra el código del Algoritmo 2. En la Figura 1 se muestra un ejemplo del recorrido en anchura de un grafo. Si empezamos por el nodo A, el primer ciclo resultante al visitar a todos sus vecinos es el denotado con el número 1:

$$\begin{aligned} &(A,B),(B,A),(A,C),(C,A),(A,D),(D,A), \\ &(A,E),(E,A),(A,F),(F,A),(A,H),(H,A) \end{aligned}$$

El siguiente nodo de la cola sin haber sido explorado completamente es el nodo B, cuyo ciclo en anchura resultante es el denotado con el número 2 en la Figura 1:

$$(B,C),(C,B),(B,D),(D,B)$$

Tras calcular el ciclo, se inserta en la ruta completa r_e, buscando un enlace cuyo destino coincida con el origen del primer enlace del ciclo obtenido (Algoritmo 3). Este proceso se repite hasta recorrer todos los

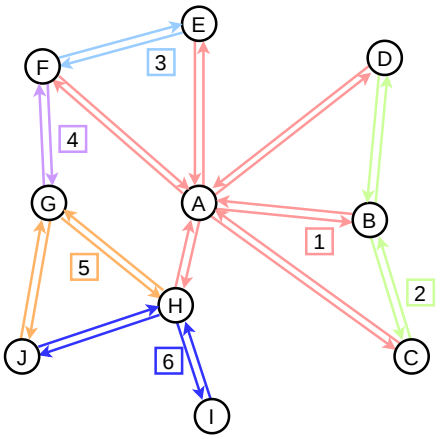


Fig. 1: Ciclos obtenidos con el algoritmo de exploración en anchura

Algoritmo 1 Camino euleriano en anchura

Entrada: Grafo dirigido G, nodo inicial n_0
Salida: ruta euleriana r_e

$r_e \leftarrow [\emptyset]$
$lNodos \leftarrow [n_0]$
for $i \in \{0, |G| - 1\}$ **do**
  $n \leftarrow lNodos[i]$
  if n no está completamente explorado **then**
    $V \leftarrow vecinos(G, n)$
    $lNodos \leftarrow lNodos \cup V$
    $V_r \leftarrow [\emptyset]$
    for $v \in V$ **do**
      if v no explorado **then**
        $V_r \leftarrow V_r + v$
      end if
    end for
    $r \leftarrow cicloAnchura(G, n, V_r)$
    $r_e \leftarrow insertarCiclo(r_e, r)$
  end if
end for
return r_e

Algoritmo 2 Función: *cicloAnchura*

Entrada: Grafo dirigido G, nodo actual n, vecinos de n $V_n \neq [\emptyset]$
Salida: Ciclo c

$c \leftarrow [\emptyset]$
for $v_i \in V_n$ **do**
  $c \leftarrow c + (n, v_i)$
  $c \leftarrow c + (v_i, n)$
end for
return c

enlaces de la red. El resultado de insertar el ciclo de los vecinos de B en la ruta de A es:

$$(A,B),(B,C),(C,B),(B,D),(D,B),(B,A),$$
$$(A,C),(C,A),(A,D),(D,A),(A,E),(E,A),$$
$$(A,F),(F,A),(A,H),(H,A)$$

Algoritmo 3 Función: *insertarCiclo*. Inserta un ciclo en una ruta euleriana

Entrada: Ruta euleriana R_e, nuevo ciclo C_n
Salida: Ruta Euleriana R_e

for $(u_i, v_i) \in R_e$ **do**
  if $v_i == u_0 \in C_n$ **then**
    $R_e \leftarrow R_e[0:i] + C_n + R_e[i:|R_e| - 1]$
  end if
  return R_e
end for

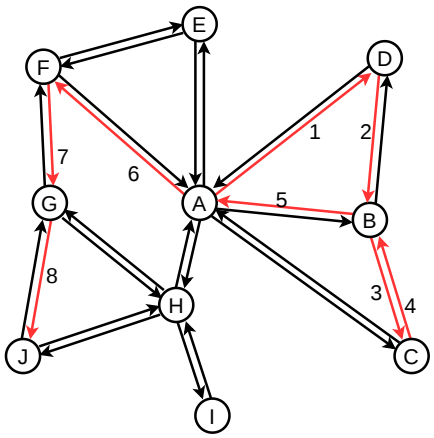


Fig. 2: Ruta de 8 saltos en profundidad mediante el algoritmo de Fleury

B.2 Algoritmo de Fleury (recorrido en profundidad)

En el caso del recorrido en profundidad del grafo, utilizamos el algoritmo de Fleury. En este algoritmo, que se muestra en el Algoritmo 4, empezamos la ruta en un nodo inicial y seleccionamos un nodo vecino al que nos desplazamos. Añadimos la arista que hemos usado a la ruta euleriana y la eliminamos del grafo. Se repite el proceso con cada nuevo salto, hasta eliminar todas las aristas del grafo. El requisito para ir de un nodo u a un nodo vecino v es que, al eliminar la arista (u, v) que los conecta, el grafo siga siendo conexo (línea 10, Algoritmo 4). Además, si el grado de salida de un nodo es 1, debemos coger ese salto de manera obligatoria. En la Figura 2 se muestran 8 saltos calculados mediante el algoritmo de Fleury.

B.3 Algoritmo de Hierholzer (implementación de NetworkX)

El algoritmo del camino euleriano de NetworkX es una implementación del algoritmo de Hierholzer [15]. Este algoritmo busca ciclos que empiecen y terminen en el mismo nodo. Cuando se encuentra un ciclo, se buscan nuevos ciclos a partir de los nodos cuyos enlaces no hayan sido recorridos en su totalidad, que forman parte de un ciclo ya encontrado. Una vez encontrados todos los ciclos del grafo, se insertan en una ruta euleriana, como se hace en el algoritmo de búsqueda en anchura. En la Figura 3 se muestra un ejemplo de los distintos ciclos encontrados en el grafo mediante el algoritmo de Hierholzer.

Algoritmo 4 Camino euleriano en profundidad, algoritmo de Fleury

Entrada: Grafo dirigido G , nodo inicial n_0
Salida: Ruta euleriana r_e

1: $r_e \leftarrow [\emptyset]$
2: $n \leftarrow n_0$
3: **while** $G \neq \emptyset$ **do**
4: **if** $out_{deg}(n) = 1$ **then**
5: $m \leftarrow vecinos(G, n), |m| = 1$
6: $r_e \leftarrow r_e + (n, m)$
7: $G \leftarrow G - (n, m)$
8: $n \leftarrow m$
9: **else if** $out_{deg}(n) > 1$ **then**
10: $(n, m_{sel}) \leftarrow seleccionarSalto(G, n)$
11: $r_e \leftarrow r_e + (n, m_{sel})$
12: $G \leftarrow G - (n, m_{sel})$
13: $n \leftarrow m_{sel}$
14: **end if**
15: **end while**
16: **return** r_e

B.4 Algoritmo modificado de Hierholzer

En nuestro caso, además de utilizar el algoritmo existente en la librería NetworkX descrito en la Sección III-B.3, implementamos una variación, que se muestra en el Algoritmo 5. Esta variación introduce los siguientes cambios:

1. La selección del siguiente nodo se realiza en base al nodo que tenga un enlace no explorado con mayor utilización (línea 24, Algoritmo 5). El ciclo resultante será "hijo"del ciclo del que se obtuvo el nodo.

2. Ordenamos los ciclos en una estructura de árbol, en el que cada nuevo ciclo tiene como nodo padre el ciclo del que proviene.

3. Si uno de los ciclos que se están calculando supera la latencia máxima establecida o el MTU, se añade al árbol y se continúa en un nodo nuevo del árbol. La Figura 4 muestra un ejemplo de organización en árbol. Suponiendo un MTU máximo alcanzado tras 5 saltos, podemos ver cómo el ciclo 5 se divide en dos nodos en el árbol: 5A y 5B.

Para generar una ruta euleriana completa, se concatenan los ciclos en una única ruta euleriana como hacemos en la exploración en anchura, utilizando el Algoritmo 3. Sin embargo, la organización en árbol nos permitirá agrupar ciclos por límite de latencia y de MTU, como veremos en el apartado siguiente.

C. División de rutas

Para enviar los paquetes sonda de telemetría, es importante no sobrepasar el MTU de la red al añadir nuevos datos al paquete en cada salto. Para ello, conociendo el MTU de los enlaces de la red y la cantidad de datos que se añaden al paquete de telemetría en cada nodo *transit*, se puede dividir la ruta euleriana en segmentos y enviar tantos paquetes como segmentos hay, pues cada segmento tendrá un nodo *source* inicial y un nodo *sink* final.

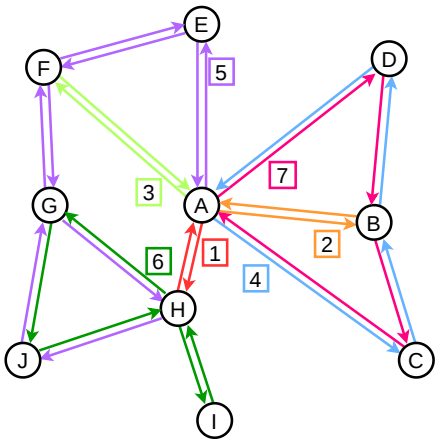


Fig. 3: Ciclos encontrados en el grafo mediante el algoritmo de Hierholzer

Algoritmo 5 Árboles de Hierholzer

Entrada: Grafo dirigido G , nodo inicial n_0
Salida: Árbol de Hierholzer $arbol_h$

1: **const** MTU_M,LAT_M
2: $arbol_h$: Grafo dirigido
3: $n \leftarrow n_0$
4: **while** $edges(G) \neq \emptyset$ **do**
5: $c \leftarrow [\emptyset]$ Ciclo
6: $c_r \leftarrow [\emptyset]$ Ciclo raíz
7: **if** $out_{deg}(n) > 0$ **then**
8: $m \leftarrow vecinoLatMax(G, n)$
9: $c \leftarrow c + (n, m)$
10: $n \leftarrow m$
11: $G \leftarrow G - (n, m)$
12: **if** $lat(r) \geq LAT_M$ || $mtu(r) \geq MTU_M$ **then**
13: $addNode(arbol_h, c)$
14: $addEdge(arbol_h, c, c_r)$
15: $r_{Raiz} \leftarrow r$
16: $r \leftarrow [\emptyset]$
17: **end if**
18: **else**
19: $addNode(arbol_h, c)$
20: $addEdge(arbol_h, c, c_r)$
21: $c \leftarrow [\emptyset]$
22: $n, c_n \leftarrow nodoLatMax(arbol_h)$
23: $c_r \leftarrow c_n$
24: **end if**
25: **end while**
26: **return** $arbol_h$

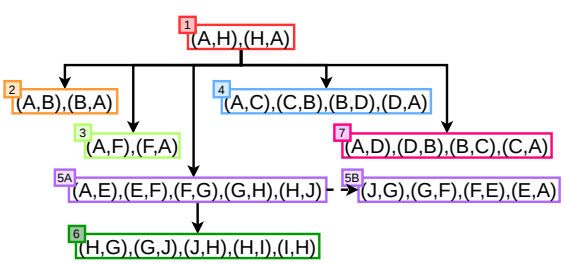


Fig. 4: Ciclos del algoritmo de Hierholzer organizados en árbol

Además, si tenemos un límite de latencia que no queremos sobrepasar para respetar los requisitos de AoI, podemos utilizar información de la red para calcular la latencia máxima que alcanzará el paquete de telemetría desde su emisión hasta su recogida y establecer segmentos de la ruta euleriana en función del MTU y de dicha latencia máxima.

La división en segmentos de las rutas eulerianas obtenidas con los algoritmos de búsqueda en anchura (Sección III-B.1), profundidad (Sección III-B.2) y el algoritmo de la librería de NetworkX (Sección III-B.3) consiste en evaluar, en un instante dado, el uso de cada enlace que forma parte de la ruta y segmentarla cuando se supere este límite o, a lo sumo, el MTU. El resultado será un conjunto de rutas segmentadas a través de las cuales se enviará un paquete de telemetría que no excederá el MTU ni la latencia máxima establecida. En nuestro caso, los datos de utilización de la red se simulan mediante el modelo de latencias del switch descrito en la Sección III-A.

Sin embargo, la división de segmentos de las rutas del árbol con ciclos de Hierholzer (Sección III-B.4) nos permite obtener segmentos de rutas eulerianas sin necesidad de recorrer la ruta completa y, además, permite formar segmentos con ciclos no consecutivos al dar prioridad a los ciclos por la latencia de sus enlaces. Para obtener las rutas de telemetría a partir del árbol de ciclos de Hierholzer, se siguen los siguientes criterios:

1. Identificar todos los ciclos que cumplen con el límite establecido. En la Figura 4, podemos observar que los ciclos 5A y 6 cumplen con el límite de 5 saltos. Estos ciclos se considerarán completos y se eliminarán del árbol. Si algún ciclo queda desconectado del árbol, como sucede con el 5B, se le asignará como nodo padre el ciclo raíz del nodo del que fue desconectado. En la Figura 5 vemos cómo ahora el ciclo 5B es hijo del ciclo 1.

2. Para cada nodo hoja del árbol, evaluar cuál es el mejor candidato para concatenar con su ciclo raíz. Para evaluar al mejor candidato, usamos el criterio de minimización de la latencia. En la Figura 5 se observa que los únicos candidatos posibles que no superan el criterio de 5 saltos al concatenar con su ciclo raíz son los ciclos 2 y 3. Suponiendo que el ciclo de menor latencia es el 2, el árbol resultante se muestra en la Figura 6. Se utiliza el criterio de menor latencia con el objetivo de obtener segmentos lo más grandes posible, dentro del límite de latencia máxima y de MTU establecidos, y así reducir al máximo el número de segmentos del árbol.

Iterando sobre estos 2 criterios hasta que no sea posible concatenar más ciclos, obtendremos las rutas que recorrerán los paquetes de telemetría. Como se muestra en la Figura 6, ninguno de los nodos hoja restantes puede cumplir con el límite de 5 saltos, por

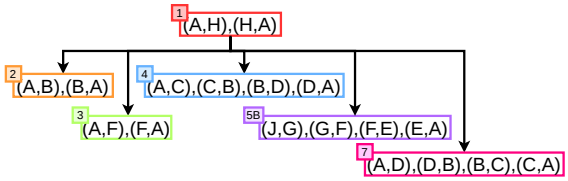


Fig. 5: Árbol de ciclos de Hierholzer tras eliminar los ciclos 5A y 6

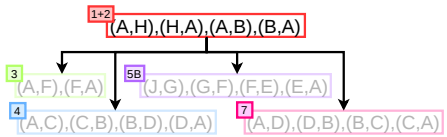


Fig. 6: Árbol de ciclos de Hierholzer tras concatenar los ciclos 1 y 2

lo que no es posible concatenar más rutas. Como no podemos reducir el número de rutas, el resultado es:

$$
\begin{aligned}
1+2 &: (A,H),(H,A),(A,B),(B,A)\\
3 &: (A,F),(F,A)\\
4 &: (A,C),(C,B),(B,D),(D,A)\\
5A &: (A,E),(E,F),(F,G),(G,H),(H,J)\\
5B &: (J,G),(G,F),(F,E),(E,A)\\
6 &: (H,G),(G,J),(J,H),(H,I),(I,H)\\
7 &: (A,D),(D,B),(B,C),(C,A)
\end{aligned}
$$

IV. Evaluación

Tras obtener los caminos eulerianos necesarios para recoger la información de la red, vamos a analizar los segmentos generados por cada algoritmo en torno a 3 ejes principales: la longitud de los segmentos tras dividir la ruta euleriana completa, la cantidad de segmentos en los que se ha dividido una ruta, y la localización de los segmentos en una red, según el estado de ocupación de los enlaces.

Para realizar la evaluación, utilizaremos grafos dirigidos conexos de grado par, tal como indicamos en nuestra propuesta, para modelar redes de comunicaciones, lo que asegura la existencia de caminos eulerianos. Todos los nodos del grafo tendrán como mínimo 4 vecinos, con excepción de aquellos que simulan nodos de acceso a la red, que tienen solo uno. El número de nodos del grafo será variable, aunque el número de aristas del grafo será de 200 (la mitad de entrada, y la mitad de salida) para simular una red con 100 enlaces.

Las aristas del grafo tendrán asignada una utilización u, expresada entre 0 y 1, que indica la ocupación del buffer de entrada del nodo destino de cada enlace. Según la documentación del *switch* [19], cuyo modelo de latencia se describe en la Sección III-A, el tamaño de buffer de paquetes compartido entre los puertos es de 22 MB. Considerando 32 puertos y sus correspondientes buffer de entrada y salida, el tamaño de buffer de entrada dedicado por puerto es de 800 KB. La ocupación de un enlace en cada salto de la ruta será $q_{occ} = u \cdot 800$ KB, $(0 < u < 1)$, a la que sumamos el tamaño del paquete de

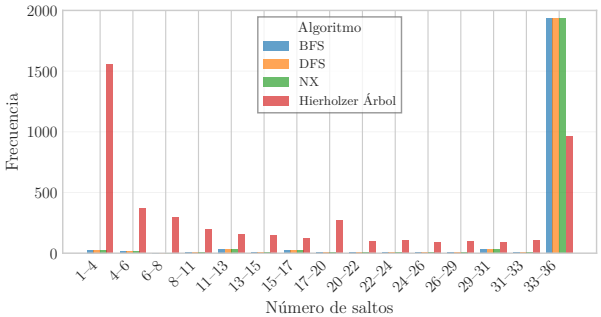


Fig. 7: # Saltos de cada segmento sin límite de latencia

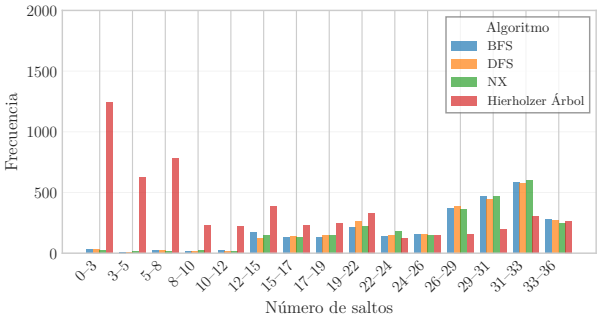


Fig. 8: # Saltos de cada segmento con límite del 30 % de latencia máx.

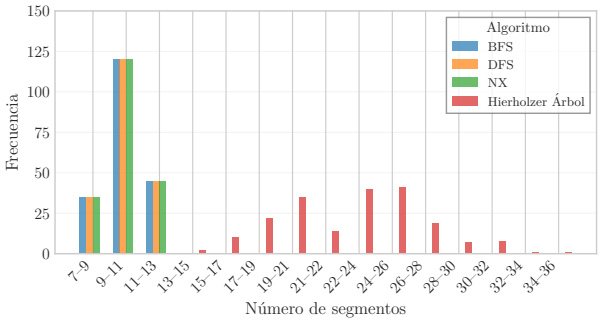


Fig. 9: # Segmentos sin límite de latencia

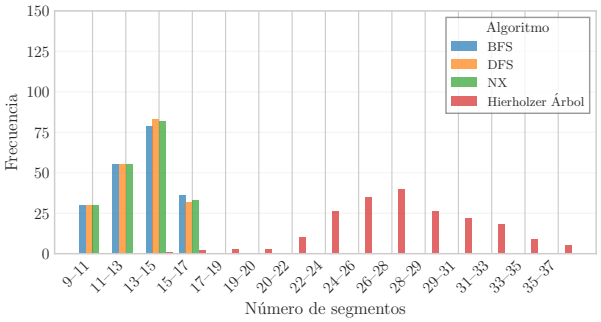


Fig. 10: # Segmentos con límite del 30 % de latencia máx.

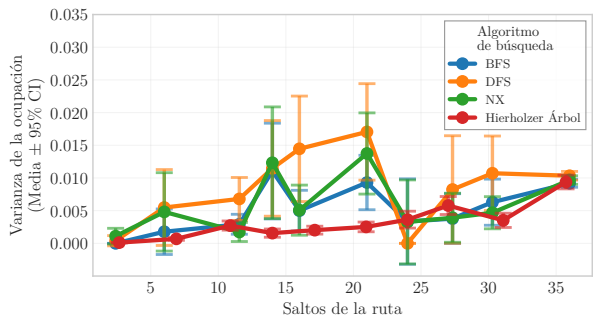


Fig. 11: Varianza de ocupación de enlaces de los segmentos sin límite de latencia

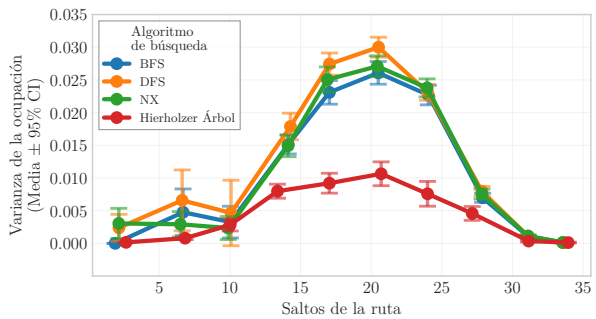


Fig. 12: Varianza de ocupación de enlaces de los segmentos con límite del 30 % de latencia máx.

telemetría L en cada salto de la ruta para obtener la latencia de origen a destino.

Para asignar la ocupación a cada enlace se generará una zona de congestión seleccionando un nodo aleatorio del grafo. A todas las aristas de este nodo se les asigna una utilización de 0,9. A las aristas de los vecinos de este nodo se les asigna una utilización de 0,7, excluyendo las aristas que conectan con el nodo inicial. Del mismo modo, se asigna una ocupación de 0.3 a las aristas de los nodos vecinos del tercer nivel, de modo que obtenemos una zona de congestión en la que la ocupación de los enlaces aumenta a medida que nos acercamos al nodo central.

Se utilizarán 40 grafos generados aleatoriamente, sobre los cuales se calcularán 5 rutas eulerianas por cada algoritmo, siempre comenzando en un nodo inicial aleatorio. Con ello, compararemos los resultados de 200 rutas diferentes por cada algoritmo.

Sobre este escenario se realizarán 2 experimentos. En el primero, se calcularán las rutas eulerianas de los grafos y se dividirán en segmentos sin límite de latencia. En este caso, el límite lo marca el MTU. Esto significa que en una ruta euleriana de 200 saltos calculada con los algoritmos de anchura (BFS), Fleury (DFS) o el incluido en NetworkX (NX), obtendremos 6 segmentos: los 5 primeros de 36 saltos (el número máximo de saltos que permite añadir telemetría al paquete sin sobrepasar un MTU de 1.500 bytes) y un último segmento de 20 saltos como se observa en la Figura 7. Sin embargo, al realizar la agrupación de ciclos del algoritmo modificado de Hierholzer, obtenemos una mayor variedad de longitudes de segmentos, debido a la forma en que este algoritmo recorre el árbol de ciclos para crear las rutas de telemetría.

Evidentemente la dispersión en el número de saltos en cada segmento en el caso del algoritmo modificado de Hierholzer hace que el número de rutas necesarias para recorrer todo el grafo sea también más heterogéneo y más alto que en el caso de los algoritmos BFS, DFS y NX que como se ve en la Figura 9.

En el segundo experimento se establece una limitación de latencia, es decir, del tiempo que puede

pasar un mensaje de telemetría en la red desde que se inyecta hasta que es enviado al colector de telemetrías (conocido también como AoI). La limitación de latencia se calcula en base a una latencia máxima, que en este caso será la latencia de un camino de 36 saltos con todos los enlaces al 30 % de utilización, resultando en un límite de 2,86 ms. A la hora de realizar la división en segmentos de las rutas eulerianas, o el agrupamiento de ciclos en el algoritmo modificado de Hierholzer, los segmentos no podrán superar este límite. En este caso, el comportamiento de los algoritmos BFS, DFS y NX es algo diferente al del experimento anterior: el número de saltos que tienen cada segmento está más distribuido y no concentrado en valores altos (Figura 8) debido a que ahora la suma de las latencias de cada salto de un segmento debe ser menor a la latencia máxima, y cada salto tiene una latencia determinada por su ocupación, por lo que el número de saltos no es fijo de antemano como en el caso del MTU. Sin embargo, en el caso del algoritmo modificado de Hierholzer los resultados no varían, ya que en el cálculo del árbol intrínsecamente se tiene en cuenta la latencia de los enlaces.

Por otra parte, en las Figuras 11 y 12 se muestra la varianza de la ocupación de los enlaces de cada segmento, calculada sin límite de latencia y con límite, respectivamente. De tal forma que, a mayor varianza, mayor diferencia en la ocupación de los enlaces que componen un segmento. Como se observa, el algoritmo modificado de Hierholzer es el que presenta una mayor homogeneidad en la carga de los enlaces de cada segmento, debido a que prioriza los enlaces de mayor ocupación. Mientras que el resto de métodos elige los enlaces a recorrer de forma pseudo-aleatoria. Esto implica que las rutas seleccionadas por el algoritmo modificado de Hierholzer se concentran en zonas de la red con un grado de ocupación similar.

V. Conclusiones

En este trabajo se ha abordado el desafío de la monitorización en tiempo real de redes congestionadas mediante estrategias de generación de rutas de telemetría INT-MD, y se ha realizado una comparativa entre técnicas convencionales, basadas en la búsqueda de caminos eulerianos, y una modificación del algoritmo de Hierholzer propuesta por los autores de este trabajo.

Para ello, se han comparado las características de las rutas obtenidas con estos algoritmos: el número y el tamaño de las rutas generadas, y la homogeneidad de la carga de los enlaces que componen cada ruta. Siempre teniendo en cuenta las restricciones de MTU o de AoI impuestas por las técnicas de telemetría INT-MD.

Los resultados experimentales obtenidos han mostrado que la modificación del algoritmo de Hierholzer propuesta en este trabajo obtiene rutas de enlaces con una ocupación similar y, por lo tanto, concentradas en zonas del mismo nivel de congestión. Esta característica es deseable a la hora de obtener un mecanismo adaptativo para ajustar la frecuencia de muestreo de una red.

En definitiva, este enfoque adaptativo permite mejorar la monitorización en entornos congestionados, aspecto especialmente crítico en escenarios de computación en el borde donde las aplicaciones tienen requisitos estrictos de latencia y fiabilidad. Como trabajo futuro, se plantea integrar y desplegar los algoritmos de generación de rutas descritos en un entorno real para validar los resultados obtenidos, así como evaluar su efecto en aplicaciones típicas.

Agradecimientos

Esta publicación es parte de la beca PRE2022-102090 asociada al proyecto PID2021-123627OB-C52, y parte del proyecto de I+D+i PID2024-158682OB-C32, financiados por MICIU/AEI/10.13039/501100011033 y FEDER/UE.

Referencias

[1] The P4 Language Consortium, "$P4_{16}$ Language Specification. version 1.2.5," Tech. Rep., The P4 Language Consortium, 2024-10-11, `https://p4.org/wp-content/uploads/sites/53/2024/10/P4-16-spec-v1.2.5.pdf`.

[2] Alibaba, CableLabs, Cisco Systems, Dell, Intel, Marwell, Netronone, VMware, "In-band Network Telemetry (INT) Dataplane Specification. Version 2.1," Tech. Rep., P4.org Applications Working Group, 2020-11-11, `https://github.com/p4lang/p4-applications/blob/master/docs/INT_latest.pdf`.

[3] Mei-ko Kwan, "Graphic programming using odd or even points," *Chinese Math*, vol. 1, pp. 273–277, 1962.

[4] George Pólya, Robert E. Tarjan, and Donald R. Woods, *Hamiltonian and Eulerian Paths*, Birkhäuser Boston, 2009.

[5] Tian Pan, Enge Song, Zizheng Bian, Xingchen Lin, Xiaoyu Peng, Jiao Zhang, Tao Huang, Bin Liu, and Yunjie Liu, "INT-path: Towards Optimal Path Planning for In-band Network-Wide Telemetry," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, Apr. 2019, pp. 487–495, ISSN: 2641-9874.

[6] Qianchen Yuan, Fuliang Li, Tian Pan, Yifan Xu, and Xingwei Wang, "INT-react: An O(E) Path Planner for Resilient Network-Wide Telemetry Over Megascale Networks," in *2022 IEEE 30th International Conference on Network Protocols (ICNP)*, 10 2022, pp. 1–11.

[7] Tian Pan, Xingchen Lin, Haoyu Song, Enge Song, Zizheng Bian, Hao Li, Jiao Zhang, Fuliang Li, Tao Huang, Chenhao Jia, and Bin Liu, "INT-probe: Lightweight In-band Network-Wide Telemetry with Stationary Probes," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, 2021, pp. 898–909.

[8] Qianchen Yuan, Fuliang Li, Tian Pan, Yuhua Lai, Yetao Gu, and Xingwei Wang, "INT-Segment: MTU-Adaptive Single-Path In-Band Network-Wide Telemetry," in *2022 IEEE 30th International Conference on Network Protocols (ICNP)*, 10 2022, pp. 1–11.

[9] Fuliang Li, Qianchen Yuan, Tian Pan, Xingwei Wang, and Jiannong Cao, "MTU-Adaptive In-Band Network-Wide Telemetry," *IEEE/ACM Transactions on Networking*, vol. 32, no. 3, pp. 2315–2330, jan 2024.

[10] Ariel G. Castro, Arthur F. Lorenzon, Fábio D. Rossi, Roberto I. T. da Costa Filho, Fernando M. V. Ramos, Christian E. Rothenberg, and Marcelo C. Luizelli, "Near-Optimal Probing Planning for In-Band Network Telemetry," *IEEE Communications Letters*, vol. 25, no. 5, pp. 1630–1634, May 2021.

[11] Xuetong Hu, Yuming Zhang, Mingyuan Liu, Xiaoting Ma, Wei Quan, Deyun Gao, and Xue Zhang, "Adaptive-INT: Fast Network Anomaly Perception Based on Multidimensional Attribute Measurement," in *2024 Asia Communications and Photonics Conference (ACP) and International Conference on Information Photonics and Optical Communications (IPOC)*, Nov. 2024, pp. 1048–1053.

[12] Peiran Zhong, Fanqin Zhou, Lei Feng, and Wenjing Li, "FINT: Freshness-Based In-Band Network-Wide Telemetry in Resource-Constrained Environments," *IEEE Transactions on Network and Service Management*, vol. 22, no. 2, pp. 1314–1329, 2025.

[13] Jinghan Xu, Xiaolong Xu, Juan Zhao, and Honghao Gao, "FDSR-INT: A Flexible On-Demand In-Band Telemetry Approach for Aerial Computing Networks," *IEEE Internet of Things Journal*, vol. 12, no. 13, pp. 23257–23274, 2025.

[14] Adrian Bondy and U.S.R. Murty, *Graph Theory*, Springer London, 2010.

[15] Herber Fleischner, "X.1 Algorithms for Eulerian Trails," *Eulerian Graphs and Related Topics: Part 1, Volume 2, Annals of Discrete Mathematics*, vol. 50, pp. X.1–3, 1991.

[16] William James Dally and Brian Patrick Towles, *Principles and practices of interconnection networks*, The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, Oxford, England, Dec. 2003.

[17] David Franco, Eder Ollora Zaballa, Mingyuan Zang, Asier Atutxa, Jorge Sasiain, Aleksander Pruski, Elisa Rojas, Marivi Higuero, and Eduardo Jacob, "A comprehensive latency profiling study of the Tofino P4 programmable ASIC-based hardware," *Computer Communications*, vol. 218, pp. 14–30, Mar. 2024.

[18] "NetworkX. Network Analysis in Python," `https://networkx.org/en/`.

[19] "Intel® Tofino 2.0 Tbps," `https://www.intel.com/content/www/us/en/products/sku/218656/intel-tofino-2-0-tbps-2-pipelines/specifications.html`.

Performance Evaluation of Dragonfly Connection Rules

Javier Navaridas[1] and Jose A. Pascual[1]

Resumen— **Dragonfly is becoming one of the networks of choice for datacenters and high-performance computer systems as it is designed with current technology trends and offers a sweet spot in terms of cost, simplicity, performance, fault-tolerance and power. In a Dragonfly topology, compute nodes are connected to switches forming groups in such a way that all switches of a group have a direct connection with each other. Then each of the groups have a connection to every other group, resulting in a clique of cliques. This provides a very high connectivity with a very low diameter which are desired characteristics of HPC and DC interconnection networks. However, the way in which these groups are connected among themselves, namely the *connection rule*, has not been studied in detail. In this paper, we compare all proposed connection rules under a wide variety of architectural and workload configurations focusing on path length and maximum through under independent sources and on execution time from application traffic. Simulation results reveal that the impact of the connection rule on network performance seems to be much smaller than other aspects of the architecture such as routing or switch allocation. With the exception of a few anomalous cases, differences between connection rules were typically below 5%.**

Palabras clave— **Dragonfly topology, Connection rules, Routing algorithms, Performance evaluation, Simulation and modeling.**

I. Introduction

DATACENTER (DC) and High performance computing (HPC) systems are highly valuable resources which are continuously growing in importance as they form the backbone of large parts of our day-to-day thanks to the advent of cloud computing, but are also pivotal both in business and science contexts. Most current scientific studies rely on modeling and analyzing different natural phenomena and/or technological processes, which often require a huge amount of computing power impossible to attain using regular off-the-shelf computers. For instance, physicists, chemists or pharmaceutical researchers simulate, for different purposes, interactions between huge amounts of molecules. Likewise, in the business context, corporations demand large amounts of computing power in order to use data mining software over large databases, with the objective of extracting knowledge from raw data, and to use that knowledge to their advantage. Obtaining patterns of consumer habits, boosting sales, optimizing costs and profits, estimating stocks or detecting fraudulent behavior are just a few interesting application domains.

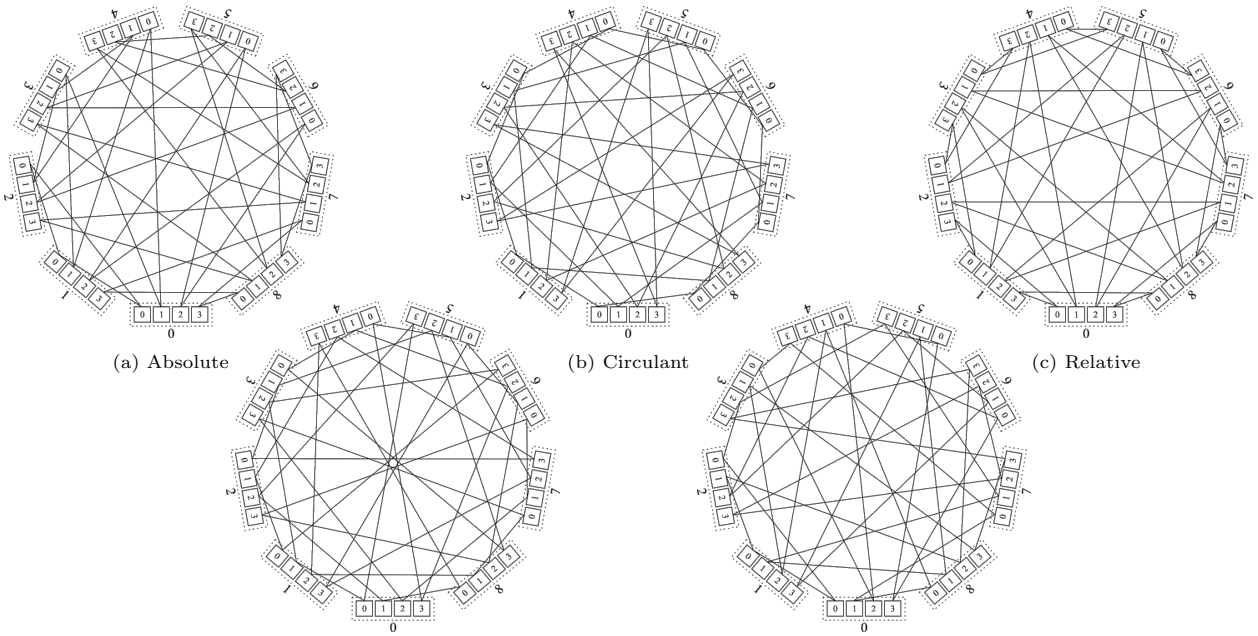
In these large scale computing systems, the interconnection network is a critical component as its performance, fault-tolerance and scalability will limit those of the full system. Most parallel applications will stall if traffic cannot flow between compute nodes rapidly enough and with sufficient data density. For this reason, research on various aspects of interconnection networks is essential to keep growing the availability of global computing resources.

The Dragonfly topology one of the most promising network architectures in modern DC and HPC systems because of its unbeatable cost-effectiveness [3]. Several systems nowadays are based on variants of dragonfly topologies, such as those which rely on Slingshot [4], or many that are built around Infiniband hardware [5, 6]. Dragonfly is a direct, hierarchical topology where endpoints are connected to switches which are organized in fully-interconnected groups. Then, these groups are connected in a fully interconnected manner, in such a way that each group has an inter-group link to every other group.

The Dragonfly topology is defined by three parameters: **p** is the number of ports in each switch which are connected to endpoints, **a** is the number of switches in each group and **h** is the number of ports in each switch that are connected to other groups However, these parameters only define the number of components, not how they are connected together. To completely defined the network, it is also necessary to define a connection rule which determines how the inter-group links are connected together. Indeed, the very first implementation of the Dragonfly topology [7] did never formally define any connection rule. Connection rules were first introduced by Camarero et al. which defined three such connection rules [1]. Later on, Belka et al. proposed another two variants [2]. Examples of these 5 variants are shown in Fig. 1. Research on Dragonfly systems does not always specify the connection rule but, typically, uses either the relative or absolute arrangements.

This paper evaluates these five connection rules against each other which, to our knowledge, have never been compared exhaustively before. Results from previous evaluations are inconclusive, with some of them reporting significant differences while others showing small performance differences [1, 2, 8, 9]. Furthermore, their comparisons were very limited investigating either theoretical bisection bandwidth or the performance of a few simple application models. In our study, we improve upon the literature by comparing the maximum throughput and the average distance under various traffic patterns achieved by large-scale networks composed of 16K nodes. In addition we also consider the execution time of more realistic trace-based workloads. Our evaluation in-

[1]Dept. of Computer Architecture and Technology, University of the Basque Country (EHU), corresponding e-mail: javier.navaridas@ehu.eus



(a) Absolute (b) Circulant (c) Relative

(d) Helix (e) Nautilus

Fig. 1: Various connection rules for a Dragonfly topology with parameters (p=2, a=4, h=2). (top) Original implementations proposed by Camarero et al. [1] (bottom) Later variants proposed by Belka et al. [2]. Boxes represent switches. Dotted rectangles delimit groups. Lines represents inter-group links. Intra-group links and endpoints are not shown for the sake of clarity as they are the same for all configurations.

cludes several router mechanisms, such as the routing algorithm and virtual channel allocator.

Across all our experiments, routing and switch allocation emerged as the main factors influencing network performance, while the choice of connection rule played a comparatively minor role. Under balanced, uniform traffic, performance differences among the Dragonfly variants were negligible. As traffic patterns became more unbalanced, small performance gaps began to appear: for instance, under Hot Region traffic, the nautilus variant achieved up to 29% higher throughput than others. However, nautilus also showed a pathological behavior under adversarial traffic because intra-group bottlenecks led to a sharp performance drop. These anomalies suggest that nautilus' performance may be skewed by input variability more than the other variants. Experiments using traces reinforced these observations: while some isolated cases favored one connection rule over the others, such instances were exceptions rather than the norm. In particular we found that absolute performs slightly worse with QVL routing (and to a lesser extent, with QVP) and that combining nautilus and QVP appears to be particularly detrimental for the stencil workload. All considered our conclusion is that these results indicate that the connection rule has a very limited and inconsistent impact on overall network performance.

II. Background

Dragonfly [7] is a hierarchical network architecture for large-scale systems which balances performance and efficiency. Furthermore, since Top-of-Rack switches are connected directly among themselves, it does not require extra racks to host the core network, as happens with tree-like [10–12] or Clos-based [13] topologies. One of the limitations of the Dragonfly topology is that it is full of cycles which could be problematic if not dealt with appropriately. However, Dragonfly comes with a specific HPC deadlock-avoidance mechanism [7] or can rely on general-purpose mechanisms [14].

When the topology was originally introduced, it did not define any specific rule to connect groups with each other, which is an important part of the network design, as we will see with our analysis later on. Camarero et al. [1] first introduced three simple connection rules: consecutive (more commonly known as absolute), palmtree (more commonly known as relative) and circulant, shown in Fig. 1a-c. In their analysis, however, they did not found any substantial differences between them. Later, Hastings et al. performed a bisection bandwidth analysis and found significant differences of up to 50% [8]. Kaplan et al. deepened the understanding of these connection rules by means of a simulation-based evaluation where the main focus was on task placement. They found insignificant performance differences in most cases, but some differences as high as 44% [9]. Later, Belka et al. [2] introduced the helix and nautilus connection rules in an attempt to improve bisection bandwidth. These two variants are shown in Fig. 1d-e. To our knowledge, there is no further analysis comparing all these connection rules together, so we aim at filling this gap by carrying out an independent analysis of all the variants. Existing literature considers either theoretical bisection bandwidth or a very limited range of application models. Our analysis goes beyond the state of the art by exploring a larger variety of traffic models.

The basic routing scheme for Dragonfly is known as dimensional (DIM), where a packet is routed within a group to the switch connected to the des-

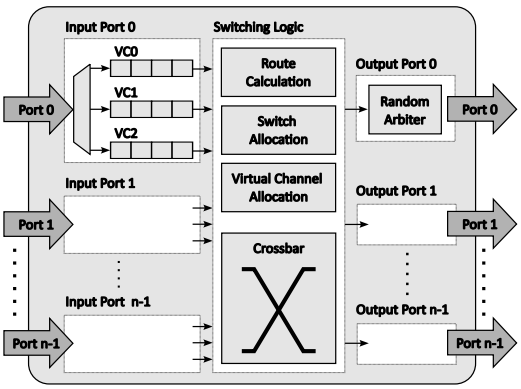


Fig. 2: Architecture of the router and its main logic blocks. For simplicity, only the internals of Port 0 are shown in detail.

tination group. Once it arrives to the destination group, it is routed within that group to the destination switch. However, DIM is known to be non-minimal [15] and, even worse, to not being capable to deal effectively with unbalanced workloads as inter-group links tend to become a bottleneck. For this reason, it is very common to rely on *proxy routing* to ensure a more balanced use of network resources. Instead of routing directly between source and destination, Proxy routing algorithms direct traffic through an intermediate group, router or server, called a Proxy. This technique is used for both load-balancing and fault tolerance purposes. The most common of these proxy routing algorithms is *Valiant routing* [16] (VAL) which selects a random proxy group to route packets through, effectively spreading traffic across the entire network to avoid generating bottlenecks in the inter-group links. Some authors argue that VAL uses exceedingly long routes and, thus, many approaches to shorten path lengths have been introduced [17–21]. The main rationale for favoring shorter paths is that each additional hop adds an extra transmission step between routers, increasing the overall latency, the utilization of network bandwidth and the energy consumed to send a message.

III. Experimental Set-up

In this section we present the experimental environment used to analyze the performance of the connection rules. We describe in detail the architecture of the system, the different traffic models employed and the performance metrics.

A. Simulation environment

Our performance evaluation was carried out using INSEE[1], an open-source, well-tested network simulator [22]. It uses a highly detailed model of the router at a phit-level and supports a wide variety of traffic models, including synthetic patterns, traces extracted from real applications [23] or artificially-generated traces [24]. In addition, it also implements many topologies, routing algorithms and router architectures.

[1]Available at https://gitlab.com/ExaNeSt/insee

Tabla I: Network configuration used in our evaluation.

Parameter	Value
Number of VCs	5
VC allocation	dally [7], DAQ [25], DAVC [14]
Routing	DIM and VAL [7], QVP and QVL [21]
Port arbitration	random
Transit queue length	4 packets
Packet length	8 phits
Phit length	64 bits
Number of injectors	1
Injection queue length	4 packets

Tabla II: Dragonfly configurations considered in our study.

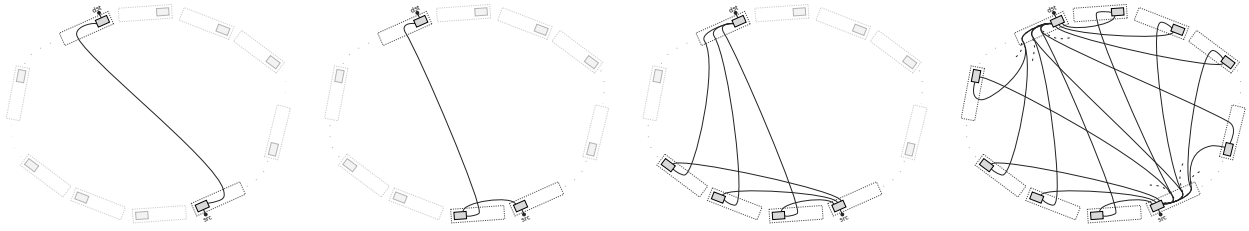
Experiment	Topology	endpoints	groups	switches	radix	links
Patterns	$D(8,16,8)$	16,512	129	2064	31	40,248
Traces	$D(4,8,4)$	1,056	33	264	15	2,508

In this work, we consider a simple router architecture with queues located in the input ports as illustrated in Fig. 2. The specific configuration that we use in our experiments is presented in Table I. The router employs virtual cut-through with adaptive virtual channel management. We consider three virtual channel allocation mechanisms: Dally [7], DAVC [14] and DAQ [25]. Packets are initially injected into a randomly selected adaptive VC and try to advance using it. If that VC is blocked, they try to switch to another adaptive VC at random. If this is also unavailable, they attempt to move to the escape subnetwork. To ensure this move does not result in a deadlock, packets moving into the escape subnetwork need to check that there is space for, at least, two packets following the bubble flow control mechanism [26]. Since we are considering lossless INs, if none of these movements is possible, the packet will stall until enough buffering is available in any of the buffers downstream. We consider 4 different routing schemes (see Fig. 3): DIM and VAL (from [7]) as well as QVP and QVL (from [21]). These range in various important aspects such as path length and path diversity to ensure a wide range of configurations is explored.

B. Topologies and Workloads

We consider the 5 connection rules mentioned above: absolute, relative and circulant (from [1]) as well as helix and nautilus (from [2]), see Fig. 1. We consider a different network size for each set of experiments, summarized in Table II together with their characteristics. Note we follow the best-practice criteria suggested by [1] where $h = p, a = 2 \cdot p$.

In the first set of experiments, we focus on raw performance by saturating the network under various synthetic traffic patterns from *independent sources*. These experiments stress the network infrastructure by injecting traffic at maximum load and, then, measure the network throughput and the average path length once a steady state is reached. Network throughput is a critical performance metric that needs to be as high as possible to ensure the network is able to deal with as much traffic as required by applications without reaching saturation. Path length is a critical metric which is closely related to transmission latency and resource utilization.



(a) Dimensional (DIM) (b) Quick Valiant Private (QVP) (c) Quick Valiant Local (QVL) (d) Valiant (VAL)

Fig. 3: Examples of paths generated by the routing strategies. *src* and *dst* represent the source and destination servers, respectively. Grey boxes represent routers and are put together in groups enclosed by the dotted areas. Solid lines represent paths generated by each algorithm. Dotted lines represent paths that are not shown for the sake of clarity.

Our experiments evaluate raw performance when handling the following synthetic traffic patterns:

Uniform: This is the traditional pattern in which packet destinations are selected uniformly at random. This generates a balanced distribution of traffic and serves the purpose of understanding the performance under balanced traffic conditions, which are the ideal for Dragonfly.

Hot Region: Another traditional pattern where destinations are selected randomly, but a given group of servers, the Hot Region, has a higher probability of being selected as destination, increasing the risk of generating localized congestion. This generates a non-uniform distribution of traffic and focuses on evaluating the performance under unbalanced traffic distributions. In our implementation, a quarter of the traffic goes to the Hot Region which spans one eighth of the network. The rest of the traffic is distributed uniformly through the whole network.

Adversarial: This is a pathological traffic pattern specifically designed for stressing Dragonfly fabrics. In Adversarial, the traffic is highly unbalanced since all the servers in a group send traffic randomly to the servers in the next group, which stresses the inter-group links between them.

While this first set of experiments helps to assess the raw performance of the connection rules, real applications behave very differently from synthetic traffic. Thus, our second set of experiments employs more realistic traffic models which follow the dynamics defined by a range of application-inspired workloads. These workloads synthesize *artificially-generated traces* which mimic representative applications and benchmarks from the high-performance computing (HPC) and datacentre (DC) domains. One of the defining characteristics of these traces is that they include causality among servers through data dependencies. The performance metric in this set of experiments is the execution time of the workloads. This is the total time required to deliver all the messages of the traces following all causal relationships.

Artificially-generated workloads are commonly used in the literature [27–30] for several reasons: (i) synthesizing artificial traces simplifies the curating process, since obtaining, handling and processing traces from real applications run in large-scale systems is complex and time-consuming. (ii) greater flexibility as the workloads can be generated for a specific target number of servers. (iii) traces that are captured in a real system may have been affected by the characteristics of the architecture where they have been captured, which may skew the results in the target system as discussed in [23]. Concretely, our evaluation considers the following application-inspired workloads:

All2All: This is a typical collective operation in HPC applications and also representative of DC traffic, as it constitutes the core of MapReduce. Servers communicate among themselves, sending messages to all other servers.

AllReduce: An optimized, binary implementation of the AllReduce collective [31], widely used in parallel applications from various domains.

RandomApp: Creates a random application by randomly adding messages between servers, which maintain causality among messages. According to [32], the traffic mix run on a typical DC is unstructured and essentially random in nature.

Sweep: This pattern resembles the sweep operation, which is very common in HPC applications with large matrices [33]. In our implementation, we consider a 3D sweep where the server in the corner of a virtual cubic mesh initiates a diagonal wavefront sending messages to their neighbors in X+, Y+ and Z+. Once a server receives the messages it is waiting for, it sends messages to its respective neighbors. The process repeats until all messages are delivered and the server in the opposite corner receives messages from all its neighbors.

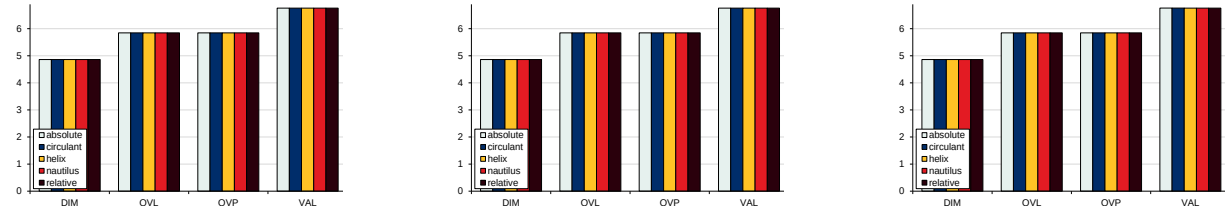
Stencil: This is another pattern resembling a typical communication pattern in HPC applications with large matrices. The application servers are arranged in a 2D lattice processing a certain part of the matrix and only need communicating with its neighbors to the North, South, East, and West.

IV. Performance evaluation

In this section we present and discuss the results of our experimental work. For simplicity we have organized our discussion based on the traffic model employed.

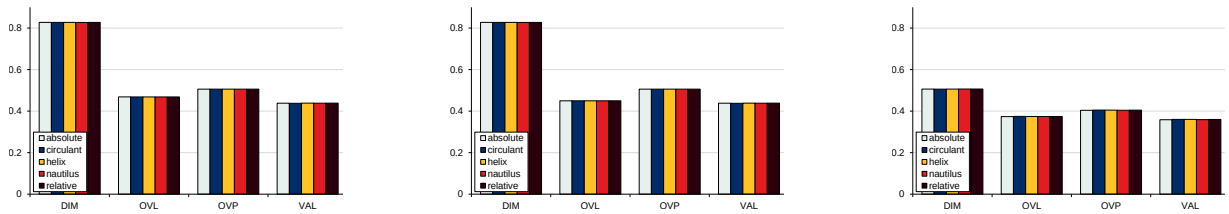
A. Uniform traffic

We start our discussion by analyzing the results with Uniform traffic, shown in Figs. 4 and 5. Uniform traffic is well balanced and should not generate any long-lasting bottleneck. The first thing that we



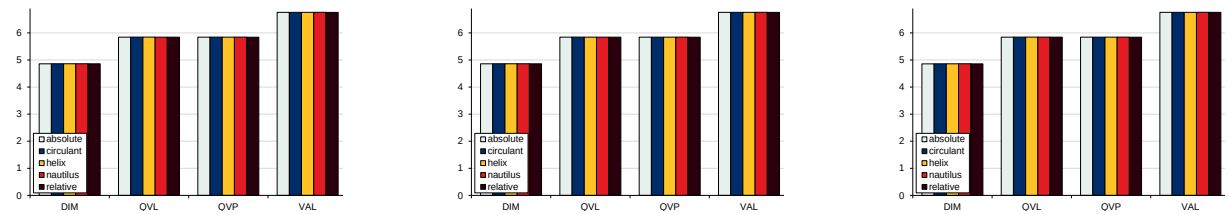
(a) DAQ (b) Dally (c) DAVC

Fig. 4: Average path distance – Uniform traffic.



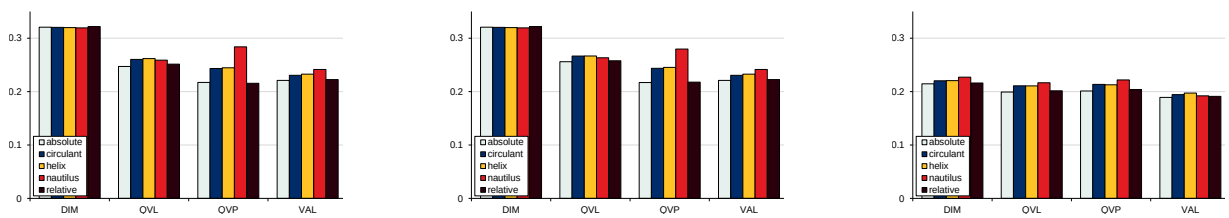
(a) DAQ (b) Dally (c) DAVC

Fig. 5: Maximum throughput – Uniform traffic.



(a) DAQ (b) Dally (c) DAVC

Fig. 6: Average path distance – Hot Region traffic.



(a) DAQ (b) Dally (c) DAVC

Fig. 7: Maximum throughput – Hot Region traffic.

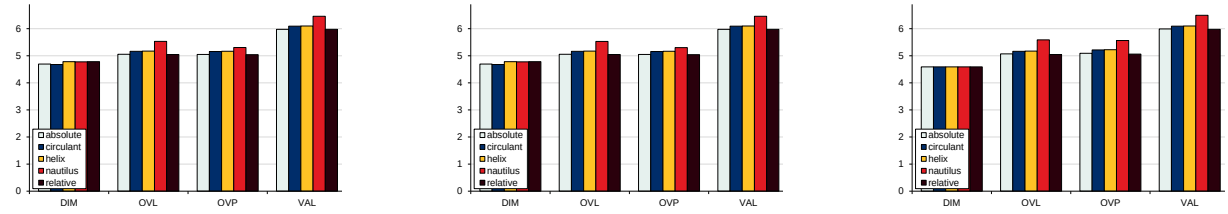
can see is that neither the connection rule nor the channel allocation mechanism have a visible effect on path length. Only the routing algorithm seems to have an impact on this metric. As expected DIM generates the shortest paths, whereas VAL generates the longest. Concretely, there is roughly one extra hop from DIM (4.86) to QVP (5.84) and QVL (5.84) plus another extra hop to VAL (6.76). The results between connection rules are identical up to the fourth significant digit, well below the random noise of the simulation.

With regards the throughput of the networks, we see that results with DAQ and Dally are very similar to each other whereas DAVC saturates at significantly lower loads. This is reasonable since DAVC is a general purpose channel allocation mechanism not specifically designed for Dragonfly. We can see, however, that the trends are similar to these with the other two mechanisms. The highest impact on the results is by the selected routing algorithm, with DIM being able to sustain roughly twice as much load as the other routing algorithms in the best cases. Looking into the effects of the connection rules on the achieved throughput, we observe that they are the same up to the third significant digit. Again, these deviations are not significative enough to make a difference.

B. Hot region traffic

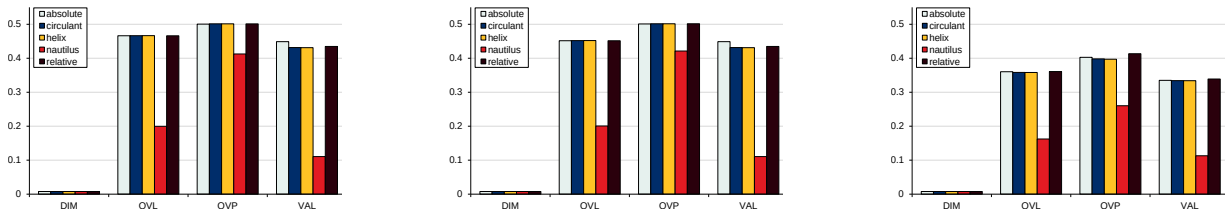
We move now to analyze the results with Hot region traffic, shown in Figs. 6 and 7. This traffic forces some unbalanced use of resources as the 16 groups affected by the hot region need to handle higher network pressure so they become a bottleneck. The distance results are very similar to these with Uniform traffic. Again, we find that the impact of the connection rule and the channel allocation mechanism have negligible effect on path length, with the routing algorithm having a definite impact. The results are slightly more noisy now, with connection rules obtaining similar path lengths up to the third significant digit, which means that differences are irrelevant.

Regarding throughput, we notice that with Hot region, it is severely reduced when compared with Uniform. This is reasonable, since most of the traffic is directed to a small portion of the network. This generates congestion around the hot region which



(a) DAQ (b) Dally (c) DAVC

Fig. 8: Average path distance – Adversarial traffic.



(a) DAQ (b) Dally (c) DAVC

Fig. 9: Maximum throughput – Adversarial traffic.

ultimately spreads through the whole network, resulting in reduced traffic flow. Now, the effect of the routing algorithms is noticeably reduced when compared with Uniform. DIM is still the one sustaining the highest throughput, but the advantage with the other algorithms is greatly reduced. DAQ and Dally, as before, maintain very similar saturation points and trends. Further, DAVC is outperformed by the others but to a lesser extent. We start to notice some obvious differences between the connection rules now. With DIM, we observe small differences of up to 1%. With QVL, differences are slightly larger, up to 4%. With QVP, we found the largest differences. In this case, with DAQ and Dally, nautilus can sustain up to 29% higher load than the other connection rules. With DAVC the differences are diluted, but nautilus still peaks over the other variants. Finally, with VAL, the differences are within a 5%, except for nautilus that offers 9% higher load than the worst configurations.

C. Adversarial traffic

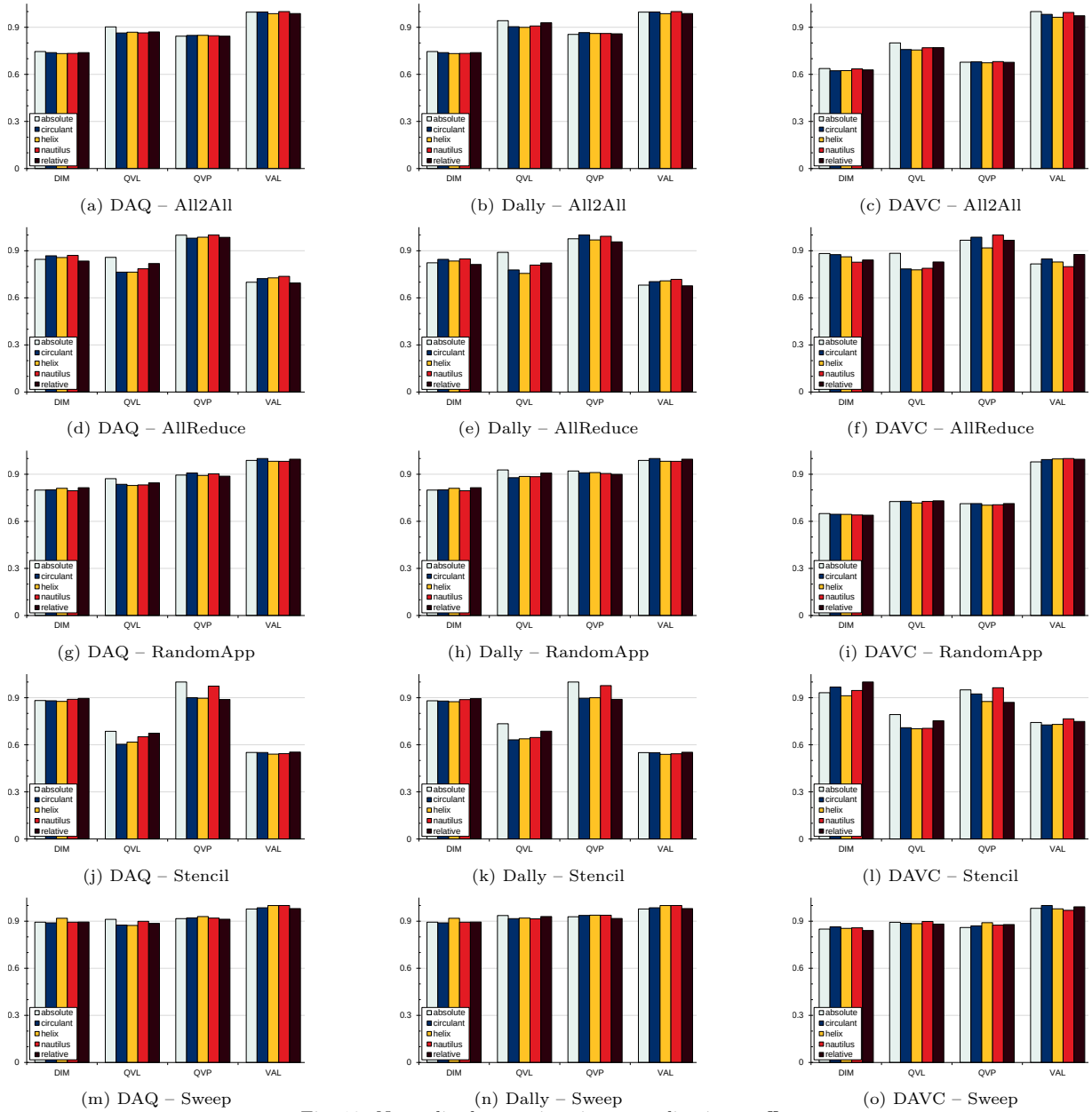
Let's discuss now the results with Adversarial traffic, depicted in Figs. 8 and 9. This traffic forces a highly unbalanced use of resources because all the endpoints in each group are randomly sending traffic to endpoints in the next group. This means that, for dimensional routing, the *unique* link between each group and the next becomes the bottleneck. For this reason, we can see that the load that can be accepted when using Dimensional routing is minuscule, practically tending to zero as the 128 endpoints in a group are saturating a single link. Therefore, we will focus on the other routing algorithms in our discussion below.

In this case, we can appreciate substantial differences between connection rules. In terms of distance, it is evident that path lengths are substantially shorter than in the case of uniform traffic. Moreover, this is the only traffic pattern where the results from the different variants are different. This is because the static shape of the pattern imposes different path distributions in each connection rule. It is clear that nautilus is the most affected as it features the longest paths with up to a 10% difference with the others. This relatively small difference in path length translates into massive differences in terms of throughput, which in the worst case can be as little as one quarter of the other variants. Upon close inspection, we found that Adversarial is specially pathological for nautilus as it generates localized bottlenecks in a few of the intra-group links. These bottlenecks throttle traffic flow of a significant part of the network. Interestingly, the higher the path diversity the more the effects of these bottlenecks spread and, thus, QVP (1 path) is harmed the least by this effect whereas VAL (129 paths) is the most affected. The differences among the other variants are less significant; well within a 1% for all cases except for absolute with VAL routing, which achieves a 4% higher throughput than the other variants.

D. Traffic from traces

We finally move to analyze the results obtained with the synthetic traces, collected in Fig. 10. Traffic from traces is more diverse and more dynamic and provides a wider, more realistic test for the networks. We can see that, in most cases, the differences among connection rules are relatively small (typically below 3%) and, indeed, less substantial than the differences from routing (up to 45%) or switch allocation (up to 17%). Detailed case-by-case discussion is outside of the scope of this paper, but we can highlight a few tendencies.

DIM and VAL routing appear to be the least affected by the connection rule, whereas QVL and QVP have larger differences among connection rules. The absolute connection rule is counterproductive when combined with QVL routing in most workloads, independently of the switch allocation. A similar effect is also noticeable for QVP with Stencil and, to a lesser extent, with RandomApp. With Stencil, the nautilus variant also seems to be hurted by



(a) DAQ – All2All (b) Dally – All2All (c) DAVC – All2All

(d) DAQ – AllReduce (e) Dally – AllReduce (f) DAVC – AllReduce

(g) DAQ – RandomApp (h) Dally – RandomApp (i) DAVC – RandomApp

(j) DAQ – Stencil (k) Dally – Stencil (l) DAVC – Stencil

(m) DAQ – Sweep (n) Dally – Sweep (o) DAVC – Sweep

Fig. 10: Normalized execution time – application traffic.

employing QVP, regardless of the switch allocation mechanism.

In general, we can observe that there seems to be no clear winner or loser among the connection rules. Differences are generally small and different workload+routing+switching mechanism combos are slightly benefited by different connection rules. We conclude, then, that the selection of a connection rule is not of special importance when designing Dragonfly networks.

V. Conclusions and future work

This paper has compared, for the first time, the five connection rules that have been proposed for the Dragonfly topology. Our study has considered networks of different sizes, with up to tens of thousands of nodes. In addition, we co-investigate the effect of the routing algorithms as well as the switch allocation mechanisms. Our results show that, overall, the impact of the connection rules on network performance are relatively small, except for a few anomalous cases. Routing and switch allocation, on the other hand, have a more significant influence on performance.

With balanced, uniform traffic, we found negligible differences among the Dragonfly variants. As we moved to increasingly more unbalanced traffic models, the differences between connection rules became slightly more noticeable. With Hot Region traffic, nautilus was able to achieve higher throughput in a few cases, with two specific cases where the differences where substantial, up to about 29% higher than the others. With adversarial, however, nautilus suffered a pathological case of intra-group bottlenecks which made its performance to plummet. These anomalies suggest that nautilus could be the most performance-sensitive of the connection rules. The experiments with traces evaluate the connection rules within more dynamic conditions and also found that, in general, differences between variants

were small and no clear champion could be discerned. We identified a few cases where the connection rule could affect performance. In particular we found that the absolute connection rule could be harmed when QVL, and to some extend QVP, are used. Furthermore, we found that the nautilus plus QVP combo was inferior for the stencil workload.

Acknowledgment

This work is supported by the Basque Government grants IT1504-22, KK-2024/00030, KK-2024/00068 and KK-2025/00098. It is also supported by the grants PID2023-152390NB-I00 and CNS2023-144315 funded by MICIU/AEI/10.13039/9/501100011033, "European Union NextGenerationEU/PRTR" and by "FEDER funds". Dr. Javier Navaridas holds a Ramón y Cajal fellowship funded by the Spanish Ministry of Science, Innovation and Universities (RYC2018-024829-I), funded by MCIN/ AEI/10.13039/501100011033 and, as appropriate, by "ESF Investing in your future" or by "European Union NextGenerationEU/PRTR".

Authors want to thank David P. Bunde for his support with ensuring that our implementations of Helix and Nautilus were consistent with their design.

Referencias

[1] C. Camarero, E. Vallejo, and R. Beivide, "Topological characterization of hamming and dragonfly networks and its implications on routing," *ACM Trans. on Arch. and Code Optimization*, vol. 11, no. 4, 2014.

[2] M. Belka *et al.*, "New link arrangements for dragonfly networks," *High-Perf. Interconn. Networks in the Exascale and Big-Data Era (HiPINEB)*, pp. 17–24, 2017.

[3] A. Khan *et al.*, "An evaluation of the effect of network cost optimization for leadership class supercomputers," in *Procs. of the Intl. Conf. for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '24. IEEE Press, 2024.

[4] D. De Sensi *et al.*, "An in-depth analysis of the slingshot interconnect," in *Intl. Conf. for High Performance Computing, Networking, Storage & Analysis*, 2020, pp. 1–14.

[5] G. Maglione-Mathey *et al.*, "Reducing the impact of interjob interference in dragonfly networks using virtual partitions," *IEEE Micro*, vol. 42, no. 3, pp. 50–56, 2022.

[6] N. McGlohon, R. Ross, and C. Carothers, "Evaluation of link failure resilience in multirail dragonfly-class networks through simulation," in *ACM SIGSIM Conf. on Principles of Advanced Discrete Simul.*, 2020.

[7] J. Kim *et al.*, "Technology-driven, highly-scalable dragonfly topology," in *35th Intl. Symposium on Computer Architecture*, 2008, pp. 77–88.

[8] E. Hastings *et al.*, "Comparing global link arrangements for dragonfly networks," *IEEE Intl. Conf. on Cluster Computing*, pp. 361–370, 2015.

[9] F. Kaplan, O. Tuncer, V. J. Leung, S. K. Hemmert, and A. K. Coskun, "Unveiling the interplay between global link arrangements and network management algorithms on dragonfly networks," in *2017 17th IEEE/ACM Intl. Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017, pp. 325–334.

[10] F. Petrini and M. Vanneschi, "k-ary n-trees: high performance networks for massively parallel architectures," in *11th Intl. Parallel Processing Symposium*, 1997.

[11] J. Navaridas, J. Miguel-Alonso, F. J. Ridruejo, and W. Denzel, "Reducing complexity in tree-like computer

[13] C. Camarero, C. Martínez, and R. Beivide, "Random interconnection networks," *Parallel Computing*, vol. 36, no. 2, pp. 71–85, 2010.

[12] A. Greenberg *et al.*, "Vl2: A scalable and flexible data center network," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, p. 51–62, 2009.
folded clos topologies for datacenter networks," in *High Performance Computer Architecture*, 2017, pp. 193–204.

[14] J. Pascual and J. Navaridas, "High-performance, low-complexity deadlock avoidance for arbitrary topologies/routings," in *Intl. Conf. on Supercomputing*, 2018.

[15] R. Curtsinger and D. Bunde, "Shortest paths in dragonfly systems," in *High-Perf. Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, 2019, pp. 1–8.

[16] L. Valiant, "A scheme for fast parallel communication," *SIAM Journal on Computing*, vol. 11, no. 2, 1982.

[17] D. De Sensi, S. Di Girolamo, and T. Hoefler, "Mitigating network noise on dragonfly networks through application-aware routing," in *High Performance Computing, Networking, Storage & Analysis*, 2019.

[18] M. Besta *et al.*, "Fatpaths: Routing in supercomputers and data centers when shortest paths fall short," in *SC20: Intl. Conf. for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–18.

[19] G. Kwauk *et al.*, "Boomgate: Deadlock avoidance in non-minimal routing for high-radix networks," in *High-Performance Computer Architecture*, 2021, pp. 696–708.

[20] M. Benito, E. Vallejo, and R. Beivide, "Lia: Latency-improved adaptive routing for dragonfly networks," *ACM Trans. Archit. Code Optim.*, vol. 22, no. 1, Mar. 2025.

[21] J. Navaridas and J. Pascual, "Improving the performance of dragonfly networks through restrictive proxy routing strategies," *Computer Networks*, 2025.

[22] J. Navaridas *et al.*, "Simulating and evaluating interconnection networks with insee," *Simulation Modelling Practice and Theory*, vol. 19, no. 1, pp. 494–515, 2011.

[23] J. Miguel-Alonso, J. Navaridas, and F. Ridruejo, "Interconnection network simulation using traces of mpi applications," *Intl. Journal of Parallel Programming*, vol. 37, pp. 153–174, 2009.

[24] J. Navaridas, J. Miguel-Alonso, and F. Ridruejo, "On synthesizing workloads emulating mpi applications," *2008 IEEE Intl. Symposium on Parallel and Distributed Processing*, pp. 1–8, 2008.

[25] J. Navaridas and J. Pascual, "Deadlock-free proxy routing with 2 virtual channels in dragonfly networks," *Journal of Parallel and Distributed Computing*, Under Review.

[26] M. García *et al.*, "On-the-fly adaptive routing in high-radix hierarchical networks," in *Intl. Conf. on Parallel Processing*, 2012, pp. 279–288.

[27] G. Zheng, G. Kakulapati, and L. Kale, "Bigsim: a parallel simulator for performance prediction of extremely large parallel machines," in *18th Intl. Parallel and Distributed Processing Symposium*, 2004, pp. 78–.

[28] R. Barrett *et al.*, "Assessing the role of mini-applications in predicting key performance characteristics of scientific and engineering applications," *Journal of Parallel and Distributed Computing*, vol. 75, pp. 107–122, 2015.

[29] M. Mubarak *et al.*, "Enabling parallel simulation of large-scale hpc network systems," *IEEE Trans. on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 87–100, 2017.

[30] F. J. Andújar *et al.*, "Extending the VEF traces framework to model data center network workloads," *J. Supercomput.*, vol. 79, no. 1, pp. 814–831, 2023.

[31] R. Thakur and W. Gropp, "Improving the performance of collective operations in MPICH," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, 2003, pp. 257–267.

[32] S. Kandula *et al.*, "The nature of data center traffic: measurements & analysis," in *ACM SIGCOMM Conf. on Internet Measurement*, 2009, p. 202–208.

[33] A. Hoisie, O. Lubeck, and H. Wasserman, "Performance analysis of wavefront algorithms on very-large scale distributed systems," in *Workshop on wide area networks and high perf. computing.* Springer, 2007.

Evaluación de arquitecturas de comunicación distribuidas para la codificación de imágenes acelerada por GPU

Haojie Zheng[1,2], Carlos Reaño[1] y Juan F. Ariño-Sales[2]

Resumen— **La inteligencia artificial (IA) ha transformado la manera en que interactuamos con los datos visuales, especialmente en el contexto de las empresas. Los sistemas de codificación multimodal permiten crear conexiones semánticas entre texto y datos visuales mediante modelos de IA. Esto abre nuevos mercados para las empresas al posibilitar motores de búsqueda visual, sistemas de recomendación y etiquetado automático de datos visuales. Sin embargo, la implementación de estos sistemas presenta desafíos técnicos significativos. El flujo de trabajo típico consiste en codificar las imágenes utilizando un modelo de IA, convertir estas representaciones en vectores semánticos e insertarlos en bases de datos optimizadas para búsquedas rápidas. Esto no solo afecta la eficiencia técnica, sino también la capacidad de las empresas para escalar estos sistemas a nivel comercial. Este artículo presenta un análisis comparativo exhaustivo de arquitecturas de comunicación para sistemas de codificación de imágenes a gran escala, evaluando los enfoques basados en gRPC y RabbitMQ, en términos de rendimiento y eficiencia en el uso de recursos. A través de experimentos controlados que procesan hasta 18.000 imágenes utilizando el modelo SigLIP, establecemos relaciones claras entre el rendimiento y la arquitectura que orientan las decisiones de diseño de sistemas para aplicaciones de búsqueda basada en contenido visual.**

Palabras clave— **Bases de datos, computación en la nube, visión por computador.**

I. Introducción

El desarrollo de la inteligencia artificial (IA) en los últimos años ha transformado por completo la manera en que interactuamos con los datos visuales, especialmente en el contexto empresarial. Los sistemas de codificación multimodal permiten crear conexiones semánticas entre texto y datos visuales, como imágenes o videos, mediante el uso de modelos de IA como CLIP (Contrastive Language-Image Pretraining) [1], entre otros.

Este avance abre nuevos mercados para las empresas al posibilitar motores de búsqueda visual, sistemas de recomendación y etiquetado automático de datos visuales. Uno de los ejemplos más relevantes es un sistema de búsqueda basado en contenido visual, en el que las compañías utilizan modelos de inteligencia artificial para ofrecer a los clientes servicios como comparación de productos, búsqueda de imágenes y descubrimiento visual [2]. Por ejemplo, al enviar la imagen de un producto de su interés, los clientes pueden encontrar productos similares. Además, las empresas pueden ofrecer estos servicios de motor de búsqueda a otras compañías, utilizando sus propios datos de productos y creando bases de datos con codificaciones multimodales de los datos visuales relacionados con el producto.

Sin embargo, la implementación de estos sistemas presenta importantes desafíos técnicos. El flujo de trabajo típico consiste en codificar las imágenes mediante un modelo de IA, convertir estas representaciones en vectores semánticos e insertarlos en bases de datos optimizadas para búsquedas rápidas. Un cuello de botella crítico en este proceso es la etapa inicial de adquisición de datos: descarga, preprocesamiento y preparación de las imágenes para su integración [3]. Este problema no solo afecta la eficiencia técnica, sino que también incide en la capacidad de las empresas para escalar estos sistemas a nivel comercial, donde la velocidad y la gestión de grandes volúmenes de datos son esenciales para mantener la competitividad.

El objetivo principal de este artículo es optimizar el pipeline de descarga y preparación de imágenes para maximizar la velocidad de inserción en las bases de datos y mejorar el rendimiento general del sistema. El enfoque propuesto incluye descarga en paralelo, uso de pipelines automatizados y procesamiento distribuido, todo ello diseñado para utilizar los recursos de manera eficiente.

Desde una perspectiva empresarial, estas mejoras no solo permiten flujos de trabajo más eficientes, sino que también posibilitan que las compañías ofrezcan servicios más ágiles y escalables. En un mercado donde la experiencia de usuario y la velocidad de respuesta son factores de diferenciación críticos, optimizar las etapas iniciales del flujo de datos puede convertirse en un componente estratégico para maximizar el retorno de la inversión en soluciones de IA.

El resto de este artículo se organiza de la siguiente manera. La Sección II presenta los trabajos relacionados. A continuación, la Sección III expone las preguntas de investigación y la solución propuesta. Posteriormente, la Sección IV presenta y analiza los resultados. Finalmente, la Sección V ofrece las conclusiones del artículo.

II. Trabajos Relacionados

La optimización de bases de datos para sistemas de búsqueda basados en contenido visual está estrechamente vinculada a los sistemas distribuidos subyacentes que les dan soporte. En el comercio electróni-

[1]Departament d'Informàtica, Escola Tècnica Superior d'Enginyeria (ETSE), Universitat de València, e-mail: haojie@alumni.uv.es, carlos.reano@uv.es.

[2]Kimera Technologies, S.L., e-mail: juan.arino@kimeratechnologies.com.

co, donde los sistemas de búsqueda visual y recomendación son fundamentales, la elección de la arquitectura distribuida influye de manera significativa en el rendimiento, la escalabilidad y la fiabilidad. Esta sección explora trabajos previos sobre sistemas distribuidos en comercio electrónico y cómo distintas arquitecturas usan pipelines para optimizar los flujos de trabajo, con especial atención a sus implicaciones para la gestión de bases de datos *Approximate Nearest Neighbor* (ANN).

A. Sistemas distribuidos en el comercio electrónico

Los sistemas distribuidos desempeñan un papel fundamental en las plataformas modernas de comercio electrónico, ya que permiten el manejo escalable y eficiente de grandes volúmenes de datos, incluidos imágenes, interacciones de usuarios y registros de transacciones. Por ejemplo, Yang et al. [3] presentan un sistema que se apoya en almacenamiento y procesamiento distribuidos para gestionar millones de imágenes de forma eficiente. Mediante el uso de micro-lotes (*micro-batches*) y técnicas de *hashing* distribuido, el sistema garantiza que se eviten imágenes duplicadas y que se optimice la ingesta de datos. Los hashes generados durante este proceso se almacenan en Google Bigtable, una base de datos distribuida diseñada para ofrecer alta escalabilidad y baja latencia. Esto pone de relieve la importancia de los sistemas distribuidos para alcanzar un alto rendimiento y una elevada fiabilidad en aplicaciones de comercio electrónico.

Otro estudio relevante de Zheng et al. [4] demuestra cómo las arquitecturas distribuidas mejoran la eficiencia de los motores de búsqueda visual. Técnicas como la partición de datos, la replicación y el uso de bases de datos distribuidas aceleran las consultas y reducen la latencia. Estos resultados subrayan el papel crítico de los sistemas distribuidos en la optimización del rendimiento de los sistemas de búsqueda basados en contenido visual, especialmente cuando se gestionan conjuntos de datos a gran escala. Además, estas arquitecturas influyen directamente en el rendimiento de las bases de datos basadas en ANN al mejorar el preprocesamiento de los datos, la indexación y la recuperación.

B. Pipelines en arquitecturas distribuidas

Numerosas arquitecturas distribuidas utilizan pipelines para optimizar los flujos de trabajo en el comercio electrónico y en otros dominios. Las siguientes subsecciones exploran enfoques clave, destacando su impacto en la optimización de la base de datos ANN.

B.1 Event-Driven Pipeline Framework (EPypes)

EPypes [5] es un marco de software y una arquitectura que permiten desarrollar sistemas distribuidos basados en pipelines. Facilita la descomposición de operaciones complejas en tareas más pequeñas e interrelacionadas que pueden ejecutarse en un entorno distribuido. Por ejemplo, en una plataforma de comercio electrónico, EPypes puede emplearse para construir un pipeline de procesamiento de imágenes en la que cada etapa (por ejemplo, descarga, preprocesamiento y codificación) es gestionada por un componente diferente.

Por ejemplo, Semeniuta et al. [6] utilizan la definición de canalización (pipeline) para desarrollar algoritmos de visión integrados con un sistema de distribución de comunicaciones de tipo publicación–suscripción que facilita la detección de bordes y el emparejamiento de características, demostrando que su sobrecarga computacional es mínima en comparación con el tiempo de procesamiento inherente de la aplicación.

B.2 Arquitecturas dirigidas por eventos

Las arquitecturas dirigidas por eventos (*event-driven*), como las construidas con RabbitMQ [7], Apache Kafka [8], Amazon Web Services (AWS), Simple Queue Service (SQS) [9] y otros sistemas de colas similares, se basan en colas de mensajes para coordinar operaciones de forma asíncrona. Cada tarea del pipeline genera eventos que son consumidos por las tareas posteriores. En los sistemas de búsqueda visual, un pipeline basado en eventos puede constar de etapas como la ingesta de imágenes, la extracción de características y la inserción en la base de datos.

Hinze et al. [10] analizan los principios fundamentales y las aplicaciones de las arquitecturas dirigidas por eventos, destacando su uso en la detección de fraude, la monitorización del tráfico y las infraestructuras inteligentes. Estas arquitecturas permiten el procesamiento en tiempo real, lo que las hace especialmente útiles para las actualizaciones de bases de datos ANN, donde nuevos embeddings deben integrarse de manera eficiente en los índices existentes. No obstante, retos como el ordenamiento de mensajes, la latencia y la tolerancia a fallos siguen siendo consideraciones clave.

C. Arquitecturas de comunicación en sistemas distribuidos

Aunque los pipelines y los frameworks proporcionan estructura a los flujos de trabajo distribuidos, la elección de la arquitectura de comunicación influye de manera significativa en el rendimiento del sistema. Esta sección examina investigaciones recientes que comparan distintos paradigmas de comunicación relevantes para aplicaciones con alta carga computacional.

C.1 Comunicación síncrona frente a asíncrona

La distinción entre comunicación síncrona y asíncrona tiene profundas implicaciones para el diseño de sistemas distribuidos. Kumar et al. [11] caracterizan los protocolos de comunicación REST, gRPC y Apache Thrift en arquitecturas de microservicios, demostrando que gRPC y Thrift logran tiempos de respuesta significativamente menores que REST gracias a la serialización binaria y a las ventajas de HTTP/2. Su trabajo muestra que los Unix

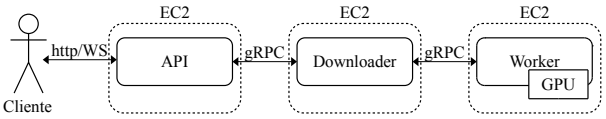


Fig. 1: Arquitectura que utiliza gRPC y colas de multiprocesamiento.

Domain Sockets pueden reducir el tiempo de respuesta en comunicaciones que se producen dentro de un mismo host. Por su parte, Weerasinghe et al. [12] demuestran que la selección de mecanismos de comunicación apropiados es crítica para reducir el tiempo de respuesta, ya que la sobrecarga de red representa uno de los principales cuellos de botella de rendimiento en arquitecturas de microservicios.

En cuanto a la comunicación asíncrona, Dobbelaere y Esmaili [13] comparan Apache Kafka y RabbitMQ, demostrando que la elección del sistema depende en gran medida de los requisitos del caso de uso: RabbitMQ destaca en la mensajería tradicional basada en colas con enrutamiento complejo, mientras que Kafka está optimizado para streaming de alto rendimiento y gran caudal de datos.

C.2 Implicaciones para los sistemas de codificación de imágenes

La literatura revisada establece principios clave para las arquitecturas de comunicación en aplicaciones con alta demanda computacional: (i) los modelos de comunicación directa minimizan la sobrecarga asociada a las colas y maximizan la utilización de la GPU [14], [15], (ii) los gestores de mensajes aportan tolerancia a fallos a costa de cierta pérdida de rendimiento [13], y (iii) la arquitectura de comunicación influye directamente en la utilización de la GPU, de modo que los patrones eficientes mantienen un alto aprovechamiento de los recursos [14]. Estos resultados motivan nuestra comparación experimental de arquitecturas de comunicación para la codificación de imágenes a gran escala, analizando cómo las decisiones arquitectónicas afectan a la utilización de la GPU y al rendimiento global del sistema.

III. Solución Propuesta

En esta sección, describimos primero la arquitectura actual y cómo operan y se comunican nuestros pipelines de procesamiento de imágenes. A continuación, proponemos una nueva arquitectura y la detallamos.

A. Arquitectura actual: gRPC y colas de multiprocesamiento

Como se muestra en Figura 1, el sistema está diseñado para gestionar de forma eficiente la descarga y el procesamiento de imágenes. Consta de dos componentes principales: Downloaders y Workers.

A.1 Downloaders

El papel de este componente es extraer imágenes desde URLs externas y realizar un preprocesamiento básico. Las principales tareas de estos componentes son:

- Descargar imágenes desde URLs externas.
- Realizar el preprocesamiento de imágenes (validación, redimensionado, centrado y conversión a tensores) utilizando la librería Pillow [16], un fork actualizado de la Python Imaging Library (PIL).
- Enviar los tensores preprocesados a los Workers mediante gRPC [17], un marco de llamadas a procedimiento remoto (remote procedure call, RPC) universal, de alto rendimiento y de código abierto.

A.2 Workers

El papel de este componente es codificar los tensores de imágenes en embeddings (un formato de datos que puede manipularse matemáticamente para búsquedas visuales) utilizando el modelo CLIP. Las principales tareas de estos componentes son:

- Recibir tensores de imágenes preprocesadas desde los Downloaders mediante gRPC.
- Utilizar colas de multiprocesamiento para gestionar las tareas procesadas por el modelo CLIP.
- Generar embeddings a partir de los tensores de imágenes y almacenar los embeddings resultantes en la base de datos de búsqueda visual.

El punto fuerte de la arquitectura actual radica en que permite el uso de múltiples procesos, lo que posibilita una distribución eficiente de la carga de trabajo, paralelismo y una escalabilidad sencilla gracias a la separación entre Downloaders y Workers.

Pero también existen varias debilidades en la arquitectura actual. Por ejemplo, el uso de comunicación síncrona, debido a que gRPC sigue un modelo estricto de solicitud–respuesta, puede generar cuellos de botella. Así, si los Workers procesan las solicitudes lentamente, los Downloaders deberán esperar la respuesta, reduciendo el rendimiento global del sistema. Cabe señalar que cada Worker dispone de una cola de multiprocesamiento para CLIP, lo que puede provocar retrasos cuando existe un gran volumen de datos. Otro aspecto a tener en cuenta es que los componentes nunca se apagan, incluso si no tienen trabajo durante largos periodos de tiempo, lo que no constituye un enfoque especialmente eficiente desde el punto de vista de coste.

B. Nueva arquitectura propuesta: dirigida por eventos y con RabbitMQ

Para abordar estas limitaciones, proponemos una nueva arquitectura dirigida por eventos que utiliza RabbitMQ [7] para la cola de mensajes.

Tal y como se muestra en Figura 2, esta arquitectura desacopla los Downloaders y los Workers, lo que permite una mayor escalabilidad y tolerancia a fallos. Mediante el uso de RabbitMQ, implementamos una arquitectura asíncrona dirigida por eventos. Este diseño desacopla los Downloaders y los Workers, favoreciendo una mejora tanto en la escalabilidad como

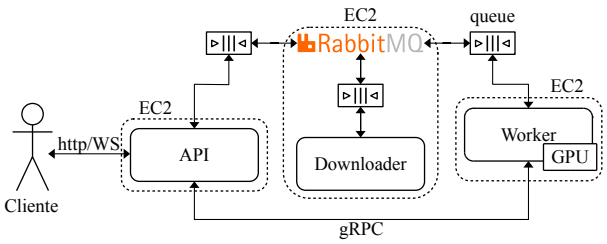


Fig. 2: Arquitectura dirigida por eventos utilizando RabbitMQ para la cola de mensajes.

en la tolerancia a fallos.

B.1 Downloaders

En este caso, los Downloaders eliminarán toda comunicación directa, se suscribirán a una cola de RabbitMQ para recibir tareas de descarga de imágenes y continuarán descargando y preprocesando las imágenes en distintos hilos. En lugar de enviar los datos directamente a los Workers mediante gRPC, los Downloaders publican mensajes en una cola de RabbitMQ a la que los Workers están suscritos (de modo que ahora los Downloaders actúan como consumidores y productores dentro de RabbitMQ). De esta forma, los Downloaders no esperan las respuestas de los Workers, eliminando la necesidad de comunicación síncrona.

B.2 Workers

Como se ha explicado anteriormente, los Workers se suscribirán a la cola de RabbitMQ para recibir los mensajes (RabbitMQ garantiza una entrega fiable de los mensajes y proporciona balanceo de carga entre múltiples Workers), actuando así los Workers como consumidores de RabbitMQ. De este modo, una vez que se obtiene el mensaje con los datos, un Worker lo envía a la cola de multiprocesamiento de CLIP y, posteriormente, genera y almacena los embeddings resultantes en la base de datos.

Los beneficios de la arquitectura dirigida por eventos se ven reforzados en comparación con la arquitectura actual. En primer lugar, la escalabilidad facilita el escalado horizontal tanto de los Workers como de los Downloaders. En segundo lugar, esta arquitectura incrementa la flexibilidad del sistema al permitir que los Downloaders y los Workers funcionen de manera independiente. Además, RabbitMQ ofrece tolerancia a fallos, lo que le permite reasignar trabajos a otros Workers o Downloaders en caso de que alguna de las tareas falle, garantizando así la continuidad del servicio.

No obstante, también existen algunos posibles inconvenientes, como una mayor latencia y un posible ligero aumento del tiempo de procesamiento debido a que los mensajes deben atravesar las colas. Además, para minimizar latencias adicionales, es necesario diseñar un sistema de colas eficiente.

C. Metodología experimental

En la siguiente sección, cada arquitectura se evalúa utilizando cargas de trabajo idénticas procesadas mediante un patrón de envío controlado. Un ThreadPool con cuatro hilos concurrentes envía imágenes de forma secuencial a partir del proceso actual de creación de la base de datos visual, comenzando con 4,500 imágenes y aumentando progresivamente el número de usuarios que crean bases de datos.

Las métricas clave incluyen el tiempo de procesamiento de extremo a extremo (desde la primera entrega de imágenes hasta el almacenamiento final de los embeddings), la utilización de CPU y GPU, el uso de memoria, y el comportamiento del sistema bajo condiciones de carga creciente.

IV. Evaluación de prestaciones

Esta sección presenta una evaluación experimental de las dos arquitecturas de comunicación analizadas para sistemas de codificación de imágenes a gran escala. Diseñamos experimentos controlados para medir el rendimiento y la rentabilidad en distintos patrones de carga de trabajo.

A. Configuración experimental

A continuación detallamos la configuración experimental utilizada en este trabajo. En primer lugar, describimos la configuración de hardware e infraestructura empleada en los experimentos. Posteriormente, especificamos el conjunto de datos y la carga de trabajo utilizados. Por último, analizamos las configuraciones específicas en función de la arquitectura empleada en los experimentos.

A.1 Configuración de hardware e infraestructura

Todos los experimentos se llevan a cabo sobre configuraciones de hardware estandarizadas con el fin de garantizar una comparación justa. En concreto, la configuración de hardware e infraestructura empleada en los experimentos es la siguiente:

- Downloaders y gestor RabbitMQ: una instancia AWS EC2 t3.xlarge (4 vCPUs, 16GB RAM).
- Workers: una instancia AWS EC2 G5.2xlarge (8 vCPUs, 32GB RAM, 1 NVIDIA A10G Tensor Core).
- Red: todas las instancias están en la misma zona de disponibilidad AWS (eu-west-1a).

A.2 Especificación del conjunto de datos y la carga de trabajo

El conjunto de datos y la carga de trabajo utilizados para las pruebas son los siguientes:

- Características de las imágenes: resolución desde 640×480 a 1024×768, el tamaño de fichero es de alrededor 50KB-100KB.
- Tamaño de las cargas de trabajo: 4.500; 9.000; 13.500; 18.000 imágenes.
- Patrón de envío: 4 hilos concurrentes enviando imágenes de manera secuencial.

Se utiliza el modelo SigLIP (variante ViT-B/16) [18], el cual procesa las imágenes para generar vectores de embeddings de 768 dimensiones,

representando cada imagen en un espacio vectorial semánticamente significativo.

A.3 Configuraciones específicas para cada arquitectura

Las configuraciones específicas en función de la arquitectura utilizada en los experimentos son las siguientes.

Arquitectura gRPC (Figura 1):

- Agrupación de conexiones (*connection pooling*) con 4 canales persistentes por Downloader.
- Compresión gRPC no habilitada.
- Tiempo de espera de las solicitudes (*request timeout*): 180 segundos.

Arquitectura RabbitMQ (Figura 2):

- Dos colas: una para los Downloaders y otra para los Workers.
- Durabilidad de las colas: habilitada para tolerancia a fallos.
- Prefetch count: 50 mensajes por Worker.
- Persistencia de mensajes: habilitada.
- Dead letter queue: configurada para los mensajes fallidos.

B. Métricas de rendimiento

Medimos los siguientes indicadores clave de rendimiento:

- Latencia de extremo a extremo: tiempo desde el envío de la primera imagen hasta el almacenamiento final del embedding.
- Rendimiento medio: número de imágenes procesadas por segundo a lo largo de toda la carga de trabajo.
- Utilización de la GPU: uso medio de la GPU durante el procesamiento.
- Utilización de la CPU: uso medio de la CPU a lo largo de todos los componentes.
- Uso de memoria: consumo máximo de memoria por componente.

C. Resultados experimentales

En esta sección evaluamos las arquitecturas de comunicación anteriormente detalladas. Comenzamos con una comparación del rendimiento de extremo a extremo para establecer una referencia. A continuación, analizamos la utilización de recursos con el fin de cuantificar la eficiencia computacional y las demandas de hardware asociadas a cada enfoque.

C.1 Comparación de rendimiento de extremo a extremo

Figura 3 y Figura 4 presentan un análisis de prestaciones para todas las arquitecturas y tamaños de carga de trabajo, mostrando tanto las métricas de tiempo de procesamiento como de rendimiento. A partir de los resultados mostrados en estas tablas, analizamos a continuación el rendimiento de cada una de las arquitecturas evaluadas.

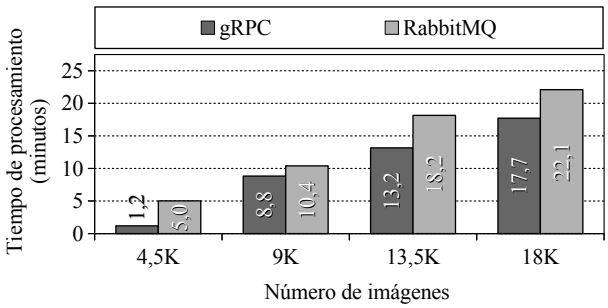


Fig. 3: Tiempo de procesamiento de las arquitecturas evaluadas.

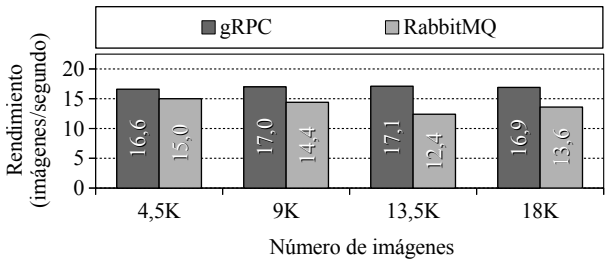


Fig. 4: Rendimiento de las arquitecturas evaluadas.

- Arquitectura gRPC: Obtiene el rendimiento más alto y más consistente en todos los tamaños de carga de trabajo (∼17 imágenes por segundo), con una degradación mínima del rendimiento a medida que el conjunto de datos escala. La comunicación directa entre Downloaders y Workers elimina la sobrecarga asociada a las colas, lo que se traduce en una utilización óptima de los recursos y en un escalado del rendimiento predecible.
- Arquitectura RabbitMQ: Muestra patrones de prestaciones variables, con un rendimiento que oscila entre 12,4 y 15,0 imágenes por segundo. En particular, la arquitectura experimenta una degradación del rendimiento con una carga de trabajo de 13,5K imágenes (12,4 img/s), pero recupera eficiencia a cargas mayores (13,6 img/s para 18K imágenes). Este comportamiento sugiere una sobrecarga asociada a la gestión de colas que se vuelve más eficiente bajo un procesamiento sostenido de alto volumen. Una optimización adicional de la configuración de las colas, los valores de prefetch y las estrategias de agrupamiento de mensajes podría mejorar potencialmente estos resultados.

C.2 Utilización de recursos

El análisis de la utilización de recursos (Tabla I y Tabla II) se centra en los periodos de procesamiento activo, excluyendo las fases de configuración y los tiempos de inactividad entre experimentos. Cada arquitectura fue monitorizada durante cuatro cargas de trabajo secuenciales (4,5K, 9K, 13,5K y 18K imágenes). Como puede observarse en dichas tablas, los resultados varían en función de la arquitectura:

- Arquitectura gRPC: durante los periodos de procesamiento activo, esta arquitectura mues-

Tabla I: Utilización de la GPU durante los periodos de procesamiento activo.

Arquitectura	4,5K imgs	9K imgs	13,5K imgs	18K imgs
gRPC	95,5 %	93,6 %	94,8 %	94,9 %
RabbitMQ	68,2 %	74,5 %	71,5 %	72,8 %

Tabla II: Utilización de la CPU y estabilidad durante el procesamiento.

Arquitectura	CPU Avg (%)	CPU Std Dev (%)	CPU Pico (%)
gRPC	36,4	1,9	42,1
RabbitMQ	35,71	8,6	64,7

tra una mayor utilización de la GPU (superior al 90 %), independientemente del tamaño de la carga de trabajo. La utilización de la CPU durante el procesamiento presenta una excelente estabilidad, con valores medios por debajo del 40 % y algunos picos que superan dicho valor. Este mayor aprovechamiento de los recursos explica los mejores resultados previamente observados en términos de tiempo de procesamiento y rendimiento de esta arquitectura.

- Arquitectura RabbitMQ: muestra la menor utilización de la GPU (en torno al 70 %). La utilización media de la CPU (por encima del 35 %) es similar a la de otras arquitecturas; sin embargo, se observan algunos picos que superan el 64 %, lo que reduce la estabilidad del sistema.

V. Conclusiones

Este estudio demuestra que la selección de la arquitectura de comunicación influye de manera significativa en el rendimiento del sistema en aplicaciones de codificación de imágenes a gran escala. La elección entre arquitecturas debe estar guiada por requisitos específicos: gRPC para un rendimiento máximo y RabbitMQ para tolerancia a fallos.

En futuros trabajos, queremos extender el presente estudio incluyendo arquitecturas *serverless*, utilizando AWS Lambda [19] o servicios como AWS SageMaker [20].

Por último, puntualizar que el marco comparativo desarrollado en este estudio proporciona una base para la evaluación de arquitecturas de comunicación emergentes y puede extenderse a otras aplicaciones de IA con alta demanda computacional más allá de los sistemas de búsqueda visual.

Agradecimientos

Este trabajo ha sido financiado por la Agencia Valenciana de Innovación (AVI) mediante el proyecto INNTA3/2023/17. Los autores agradecen también el generoso apoyo prestado por la empresa Kimera Technologies.

Referencias

[1] Alec Radford et al., "Learning transferable visual models from natural language supervision," in *Proceedings of the 38th International Conference on Machine Learning*, Marina Meila and Tong Zhang, Eds. 18–24 Jul 2021, vol. 139 of *Proceedings of Machine Learning Research*, pp. 8748–8763, PMLR.

[2] Yanhao Zhang, Pan Pan, Yun Zheng, Kang Zhao, Yingya Zhang, Xiaofeng Ren, and Rong Jin, "Visual search at alibaba," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, New York, NY, USA, 2018, KDD '18, p. 993–1001, Association for Computing Machinery.

[3] Fan Yang, Ajinkya Kale, Yury Bubnov, Leon Stein, Qiaosong Wang, Hadi Kiapour, and Robinson Piramuthu, "Visual search at ebay," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2017, KDD '17, p. 2101–2110, Association for Computing Machinery.

[4] Yu Zheng, Xing Xie, and Wei-Ying Ma, "Distributed architecture for large scale image-based search," in *2007 IEEE International Conference on Multimedia and Expo*, 2007, pp. 579–582.

[5] Oleksandr Semeniuta, "EPypes: a Python library for developing event-driven pipelines," 2025, Accedido: 14 abril 2026.

[6] Oleksandr Semeniuta and Petter Falkman, "Epypes: a framework for building event-driven data processing pipelines," *PeerJ Computer Science*, vol. 5, pp. e176, 2019.

[7] Broadcom Inc., "RabbitMQ: One broker to queue them all," 2025, Accedido: 14 abril 2026.

[8] Apache Software Foundation, "Apache Kafka," 2025, Accedido: 14 abril 2026.

[9] Amazon Web Services, "Amazon Simple Queue Service (Amazon SQS)," 2025, Accedido: 14 abril 2026.

[10] Annika Hinze, Kai Sachs, and Alejandro Buchmann, "Event-based applications and enabling technologies," in *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, New York, NY, USA, 2009, DEBS '09, Association for Computing Machinery.

[11] Prajwal Kiran Kumar, Radhika Agarwal, Rahul Shivaprasad, Dinkar Sitaram, and Subramaniam Kalambur, "Performance characterization of communication protocols in microservice applications," in *2021 International Conference on Smart Applications, Communications and Networking (SmartNets)*, 2021, pp. 1–5.

[12] L.D.S.B Weerasinghe and I Perera, "Evaluating the inter-service communication on microservice architecture," in *2022 7th International Conference on Information Technology Research (ICITR)*, 2022, pp. 1–6.

[13] Philippe Dobbelaere and Kyumars Sheykh Esmaili, "Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper," in *Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems*, New York, NY, USA, 2017, DEBS '17, p. 227–238, Association for Computing Machinery.

[14] Seonho Lee, Jihwan Oh, Seokjin Go, and Divya Mahajan, "Characterizing compute-communication overlap in gpu-accelerated distributed deep learning: Performance and power implications," in *2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2025, pp. 353–355.

[15] Md Sultanul Islam Ovi, "A study on distributed strategies for deep learning applications in gpu clusters," 2026.

[16] Jeffrey A. Clark, Fredrik Lundh, and contributors, "Pillow (PIL Fork)," 2025, Accedido: 14 abril 2026.

[17] gRPC Authors, "gRPC: A high performance, open source universal RPC framework," 2025, Accedido: 14 abril 2026.

[18] Michael Tschannen et al., "Siglip 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features," 2025, Accedido: 14 abril 2026.

[19] Amazon Web Services, "AWS Lambda: serverless function, FaaS serverless," 2025, Accedido: 14 abril 2026.

[20] Amazon Web Services, "Amazon SageMaker: The center for all your data, analytics, and AI," 2025, Accedido: 14 abril 2026.

Tecnologías clúster, plataformas distribuidas, BigData y Deep Learning (2/4)

Efficient evaluation of video encoding acceleration techniques: methodology and case study

Xiangyu Wan[1], Clara Baltasar[1], Guillermo Vigueras[1], and Antonio Jesús Díaz-Honrubia[1]

Resumen— **The increasing complexity of modern video coding standards, especially Versatile Video Coding (VVC), demands efficient encoding strategies that often use Machine Learning (ML) to make encoder and decoder implementations practical. Conventional evaluation procedures require substantial computational effort, frequently leading to hundreds of hours of encoding, particularly when multiple models are analyzed. This work introduces a novel methodology for the systematic evaluation and comparison of early-stage ML-based solutions aimed at accelerating video encoders, allowing assessments within feasible timeframes while maintaining consistency across comparisons. A case study explores the use of several ML classifiers to speed up the first-level quadtree partitioning in VVC, achieving a 19.89 % reduction in encoding time with only a 1.16 % increase in BD-rate.**

Palabras clave— **Video coding, evaluation, methodology.**

I. Introduction

DUE to the increasing quality of videos and the growing popularity of Ultra-High Definition (UHD), 2K, and 4K formats, new efficiency requirements for video compression have emerged. To address these needs, the Versatile Video Coding (VVC) standard [1] was released by the Joint Video Experts Team (JVET) in 2020 as the latest video encoding technology. Compared to High Efficiency Video Coding (HEVC) [2], VVC achieves around a 50 % bitrate reduction while maintaining similar subjective quality [3]. However, this improvement comes with a significant increase in computational complexity, making encoding very time-consuming.

Like HEVC, VVC uses Coding Tree Units (CTUs), with Coding Units (CUs) up to 128×128 pixels, and extends quadtree partitioning through the Quadtree Plus Multi-Type Tree (QT+MTT) structure. A CU can be split into square blocks (quad split) or rectangular ones using binary or ternary splits in both directions. This flexible partitioning, together with multiple coding modes, greatly increases encoding complexity.

VVC encoders compute the Rate-Distortion (RD) cost for each CU and mode, selecting the optimal configuration. However, the large number of partitioning options and modes significantly increases computational cost.

Many ML-based methods have been proposed to accelerate encoding, but their refinement requires extensive testing under standard conditions, which is computationally expensive. To overcome this limitation, this paper proposes a methodology that enables efficient validation in reduced time, especially when evaluating multiple models.

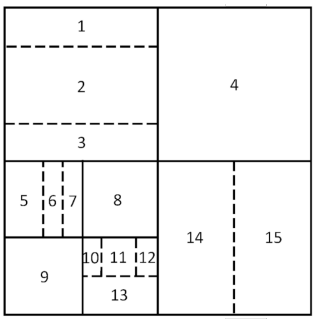


Fig. 1: Example of CTU partitioning in VVC.

The rest of this paper is organized as follows. In Section II, we present a brief technical background and a summary of the state of the art in this field. Section III introduces the proposed methodology, including the definition of the new BTR metric. Then, Section IV presents a use case in which a new fast VVC encoding algorithm is proposed to illustrate the use of the methodology. Finally, the main conclusions are drawn in Section V.

II. Technical background and related work

The partitioning scheme in VVC is a clear evolution over earlier standards. HEVC uses CTUs up to 64×64 pixels with recursive Quad Tree (QT) splitting into CUs as small as 8×8. In contrast, VVC adopts larger 128×128 CTUs and introduces the Multi-Type Tree (MTT) structure [4], improving flexibility through a two-step process: QT splitting into four equal blocks, followed by Binary Tree (BT) and Ternary Tree (TT) splits (with BT only for 128×128 nodes). Figure 1 illustrates this combined partitioning scheme.

Due to recursive splitting and multiple coding modes (inter, intra, affine, skip, merge, etc.) [3], VVC encoding is highly time-consuming. Evaluations under the Common Test Conditions (CTCs) [5] can take months. For example, full RA encoding with four QPs may require up to 117 days on a single CPU. Therefore, a unified methodology is essential to accelerate model development and enable more efficient experimentation. To the best of the authors' know-

[1]Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingenieros Informáticos, Madrid, Spain, e-mail: {xiangyu.w, cm.baltasar, guillermo.vigueras, antoniojesus.diaz}@upm.es.

ledge, no prior work defines a unified methodology for refining fast video coding algorithms. The CTCs represent the only standardization effort, but their high computational cost limits their practical use.

Existing works mainly focus on accelerating VVC encoding. García-Lucas *et al.* [6] proposed a Naïve-Bayes model to predict QT splits of 128×128 blocks, achieving 13 % time reduction with 0.3 % BD-rate increase, later extended in [7]. CNN-based approaches, such as [8], reduce encoding time by 30.63 % with around 3 % BD-rate loss through early CU partition termination.

Random Forest models have also been explored. Amestoy *et al.* [9] introduced risk-controlled predictions, achieving 30.1 % time reduction with 0.61 % BD-rate increase, while Lindino *et al.* [10] reported 34.76 % time savings with 1.03 % loss. Non-ML approaches, such as quadtree parallelization [11], achieve up to 50.98 % time reduction at the cost of higher BD-rate (5.11 %).

III. Proposed methodology for comparing video coding acceleration algorithms

This section presents the proposed methodology for efficiently refining and comparing video coding acceleration algorithms, particularly during validation. It is based on the CTC framework, specifically the widely used Random Access configuration, where the 21 sequences are grouped into five classes:

- Class A1: 4K, 3840×2160 pixels.
- Class A2: 4K, 840×2160 pixels.
- Class B: Full HD, 1920×1080 pixels.
- Class C: WVGA, 832×480 pixels.
- Class D: WQVGA, 416×240 pixels.

From these sequences, a representative subset was selected for validation, ensuring coverage of different resolutions and spatial-temporal characteristics. One sequence per class was chosen based on their Spatial Index (SI) and Temporal Index (TI), as defined in ITU-T P.910 [12], which describe texture and motion complexity. Figure 2 shows their distribution. Sequences with both high SI and TI are rare, as strong motion typically reduces fine details. The selected sequences represent a wide range of content characteristics:

- **Campfire:** Class A1 sequence, low SI, high TI.
- **RollerCoaster2:** Class A2 sequence, low SI, medium TI.
- **ParkScene:** Class B sequence, low SI, low TI.
- **BQMall:** Class C sequence, medium SI, medium TI.
- **BQSquare:** Class D sequence, high SI, low TI.

To compute the BD-rate, four QP values (22, 27, 32, 37) are used. However, full encoding with VTM 20.0 [13] exceeds 525 hours, so in the proposed methodlogy only the first seven frames in coding order are encoded, generating two intra and five inter frames. Despite presentation gaps, only bitrate, PSNR, and encoding time are required, and results

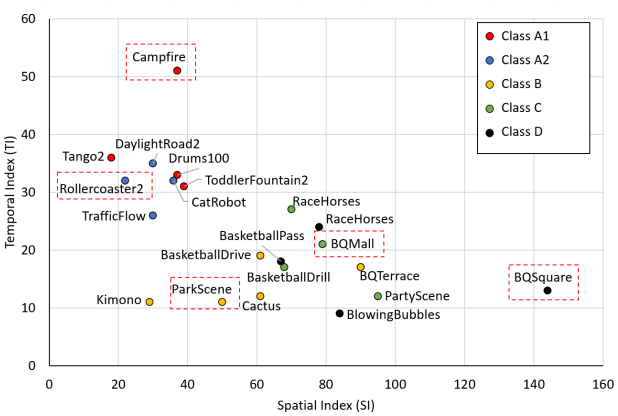


Fig. 2: SI and TI values of the CTC sequences under the *Random Access* configuration. The sequences highlighted in red indicate the selected representative subset.

remain equivalent. This selection ensures all QP offsets for inter frames are represented. Once the subset of sequences and frames is defined, both the reference encoder and the tested encoder are used to encode them at QP values of 22, 27, 32, and 37. From these runs, three metrics are obtained: average bitrate, average PSNR, and total encoding time. The BD-rate is computed from bitrate and PSNR to evaluate compression efficiency, while time savings are determined by comparing the encoding times of both encoders. Many algorithms target either inter- or intra-coding, but not both. Therefore, results from irrelevant frame types can be discarded, focusing the analysis only on the frames affected by the optimized tool.

To validate the chosen subset, the correlations of the proposed subset with the full sequences have been calcuted for the bitrate, the PSNR, and the encoding time, resulting in correlations of 0.9963, 0.9789, and 0.7288, respectively, validating this reduced approach for efficient evaluation.

Finally, Figure 3 illustrates the different steps of the proposed methodology to evaluate various models aimed at accelerating inter-coded frames using the reference VTM 20.0 software for VVC. The steps are as follows:

1. Selection of the proposed representative subset of sequences.
2. Encoding of the first 7 frames from each sequence using the four QP values specified in the CTCs with both, the reference and the modified encoders.
3. Discarding of the two intra-coded frames.
4. Collection of the bitrate, the PSNR, and the encoding time from the encoder's logs.
5. Calculation of the BD-rate.
6. Calculation of the mean of all individual BD-rates and time reductions.

IV. Case study: algorithm to accelerate the first split of the quadtree in VVC

This section presents a case study illustrating the proposed methodology by comparing ML models for accelerating first-level quadtree splitting in VVC inter-coded frames. Models classify 128×128 CUs as

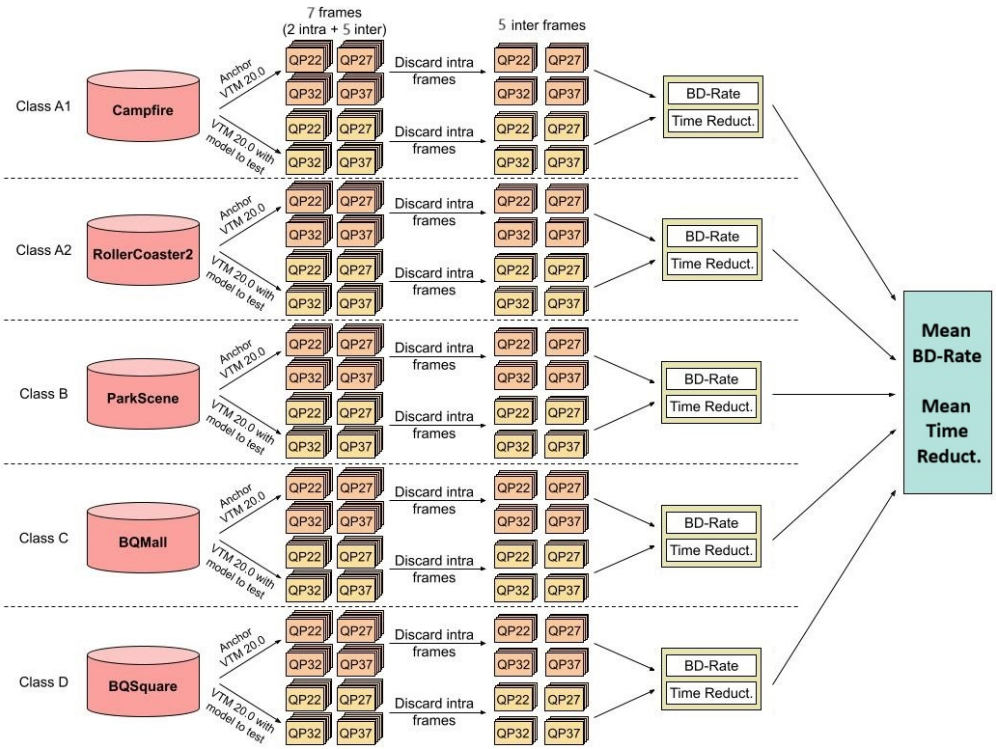


Fig. 3: Proposed methodology for the case of validating inter-coding algorithms using the reference VTM 20.0 encoder for VVC.

split (C_S) or non-split (C_N), either forcing subdivision into 64×64 blocks or evaluating alternative modes.

The dataset generation is first described, followed by the training of 17 models using different ML algorithms. All models are evaluated with the proposed methodology, and the best one is further assessed under CTC conditions, with results compared using the BTR metric.

A. Dataset

A representative dataset is essential for training ML models to predict CU splitting in VVC. The splitting decision is mainly influenced by texture complexity and motion, which can be inferred from luminance information and edge presence. To capture these characteristics, six statistical features are extracted from the CU luminance component:

- **Mean:** average luminance value of the CU.
- **Standard deviation:** measures brightness dispersion, indicating texture complexity.
- **Skewness:** describes the asymmetry of brightness distribution.
- **Kurtosis:** reflects the concentration of values around the mean.
- **Levene's test value:** evaluates variance differences among the four 64×64 sub-CUs.
- **Kruskal-Wallis test value:** assesses differences in mean values across sub-CUs.

To better capture edges and motion, these statistics are also computed after applying the following filters:

- **Sobel filter:** highlights edges using first-order gradients.
- **Laplacian filter:** detects finer details through second-order derivatives.
- **Frame residual:** captures motion by subtracting consecutive frames.

Additionally, the QP is included as a feature, as it affects compression level, image smoothness, and CU size.

The final dataset contains 25 features: six statistics from the original luminance, the same features from three filtered versions, and the QP. Class labels (C_S, C_N) are obtained from VTM 20.0 decisions. Features are denoted as $w_{i,j}$, with w_{QP} for QP and C for the class. All inter-coded CUs from CTC sequences are included as samples.

B. Model learning and hyperparameter tunning

Three ML algorithms were used to train classifiers for predicting the splitting of 128×128 CUs. In total, 17 models were trained using different hyperparameter configurations and evaluated with the proposed methodology.

For the Naïve Bayes (NB) algorithm, only one model was trained, as it has no hyperparameters. To avoid assumptions about feature distributions, all features were discretized using the method in [14]. For Random Forest (RF), combinations of $mss = 2, 5, 10, 15$ and $msl = 5, 10, 25$ were tested. For SVM, both polynomial and RBF kernels were considered.

Feature selection using a forward wrapper [15] was applied to NB and SVM to remove irrelevant features. The selected features for NB were $w_{sobel,std}$, $w_{sobel,kurt}$, $w_{residual,std}$, $w_{residual,kruskal}$, and w_{QP}. For SVM, the selected features were $w_{luma,std}$, $w_{luma,kruskal}$, $w_{sobel,mean}$, $w_{laplace,mean}$, $w_{residual,std}$, $w_{residual,kurt}$, and w_{QP}.

In contrast, RF did not require feature selection, as it inherently performs it during training. Additionally, RF ($mss = 10$, $msl = 25$) and SVM (RBF) were also evaluated on a rebalanced dataset obtained through class weighting proportional to the original class distribution in the dataset.

Overall, 1 NB model, 13 RF models (including one with rebalancing), and 3 SVM models (including one rebalanced) were trained, totaling 17 models.

C. Model selection using the proposed methodology

The proposed methodology was applied using VTM 20.0 as the reference encoder and modified versions incorporating the trained models for CU splitting decisions. Experiments were conducted on an Intel Xeon Silver 4314 CPU at 2.4 GHz.

Table I summarizes the results in terms of BD-rate, and time reduction, averaged over the *Campfire*, *RollerCoaster2*, *ParkScene*, *BQMall*, and *BQSquare* sequences. Each sequence was encoded with seven frames under RA configuration and QPs 22, 27, 32, and 37. Since the models target inter-coded frames, intra-coded frames were excluded. Figure 4 provides detailed heatmaps of BD-rate and time reduction.

According to Table I, the NB model provides the best performance due to its very low BD-rate (1.301 %), despite offering a more moderate time reduction (8.568 %). Other models, such as RF ($mss = 10$, $msl = 25$, rebalanced), SVM with a polynomial kernel, and SVM (RBF, rebalanced), achieve higher time reductions (around 10–15 %), but at the cost of noticeably higher BD-rate values, making them less efficient overall.

Figure 4 shows that most models behave similarly for *RollerCoaster2*, *Campfire*, and *BQMall*, except for the SVM with RBF kernel, which performs poorly in *RollerCoaster2*. Greater variability appears in *BQSquare*, while *ParkScene* only shows good results for the top-performing models. Among them, NB stands out for its more consistent behavior across all sequences.

Considering its minimal BD-rate increase, stable performance, and low complexity (linear, $\mathcal{O}(\mathtt{n})$), the NB model is selected for the proposed fast VVC encoding algorithm.

D. Performance evaluation using the CTCs

The selected Naïve-Bayes model, integrated into the VTM 20.0 encoder, was used for the final evaluation following the full CTC guidelines under the RA configuration. Experiments were performed on the same Intel Xeon Silver 4314 CPU at 2.4 GHz. The results of this fast encoding algorithm are presented in Table II, showing an average time reduction of 19.89 % with a low BD-rate increase of 1.16 %.

These results improve upon those obtained during validation, with lower BD-rate and higher time reduction. This confirms that the proposed validation subset properly represents the full set of sequences. For the subset (*Campfire*, *RollerCoaster2*, *ParkScene*, *BQMall*, *BQSquare*), the BD-rate range is [0,1480, 3,9787], while for all sequences it is [−0,0126, 3,9787], showing strong overlap. Similarly, time reduction ranges are [6,8981, 38,7248] for the subset and [1,8989, 43,6475] overall.

To quantify this similarity, the Jaccard index [16], defined as $J(A, B) = |A \cap B|/|A \cup B|$, is used. The obtained values are 0.96 for BD-rate and 0.76 for time reduction, indicating a high agreement between the reduced subset and the full evaluation.

V. Conclusions

This work presents a methodology to validate video encoding acceleration algorithms without executing the full set of CTC tests. Instead, a reduced subset of 5 sequences and 7 frames per sequence is used, providing a reliable approximation of the overall results. Nevertheless, final evaluation must still follow the complete CTC guidelines.

The methodology is mainly intended for early-stage ML-based approaches, enabling faster validation while maintaining consistency with full evaluations.

A use case is also presented, where an ML-based algorithm is proposed to accelerate VVC encoding. A dataset is constructed from 128×128 CUs, and several models are trained and compared using the proposed methodology. The Naïve-Bayes model achieves the best balance, and its full evaluation shows a 19.89 % reduction in encoding time with a limited BD-rate increase of 1.16 %.

Agradecimientos

This work has been supported by the Spanish Ministry of Science and Innovation, the Agencia Estatal de Investigación (10.13039/501100011033) and the European Commission (FEDER: A way to make Europe) under projects PID2024-161538OB-I00, and PREP2024-003139.

Referencias

[1] ISO/IEC and ITU-T, "Versatile Video Coding (VVC). ITU-T Recommendation H.266 and ISO/IEC 23090-3," Video standard, 2020.

[2] ISO/IEC and ITU-T, "High Efficiency Video Coding (HEVC). ITU-T Recommendation H.265 and ISO/IEC 23008-2," Video standard, 2013.

[3] Benjamin Bross, Ye-Kui Wang, Yan Ye, Shan Liu, Jianle Chen, Gary J Sullivan, and Jens-Rainer Ohm, "Overview of the versatile video coding (VVC) standard and its applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3736–3764, 2021.

[4] Y.-W. Huang, J. An, H. Huang, X. Li, S.-T. Hsiang, K. Zhang, H. Gao, J. Ma, and O. Chubach, "Block partitioning structure in the VVC standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3818–3833, 2021.

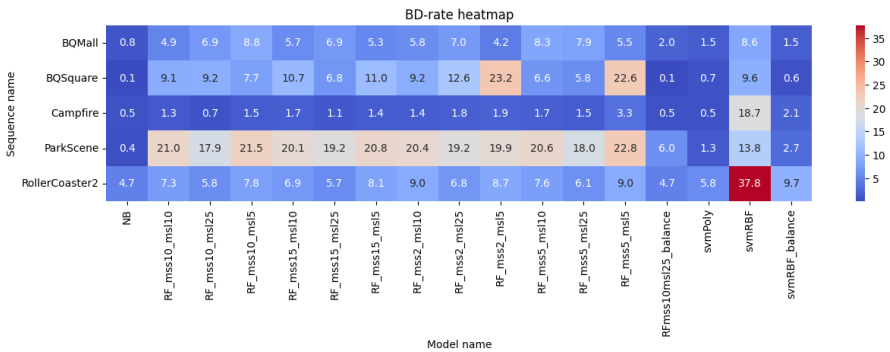
[5] X. Li and K. Suehring, "Common Test Conditions and Software Reference Configurations," in *Technical Report JVET-H1010, Joint Video Experts Team (JVET)*, 2017.

[6] David García-Lucas, Gabriel Cebrián-Márquez, Antonio Jesús Díaz-Honrubia, Thanuja Mallikarachchi, and Pedro Cuenca, "Cost-efficient HEVC-based quadtree splitting (HEQUS) for VVC Video Transcoding," *Signal Processing: Image Communication*, vol. 94, pp. 116199, 2021.

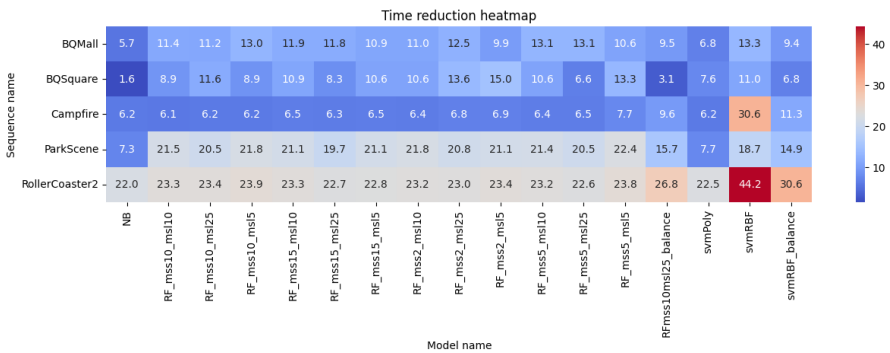
[7] D. García-Lucas, G. Cebrián-Márquez, A.J. Díaz-Honrubia, T. Mallikarachchi, and P. Cuenca, "A fast full

Tabla I: Results of applying the proposed methodology to the 17 models obtained from the derived dataset. They show that the NB-based model offers the best trade-off between BD-rate and time reduction when the BTR metric is used.

Model	BD-rate (%)	Time reduction (%)
NB	1,301	8,568
$RF_{mss=10,msl=25}$	8,099	14,575
$RF_{mss=10,msl=25,rebalanced}$	2,671	12,955
$RF_{mss=10,msl=10}$	8,728	14,242
$RF_{mss=10,msl=5}$	9,459	14,774
$RF_{mss=15,msl=25}$	7,947	13,754
$RF_{mss=15,msl=10}$	9,023	14,738
$RF_{mss=15,msl=5}$	9,320	14,406
$RF_{mss=5,msl=25}$	7,858	13,843
$RF_{mss=5,msl=10}$	8,987	14,937
$RF_{mss=5,msl=5}$	12,632	15,564
$RF_{mss=2,msl=25}$	9,465	15,330
$RF_{mss=2,msl=10}$	9,154	14,593
$RF_{mss=2,msl=5}$	11,593	15,271
$SVM_{polynomial}$	1,954	10,156
SVM_{RBF}	17,686	23,569
$SVM_{RBF,rebabalaced}$	3,345	14,598



(a) BD-rate (%) results of the different ML models across the different sequences.



(b) Time reduction (%) results of the different ML models across the different sequences.

Fig. 4: Comparison of BD-rate and time reduction achieved by the different ML models across the selected sequences in the proposed methodology.

partitioning algorithm for HEVC-to-VVC video transcoding using Bayesian classifiers," *Journal of Visual Communication and Image Representation*, vol. 94, pp. 103829, 2023.

[8] Zhaoqing Pan, Peihan Zhang, Bo Peng, Nam Ling, and Jianjun Lei, "A CNN-based fast inter coding method for VVC," *IEEE Signal Processing Letters*, vol. 28, pp. 1260–1264, 2021.

[9] Thomas Amestoy, Alexandre Mercat, Wassim Hamidouche, Daniel Menard, and Cyril Bergeron, "Tunable VVC Frame Partitioning Based on Lightweight Machine Learning," *IEEE Transactions on Image Processing*, vol. 29, pp. 1313–1328, 2020.

[10] Matheus Lindino, Bruno Zatt, Mateus Grellert, and Guilherme Correa, "Low-Complexity Multi-Type Tree Partitioning for Versatile Video Coding Based on Machine Learning," in *2022 IEEE International Conference on Image Processing (ICIP)*, 2022, pp. 3616–3620.

[11] Alberto González-Ruiz, Antonio Jesús Díaz-Honrubia, Santiago Tapia-Fernández, David García-Lucas, Gabriel Cebrián-Márquez, and Luis Mengual-Galán, "Adaptive quadtree splitting parallelization (AQSP) algorithm for the VVC standard," *The Journal of Supercomputing*, pp. 1–16, 2024.

[12] ITU-T, "Recommendation P.910, Subjective Video Quality Assessment Methods for Multimedia Applications," Tech. Rep., International Telecommunication Union, 1999.

[13] JVET, "VVC Test Model - Version 20.0," `https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_`

Tabla II: Results of the final algorithm including the NB model following the CTCs guidelines.

Class	Sequence	BD-rate (%)	Time reduction (%)
A1	Drums100	0,9812	12,7704
	Tango2	1,7156	41,9871
	Campfire	0,8235	7,0090
	ToddlerFountain2	0,0458	1,8989
A2	CatRobot	2,0687	19,8487
	TrafficFlow	2,3111	17,9235
	RollerCoaster2	3,9787	17,8234
	DaylightRoad2	2,8891	43,6475
B	Kimono	0,6643	36,4227
	ParkScene	0,6915	38,7248
	BasketballDrive	0,9540	13,3003
	Cactus	0,7605	20,1167
	BQTerrace	1,1319	15,7520
C	BasketballDrill	0,2673	5,5769
	BQMall	0,3282	6,8981
	PartyScene	0,1889	4,5926
	RaceHorsesC	−0,0032	33,8261
	Average	1,1618	19,8893

`VTM/-/tree/VTM-20.0`.

[14] U. M. Fayyad and K. B. Irani, "Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning," in *Proceedings of the International Joint Conference on Uncertainty in AI, Chambery, France*, Aug 1993.

[15] I. Guyon and A. Elisseeff, "An Introduction to Variable and Feature Selection," *Journal Machine Learning Research*, vol. 3, pp. 1157–1182, Mar 2003.

[16] Paul Jaccard, "Étude comparative de la distribution florale dans une portion des Alpes et des Jura," *Bull Soc Vaudoise Sci Nat*, vol. 37, pp. 547–579, 1901.

[17] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Transactions on circuits and systems for video technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[18] Gisle Bjøntegaard, "Improvements of the BD-PSNR Model," Tech. Rep. VCEG-AI11, ITU-T SG16 Q6, 2008.

[19] Damián Ruiz, Gerardo Fernández-Escribano, José Luis Martínez, and Pedro Cuenca, "Fast intra mode decision algorithm based on texture orientation detection in HEVC1," *Signal Processing: Image Communication*, vol. 44, pp. 12–28, 2016.

[20] Guilherme Correa, Pedro A. Assuncao, Luciano Volcan Agostini, and Luis A. da Silva Cruz, "Fast HEVC encoding decisions using data mining," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 4, pp. 660–673, 2015.

[21] Behnam Heidari and Mohammadreza Ramezanpour, "Reduction of intra-coding time for HEVC based on temporary direction map," *Journal of Real-Time Image Processing*, vol. 17, no. 3, pp. 567–579, 2020.

[22] Wei-Jung Chien, Li Zhang, Martin Winken, Xiang Li, Ru-Ling Liao, Han Gao, Chih-Wei Hsu, Hongbin Liu, and Chun-Chi Chen, "Motion vector coding and block merging in the versatile video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3848–3861, 2021.

[23] W. Yeo and B.-G. Kim, "CNN-based fast split mode decision algorithm for versatile video coding (VVC) inter prediction," *J. Multimed. Inf. Syst.*, vol. 8, pp. 147–158, 2021.

[24] A. Tissier, W. Hamidouche, J. Vanne, and D. Menard, "Machine Learning Based Efficient QT-MTT Partitioning for VVC Inter Coding," in *2022 IEEE International Conference on Image Processing (ICIP)*, 2022, pp. 1401–1405.

[25] Alexandre Tissier, Wassim Hamidouche, Souhaiel Belhadj Dit Mdalsi, Jarno Vanne, Franck Galpin, and Daniel Menard, "Machine Learning Based Efficient QT-MTT Partitioning Scheme for VVC Intra Encoders," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 33, no. 8, pp. 4279–4293, 2023.

[26] Yingbiao Yao, Jiaojiao Wang, Chenjie Du, Jinghui Zhu, and Xin Xu, "A support vector machine based fast planar prediction mode decision algorithm for versatile video coding," *Multimedia Tools and Applications*, vol. 81, no. 12, pp. 17205–17222, 2022.

[27] Qiuwen Zhang, Yihan Wang, Lixun Huang, Bin Jiang, and Xiao Wang, "Fast CU partition decision for H.266/VVC based on the improved DAG-SVM classifier model," *Multimedia Systems*, vol. 27, no. 1, pp. 14–27, 2021.

[28] Gosala Kulupana, Saverio Blasi, et al., "Fast versatile video coding using specialised decision trees," in *2021 Picture Coding Symposium (PCS)*. IEEE, 2021, pp. 1–5.

[29] C.-M. Lien, M.-J. Chen, C.-M. Yang, R.-H. Hong, Y.-H. Lin, and C.-H. Yeh, "Multi-type tree partition decision for H.266/VVC inter coding based on Random Forest classifiers," in *IET International Conference on Engineering Technologies and Applications (ICETA 2023)*, 2023, vol. 2023, pp. 97–98.

[30] Joint Video Experts Team, "VVC Software VTM-20.0," `https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/-/tree/VTM-20.0?ref_type=tags`, 2024, [Online; accessed 22-July-2024].

[31] Pedro Domingos and Michael Pazzani, "On the optimality of the simple Bayesian classifier under zero-one loss," *Machine learning*, vol. 29, pp. 103–130, 1997.

[32] Jason D Rennie, Lawrence Shih, Jaime Teevan, and David R Karger, "Tackling the poor assumptions of naive bayes text classifiers," in *Proceedings of the 20th international conference on machine learning (ICML-03)*, 2003, pp. 616–623.

[33] Geoffrey I Webb, Eamonn Keogh, and Risto Miikkulainen, "Naïve Bayes," *Encyclopedia of machine learning*, vol. 15, no. 1, pp. 713–714, 2010.

[34] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[35] Nello Cristianini, *An Introduction to Support Vector Machines and other kernel-based learning methods*, Cambridge University Press, 2000.

[36] Rikard Vinge and Tomas McKelvey, "Understanding support vector machines with polynomial kernels," in *2019 27th European signal processing conference (EUSIPCO)*. IEEE, 2019, pp. 1–5.

[37] André E Lazzaretti and David MJ Tax, "An adaptive radial basis function kernel for support vector data description," in *Similarity-Based Pattern Recognition: Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3*. Springer, 2015, pp. 103–116.

[38] Shubhkirti Sharma and Vijay Kumar, "A comprehensive review on multi-objective optimization techniques: Past, present and future," *Archives of Computational Methods in Engineering*, vol. 29, no. 7, pp. 5605–5633, 2022.

Modelo para la construcción de sistemas continuos de observación de la Tierra

Catherine A. Torres-Charles[1], Dante D. Sánchez-Gallegos[1], Diana Carrizales-Espinoza[1], Jesús Carretero[1], y J. L. González-Compeán[2]

Resumen— **Los sistemas de observación de la Tierra (SOT) requieren el procesamiento de grandes volúmenes de datos espacio-temporales para abordar problemas de cambio climático, urbanización y degradación ambiental. El continuum computacional ofrece un paradigma prometedor para soportar estas tareas, al permitir el procesamiento de datos cerca de las fuentes y la distribución masiva a través de la nube. Sin embargo, diseñar y gestionar sistemas distribuidos en este entorno plantea retos críticos de coordinación y orquestación de datos. Este artículo presenta un marco serverless basado en un modelo orbital para la construcción de sistemas continuos de observación de la Tierra. Se organizan los componentes de un SOT como un conjunto de órbitas que representan entornos computacionales —borde (edge), niebla (fog) y nube (cloud)— dispuestos alrededor de un núcleo de fuentes de datos. Cada órbita alberga satélites que despliegan servicios mediante contenedores y funciones. La comunicación entre satélites se gestiona mediante contenedores de datos que establecen canales de comunicación dinámicos en función de los recursos disponibles. Se evalúa en un estudio de caso sobre el análisis de cambios en los recursos hídricos de lagos en España, procesando imágenes de Landsat 8/9 entre 2017 y 2025. Los resultados demuestran que este trabajo supera en un 20 % a una solución del estado del arte basada en Parsl y Globus Compute.**

Palabras clave— **Hoja de estilo, LaTeX, Jornadas Sarteco.**

I. Introducción

El continuum computacional ha emergido como una alternativa complementaria a los entornos centralizados tradicionales. Este paradigma facilita el procesamiento de datos cerca de las fuentes, reduciendo tanto la latencia como el consumo de ancho de banda. Además, permite la creación de sistemas descentralizados donde múltiples entornos —edge, fog y cloud— colaboran para procesar y compartir datos de forma continua[1][2]. Los Sistemas de Observación de la Tierra (SOT) son un caso de uso paradigmático de este modelo. Un SOT típico abarca múltiples aplicaciones interconectadas para la adquisición, procesamiento y análisis de datos satelitales [3]. Por ejemplo, un SOT orientado al procesamiento de imágenes comprende tres fases principales: (i) extracción, donde estaciones terrestres capturan imágenes crudas para generar productos de Nivel 1 en el edge; (ii) preparación, donde dichos productos se refinan en infraestructuras de fog o HPC para obtener productos de Nivel 2 y 3; y (iii) procesamiento, donde los productos resultantes se explotan para estudios de cambio climático, tendencias de urbanización y degradación del suelo [4][5].

Gestionar un SOT desplegado en el continuum computacional es complejo. Requiere coordinar múltiples aplicaciones que se ejecutan en infraestructuras distribuidas [6], configurar dependencias de software, gestionar flujos de datos con sincronización de E/S correcta y afrontar fluctuaciones de carga. Sistemas de flujo de trabajo como Pegasus [7] o Nextflow [8] abordan algunas de estas necesidades, pero están diseñados principalmente para entornos HPC o multi-nube, limitando su aplicabilidad en escenarios heterogéneos del continuum.

La computación serverless ha surgido como alternativa para desplegar sistemas en el continuum computacional [9][10]. Soluciones como Globus Compute [11][12] y OpenFaaS [13] demuestran que el modelo Function-as-a-Service (FaaS) puede soportar el despliegue y ejecución en entornos diversos. Sin embargo, dos retos críticos permanecen: (i) la transferencia eficiente de grandes volúmenes de datos entre dispositivos y puntos de procesamiento, y (ii) el mantenimiento de la escalabilidad y el rendimiento durante la coordinación de flujos de datos entre infraestructuras heterogéneas.

Este artículo presenta un método serverless para la construcción de SOT continuos basado en un modelo orbital. Las contribuciones principales son:

- Un modelo orbital para gestionar SOT continuos desplegados en entornos computacionales heterogéneos.
- Un marco serverless que automatiza el despliegue mediante paradigmas CaaS y FaaS, con mecanismos de autoescalado implícitos.
- Una abstracción de comunicación unificada basada en contenedores de datos que gestiona flujos de datos distribuidos en el continuum.
- Un caso de estudio real sobre análisis de recursos hídricos que demuestra la viabilidad y eficiencia del sistema.

El resto del artículo se organiza de la siguiente manera: en la Sección 2 se mencionan los antecedentes y el trabajo relacionado. La Sección 3 introduce el diseño de los elementos principales. La Sección 4 presenta la arquitectura y la implementación del sistema. La Sección 5 describe la evaluación experimental. Finalmente, la Sección 6 resume las conclusiones.

[1]Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: {cattorre, jcarrete}@inf.uc3m.es, {dantsanc, dcarriza}@pa.uc3m.es

[2]CINVESTAV Tamaulipas, e-mail: joseluis.gonzalez@cinvestav.mx

II. Antecedentes y trabajo relacionado

A. Serverless para el continuum computacional

La computación serverless ofrece un marco sencillo para desplegar y gestionar servicios en infraestructuras remotas. Plataformas como AWS Lambda [14], Google Cloud Functions [15] y Azure Functions [16] son ejemplos consolidados en el ámbito de la nube pública. Para extender este paradigma a entornos on-premise, han surgido soluciones como Globus Compute [12], UniFaaS [17], OpenFaaS [13] y OpenWhisk [18]. Globus Compute permite convertir computadores en puntos de ejecución serverless para FaaS. En combinación con Parsl [19], proporciona un marco para ejecutar flujos de trabajo distribuidos. Sin embargo, carece de mecanismos integrados para la transferencia de grandes volúmenes de datos, dependiendo de herramientas externas como Globus Transfer. OpenFaaS soporta el despliegue de microservicios y funciones en Kubernetes, con escalado automático basado en demanda, pero sin orquestación de flujos de trabajo nativa. OpenWhisk permite encadenar funciones mediante secuencias para construir sistemas en el continuum, aunque su despliegue resulta complejo por sus múltiples dependencias. Este trabajo comparte similitudes con Globus Compute en cuanto al soporte de servicios serverless en múltiples infraestructuras. Sin embargo, el trabajo actual se diferencia al componer múltiples funciones en un sistema unificado con flujos de datos automáticos, soportando patrones flexibles como canalizaciones y flujos de trabajo.

B. Serverless en sistemas de observación de la tierra

La observación de la Tierra implica el procesamiento de grandes volúmenes de datos espaciotemporales [3]. Herramientas como Google Earth Engine [20] han popularizado el uso de plataformas de virtualización en la nube para flujos de trabajo de SOT. Sin embargo, la centralización en la nube puede generar cuellos de botella y problemas de latencia [21]. Propuestas como Kaiser et al. [22] y Ueckermann et al. [10] abordan la transformación de aplicaciones tradicionales de observación de la Tierra en despliegues serverless. Pfandzelter et al. [23] introdujeron Komet, una plataforma serverless para servicios de borde en órbita terrestre baja. Sin embargo, muchas de estas soluciones están diseñadas para un único proveedor de nube, con el consiguiente riesgo de dependencia del proveedor. Este trabajo mitiga estos problemas desplegando soluciones serverless en múltiples infraestructuras.

La tabla I muestra un resumen de las características principales de los sistemas y marcos de trabajo más relevantes de la literatura, comparándolos con el trabajo actual. Las características principales reflejan los requisitos fundamentales que se deben cumplir para que un sistema de observación de la Tierra sea desplegado en el continuum computacional; estos son un soporte multi-entorno (edge-fog-cloud), orquestación de flujos, gestión integrada de datos, modelos de despliegue, autoescalado, independencia de proveedor y orientación al dominio SOT.

Como ya se ha mencionado anteriormente, el trabajo actual integra todas estas capacidades en un único marco coherente, donde el modelo proporciona una abstracción jerárquica que organiza los recursos computacionales en función de su proximidad a las fuentes de datos, permitiendo despliegues flexibles en entornos edge-fog-cloud.

III. Diseño de elementos principales

Este diseño permite a las organizaciones construir un Sistema continuo de Observación de la Tierra (SOT continuo) acoplando múltiples etapas distribuidas —funciones y aplicaciones— a través de un modelo orbital. Este modelo ofrece una abstracción de alto nivel sobre cómo se estructura y opera un SOT en el continuum.

A. Modelo para la abstracción de SOT

El modelo se inspira en el patrón de arquitectura de cebolla (*onion architecture*), donde las capas se organizan alrededor de un dominio central —el núcleo del sistema. Cada entorno computacional lógico (edge, fog, cloud) se representa como una órbita (O) que rodea el núcleo, compuesto por las fuentes de datos. Las órbitas internas corresponden a entornos de edge e IoT, físicamente más próximos a las fuentes de datos, con baja latencia pero con recursos computacionales limitados. Las órbitas intermedias representan nodos de fog o centros HPC, con un equilibrio entre potencia de procesamiento y latencia. Las órbitas externas corresponden a infraestructuras de nube pública y privada, con mayor capacidad computacional a costa de mayor latencia.

La Figura 1 ilustra el conjunto de órbitas que contienen satélites, que son agrupados de acuerdo a sus características en términos de poder computacional y proximidad geográfica. Cada órbita se maneja como una constelación y es operada por una única organización dentro de una ubicación compartida.

Formalmente, se define el conjunto de fuentes de datos como $\mathbf{D} = \{d_1, d_2, ..., d_m\}$, con $m \geq 1$, y el conjunto de órbitas como $\mathbf{O} = \{O_1, O_2, ..., O_n\}$, con $n \geq 2$. En cada órbita, los recursos computacionales se abstraen como satélites ($\mathbf{S}$) que actúan como proveedores de servicios de almacenamiento o procesamiento.

El universo de entornos disponibles se define como:

$$U_{environments} = \{\mathbf{O}, \mathbf{D}\}, \tag{1}$$

Donde $\mathbf{O}$ es un conjunto de órbitas y $\mathbf{D}$ es un conjunto de fuentes de datos. Cada órbita $\mathbf{O}_x$ se compone de una constelación $\mathbf{C}$ de satélites $\mathbf{S}_i$ y un agente de gestión $\mathbf{G}$ que los coordina mediante un modelo maestro-trabajador:

$$\mathbf{O}_x = \{\mathbf{C}, \mathbf{G}\} \mid \mathbf{C} = \{\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_n\},\ n \geq 1, \tag{2}$$

Cada satélite $\mathbf{S}_x$ se caracteriza por sus atributos computacionales: memoria (mem), almacenamiento (stg) y número de CPUs ($\#cpu$):

$$\mathbf{S}_x = \{mem, stg, \#cpu\}. \tag{3}$$

Tabla I: Comparativa de sistemas y marcos de trabajo relacionados

Sistema / Marco	Multi-entorno	Orquestación	Transferencia de datos integrada	FaaS / CaaS	Autoescalado	Independencia de proveedor	Orientado a SOT
Pegasus [7]	×	✓	—	×	—	—	×
Nextflow [8]	—	✓	—	×	—	—	×
Parsl [19]	—	✓	—	—	—	—	×
DagOnStar / OpenWolf [24][25]	—	✓	—	—	×	—	—
Globus Compute [11]	—	—	×	✓	—	—	—
UniFaaS [17]	—	✓	—	✓	—	—	×
OpenFaaS [13]	—	×	×	✓	✓	—	×
OpenWhisk [18]	—	—	×	✓	—	—	×
Google Earth Engine [15]	×	—	—	×	—	×	✓
Kaiser et al. [22]	×	—	×	—	×	×	—
PODPAC [10]	×	—	—	—	×	—	—
Komet [23]	—	×	×	—	×	×	✓

— parcialmente × sin soporte ✓soporte completo

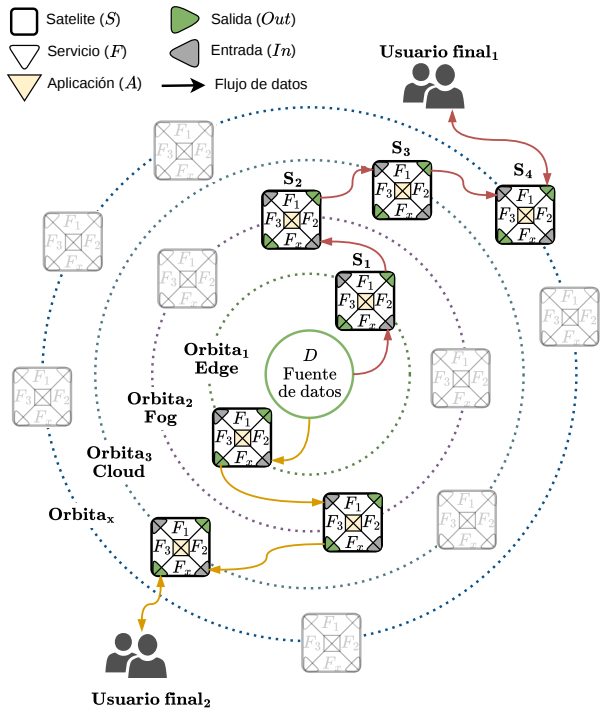


Fig. 1: Ejemplo conceptual del modelo de órbitas para SOT.

Dentro de un satélite, los usuarios pueden desplegar uno o más servicios. Un servicio **F** representa una agrupación lógica de aplicaciones **A** que realizan tareas específicas de gestión de datos, con interfaces de entrada In y salida Out para el intercambio de datos entre servicios:

$$\mathbf{F}_x = \{In, \mathbf{A}, Out\}, \mathbf{A} = \{a_1, a_2\}, n \geq 1 \quad (4)$$

Un SOT continuo se modela como un grafo acíclico dirigido (DAG), donde cada nodo corresponde a un servicio desplegado en un satélite y cada arista dirigida denota una dependencia de datos:

$$SOT = \{\mathbf{F}, \mathbf{E}\} \quad (5)$$

Donde cada enlace (**E**) es un conjunto de interconexiones entre dichos servicios. Cada enlace se define:

$$\mathbf{E}_x = (F_i, F_j) \mid F_i, F_j \in \mathbf{F} \wedge \mathbf{i} \neq \mathbf{j} \quad (6)$$

Un flujo de datos típico puede seguir el patrón:

$$\text{flujodedatos}_1 = F_{\text{edge}} \rightarrow F_{\text{fog}} \rightarrow F_{\text{hpc}} \rightarrow F_{\text{cloud}} \quad (7)$$

Donde F_{edge} es un servicio ejecutándose en la órbita $edge$, F_{fog} está en la órbita fog, F_{hpc} en un centro de HPC, y F_{cloud} está en la nube en una máquina virtual.

B. Esquema de despliegue

En este trabajo se implementan dos modelos serverless. El modelo *FaaS* es adecuado para aplicaciones de corta duración que se ejecutan de forma esporádica en dispositivos con capacidad computacional limitada, como los nodos edge. El agente de gestión utiliza una política de menor carga para asignar la función al satélite con menor utilización de recursos:

$$U(\mathbf{S}_i) = p_1 \times \frac{u_mem(\mathbf{S}_i)}{mem(\mathbf{S}_i)} + p_2 \times \frac{u_cpu(\mathbf{S}_i)}{cpu(\mathbf{S}_i)}, \quad (8)$$

Donde $u_mem(\mathbf{S}_i)$ y $u_cpu(\mathbf{S}_i)$ representan la memoria y la CPU utilizadas en el satélite $\mathbf{S}_i$, respectivamente, y p_1 y p_2 son pesos configurables por la organización. El satélite seleccionado es:

$$\mathbf{S}_x = \arg\min_{\mathbf{S}_i \in \mathbf{C}} U(\mathbf{S}_i). \quad (9)$$

Si la función a utilizar tiene un requisito de hardware, el agente filtrará los satélites para obtener el indicado antes de aplicar la función solicitada.

El modelo *Container-as-a-service* (*CaaS*) sigue un proceso similar, pero el desarrollador proporciona una imagen de contenedor que incluye la lógica de la aplicación y sus dependencias de ejecución.

El set de contenedores disponibles se denomina:

$$\mathbf{SC} = \{sc_1, sc_2, \ldots, sc_n\}, \quad (10)$$

donde cada contenedor $sc_i \in \mathbf{SC}$ es representado como $\{ID, R, \mathbf{C}\}$, siendo ID el identificador único, R los requerimientos de hardware/software y **C** la constelación a la que ha sido asociado.

Los contenedores se registran en un registry privado alojado por el agente de gestión de esa constelación, haciéndolos accesibles a todos los satélites dentro de esa órbita. Estos contenedores se mantienen en ejecución para minimizar la sobrecarga y permitir el procesamiento de datos bajo demanda.

Algorithm 1 Proceso de inicialización del almacenamiento de datos.

Entrada: S: satélite que produce los datos.
Salida: *canalesDisponibles*: conjunto de estructuras de datos que contienen las métricas de los canales de comunicación entre S y sus dependientes.
dependienteS ← obtenerSatDependientes()
$\mathcal{CH}$ ← obtenerCanalesDisponibles()
canalesDisponibles ← $\emptyset$
para cada $ds \in$ *dependienteS* **hacer**
  para cada $ch \in \mathcal{CH}$ **hacer**
    metricas ← medir(S, ds, ch)
    {Devuelve una estructura con métricas de ancho de banda y concurrencia máxima.}
    si *metricas.bw* > 0 **entonces**
      canalesDisponibles.add(S, ds, metricas)
    fin si
  fin para
fin para
devolver *canalesDisponibles*

C. Gestión de datos mediante contenedores de datos

Un aspecto clave de los sistemas en el continuum computacional es el movimiento de datos entre infraestructuras heterogéneas. En el modelo de órbitas, esta funcionalidad es gestionada por entidades denominadas contenedores de datos (CD). Durante la ejecución, un CD realiza dos tareas principales: la orquestación de datos (transferencia de objetos entre servicios) y el almacenamiento de datos (persistencia a largo plazo).

Esta implementación soporta cuatro tipos de canales de comunicación: **FSChannel** (operaciones de lectura/escritura sobre el sistema de ficheros); **MemChannel** (mecanismos de memoria compartida); **SockChannel** (transmisión de datos sobre sockets de red TCP); y **DynoChannel** (sistema de distribución de área amplia DynoStore). Dado un conjunto de canales disponibles $\mathbf{CH} = \{ch_1, ch_2, \ldots, ch_n\}$. La selección dinámica del canal se realiza maximizando el ancho de banda disponible:

$$ch_x = \arg \max_{ch \in \mathbf{CH}} (\text{bw(ch)}), \tag{11}$$

Donde $bw(\cdot)$ es una función que estima el ancho de banda de un canal dado.

Durante la inicialización del sistema, si no se dispone de información previa sobre los canales disponibles, los contenedores de datos (CD) envían mensajes de sondeo para medir el ancho de banda y el nivel de concurrencia. El Algoritmo 1 detalla el proceso de inicialización que utiliza un contenedor de datos para determinar las condiciones iniciales de los canales de comunicación entre un servicio y sus servicios dependientes.

Para acelerar la transferencia de datos, los contenedores de datos pueden establecer múltiples canales de transferencia concurrentes. El número de operaciones concurrentes se determina durante la inicialización. La distribución de objetos entre canales utiliza el algoritmo de los two-choices, que asigna cada objeto al canal con menor utilización de entre dos seleccionados aleatoriamente.

Dado un conjunto de n objetos ($\mathbf{Ob} = \{ob_1, \ldots, ob_n\}$) y un conjunto de m canales ($\mathbf{CH} = \{ch_1, \ldots, ch_m\}$), cada objeto se asigna a un canal mediante:

$$\begin{aligned} l_1 &\sim \mathbf{U}(0, m), \\ l_2 &\sim \mathbf{U}(0, m), \\ ch_x &= \arg \min_{ch \in \{l_1, l_2\}} \text{uti(ch)}, \end{aligned} \tag{12}$$

donde l_1 y l_2 son dos canales seleccionados aleatoriamente, y uti(ch) representa la carga actual del canal ch.

Es decir, se elige el canal con menor carga entre dos seleccionados al azar, logrando un balance de carga eficiente a un costo computacional muy bajo.

D. Autoescalado Basado en Patrones Paralelos

Esta implementación incluye un mecanismo de autoescalado que ajusta dinámicamente el número de trabajadores activos en respuesta a variaciones en la carga de trabajo. El gestor de replicación calcula la tasa de procesamiento de cada etapa e identifica aquella con menor rendimiento —un posible cuello de botella— (Btl) mediante:

$$Btl = \text{MINRendimientoPos(Apps_Rendimientos)}, \tag{13}$$

donde *Apps_Rendimientos* es el conjunto que contiene la tasa de procesamiento de todas las etapas. La función MINRendimientoPos($\cdot$) devuelve la posición de la etapa con menor rendimiento.

Una vez identificado el cuello de botella (Btl), se añade un nuevo trabajador a la etapa correspondiente con el objetivo de mejorar su capacidad de procesamiento. No obstante, el escalado no siempre implica una mejora del rendimiento, ya que depende de los recursos físicos disponibles (por ejemplo, el número de núcleos de CPU) y de la posible contención con otros servicios.

Para mitigar este problema, el sistema limita el número de trabajadores al número de núcleos físicos del nodo y monitoriza continuamente el rendimiento, eliminando trabajadores si su incorporación degrada el rendimiento global.

IV. Arquitectura e Implementación

La arquitectura se organiza en tres capas principales: la capa de control (front-end), la capa de nodos (endpoints) y la capa de datos (Ver Figura 2). La capa de front-end proporciona servicios de control de acceso para la autenticación y autorización de usuarios. Expone tres interfaces globales: (i) la interfaz de diseño, que permite a los desarrolladores crear instancias de SOT continuo y gestionar los satélites donde se despliegan aplicaciones y contenedores de datos; (ii) la interfaz de acceso al SOT, que gestiona

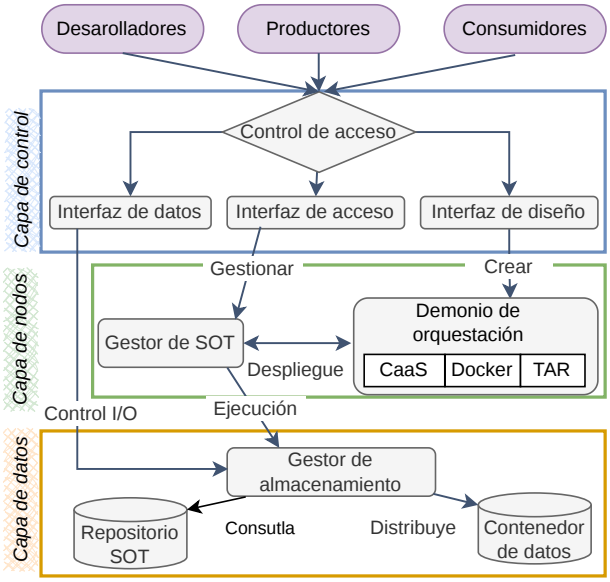


Fig. 2: Arquitectura del sistema

el acceso de usuarios a instancias definidas en esta implementación mediante un modelo de permisos similar al de los sistemas de ficheros Unix (lectura, escritura/modificación y ejecución); y (iii) la interfaz de datos, que controla el acceso a los datos de entrada y salida de un SOT, definiendo roles de usuario como consumidor o productor. La capa de nodos gestiona la infraestructura física y virtual donde se despliegan los SOT. Incluye el demonio, responsable de orquestar el despliegue de instancias de SOT, y un gestor de SOT que implementa el esquema de paralelismo. La capa de datos gestiona el almacenamiento y la transferencia de datos utilizando contenedores de datos, con un gestor de almacenamiento que supervisa estos contenedores e implementa un mecanismo de balanceo de carga para distribuir los datos en el almacén global.

El prototipo utiliza C++ para la capa de nodos (*endpoints*), incluyendo el esquema de paralelismo y la lógica de despliegue. Las interfaces de diseño, acceso al SOT y datos se implementan en Python 3.10. Los contenedores de datos también se implementan en Python 3.10. Para CaaS, los desarrolladores pueden proporcionar imágenes de contenedor como Dockerfile, paquete TAR o URI que referencia un registro público como Docker Hub.

Respecto a la seguridad de datos, se implementan tres mecanismos: integridad (verificación mediante SHA3-256), confidencialidad (cifrado mediante los algoritmos LEA y AES), y control de acceso (autenticación mediante tokens OAuth gestionados con PyJWT). El repositorio de SOT se implementa mediante MongoDB, donde las configuraciones se almacenan como documentos JSON.

V. Evaluación experimental

Se presenta la evaluación de este trabajo en un caso de estudio para el procesamiento de imágenes satelitales orientado al análisis de cambios en recursos hídricos: lagos, lagunas y embalses. El sistema procesa imágenes Landsat 8/9 y genera índices de agua de diferencia normalizada (NDWI) que destacan masas de agua usando bandas infrarrojas cercanas y verdes.

Tabla II: Características de la infraestructura utilizada para la evaluación.

Ubicación	Hostname	Almacen (TB)	Num. núcleos	RAM GB)
TACC, Texas	Host1	0.8	48	69
U. Chicago, Illinois	Host2	0.8	48	275
U. Chicago, Illinois	Host3	1.8	40	275
U. Chicago, Illinois	Stg	0.8	96	275
AWS N. Virginia	Geoportal	0.1	8	32

El sistema se compone de las siguientes etapas:

- **Adquisición de datos**: desempaquetado de imágenes satelitales descargadas del geoportal EarthExplorer de USGS.
- **Correcciones radiométricas**: mitigación de anomalías de adquisición, como el ruido del sensor, los efectos atmosféricos o las sombras topográficas.
- **Recorte**: extracción de la región de interés (bounding box del área de estudio) de cada banda espectral.
- **Derivados**: generación de índices NDWI con diferentes combinaciones de bandas espectrales.
- **Análisis**: cálculo del porcentaje de agua identificado por los índices y regresiones lineales para revelar tendencias.

Siguiendo el modelo orbital, las etapas de adquisición, corrección y recorte se despliegan en servidores de edge. Las etapas de derivados y análisis se despliegan en la capa fog, mientras que un geoportal se aloja en la nube para permitir el acceso externo a los resultados. Como línea base de comparación, se implementó el mismo sistema utilizando una combinación de Parsl[19] y Globus Compute [12], complementada con un servidor FTP multiplexado para el intercambio de datos entre funciones.

A. Entorno de experimentación

Se utilizaron 100 imágenes Landsat 8/9 (98 GB en total) de la Laguna Santa Olalla en el Parque Nacional de Doñana, España. La Tabla 1 describe las características de la infraestructura utilizada.

El objetivo era replicar un escenario realista en el que diferentes máquinas, tanto consumidoras como productoras, transfieren y procesan datos a través de dos entornos. El entorno de Chameleon Testbed, TACC y U. Chicago emula un escenario interorganizativo, donde múltiples instituciones colaboran en un sistema federado para compartir datos y recursos. Adicionalmente, se desplegó una máquina virtual EC2 en Amazon AWS para alojar el geoportal.

B. Resultados de rendimiento

En esta sección se presentan los resultados obtenidos de la ejecución y el procesamiento de imágenes satelitales.

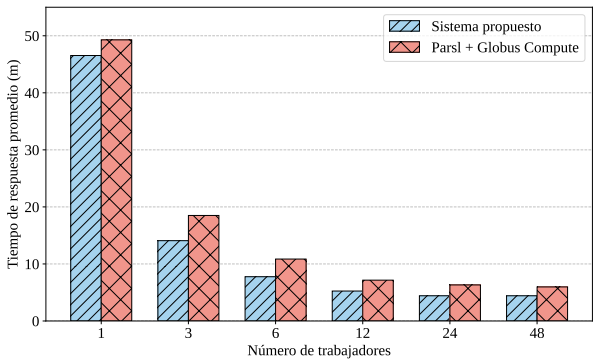


Fig. 3: Tiempo de respuesta al procesar imágenes de satélite para realizar análisis de cambios de agua en el lago Doñana, España.

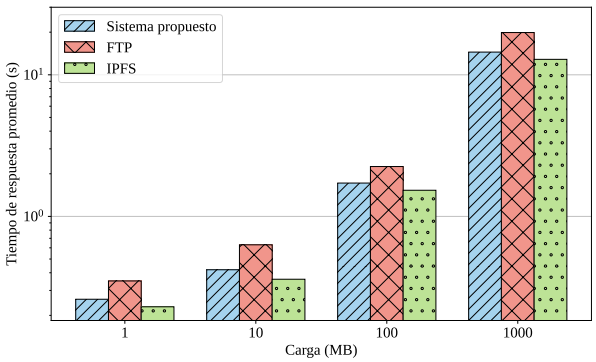


Fig. 4: Rendimiento de la transferencia de datos al transferir diferentes tamaños de carga de trabajo entre nodos en Chicago y Austin.

Procesamos imágenes de Landsat 8/9 de la laguna de Santa Olalla de Doñana, en el sur de España, utilizando la infraestructura Chameleon. El nodo *Host1* gestionaba la descarga y descompresión de datos. Las etapas de recorte y derivados se ejecutaron en *Host2*, y la etapa de análisis se ejecutó en *Host3*. A lo largo de este flujo de trabajo, el nodo *Stg* proporcionó servicios de almacenamiento utilizando un servidor FTP para el intercambio de datos.

La Figura 3 muestra el tiempo total de respuesta (eje vertical) para procesar imágenes satelitales en distintas infraestructuras, bajo diferentes configuraciones del número de trabajadores (eje horizontal), evaluando 100 imágenes con un total de 98-GB procesados en Chameleon testbed, utilizando la solución propuesta y una combinación de Parsl con Globus compute.

Los resultados demuestran que el sistema propuesto escala eficazmente durante el procesamiento de datos. Al incrementar el número máximo de trabajadores, se reduce el tiempo total de respuesta en este entorno. Por ejemplo, el sistema propuesto procesa todo el dataset en 46,5 minutos con un solo trabajador. Al pasar a 48 trabajadores, se logra procesar en solo 4,4 minutos. Esto supone una aceleración de 10x. Esta reducción equivale a una mejora de rendimiento de aproximadamente el 90 % respecto a la configuración secuencial base.

Comparando el sistema propuesto con la solución de referencia basada en Parsl + Globus Compute, se observa una mejora media del 23,45 % en el tiempo de respuesta en todos los escenarios evaluados. Cabe destacar que la eficiencia del sistema propuesto se hace más evidente al escalar la carga, alcanzando picos de mejora de hasta un 30 % (en configuraciones de 24 trabajadores), lo que demuestra su superioridad para mitigar los cuellos de botella de E/S en procesos altamente paralelos. Esta mejora se atribuye principalmente a dos factores. En primer lugar, Parsl y Globus Compute dependen de primitivas de multiprocesamiento de Python, que introducen sobrecarga en la gestión de tareas paralelas; en cambio, la implementación desarrollada utiliza primitivas de paralelismo en C++, lo que permite una sincronización más eficiente. En segundo lugar, el modelo por capas organiza los componentes del SOT en diferentes zonas, mejorando la localidad de datos al situar los contenedores cerca de los productores.

En cuanto a los mecanismos de transferencia de datos, la Figura 4 compara el tiempo de respuesta de distintas cargas de trabajo utilizando la solución propuesta, FTP y el sistema descentralizado InterPlanetary (IPFS) [26], transmitiendo información entre nodos ubicados en Chicago y Austin.

Como se observa, el sistema desarrollado es, en promedio, un 27 % más rápido que FTP. Esta ventaja radica en que nuestra solución minimiza los saltos de red. En contraste, FTP requiere subir los datos a un servidor intermedio en Chicago y luego descargarlos en el destino en Austin; esta fase de descarga representa, en promedio, el 82 % del tiempo total de transferencia.

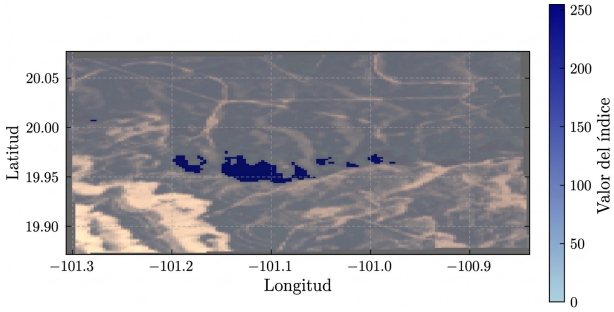
Al compararlo con IPFS, nuestro sistema registra una penalización de rendimiento de aproximadamente un 11 % en promedio. Este costo computacional es consecuencia de las operaciones de seguridad adicionales necesarias para garantizar la confidencialidad e integridad de los datos, a diferencia de IPFS, que es público y solo provee integridad mediante hashing. Notablemente, la arquitectura modular de nuestro trabajo permite a los desarrolladores integrar de forma nativa herramientas de terceros, como el propio IPFS, para extender sus capacidades.

C. Análisis de los productos ambientales generados

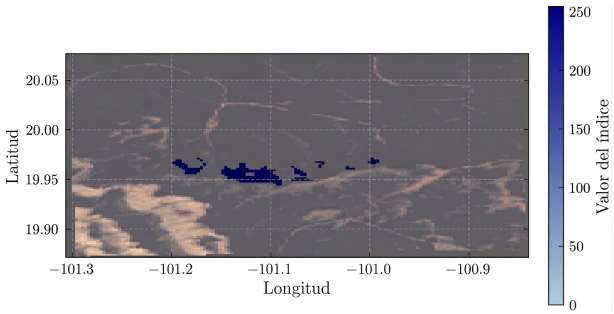
En las siguientes figuras se muestran los productos generados con el sistema evaluado en este estudio. En la Figura 5 se observa que la Laguna Santa Olalla (Huelva, España) muestra cambios significativos en la cobertura de agua (resaltado en azul oscuro) entre 2017 y 2025. Se observa una notable reducción en la superficie del lago. Este descenso es coherente con los efectos de una sequía prolongada.

La figura 6 muestra los resultados producidos por la etapa de *análisis*, incluye gráficos de regresión lineal y de puntos de inflexión. El eje vertical muestra el porcentaje estimado de superficie acuática, mientras que el eje horizontal abarca el periodo comprendido entre 2017 y 2025.

Los puntos azules representan los valores reales obtenidos a partir de los índices de agua, y la línea roja indica una regresión lineal utilizada para reve-

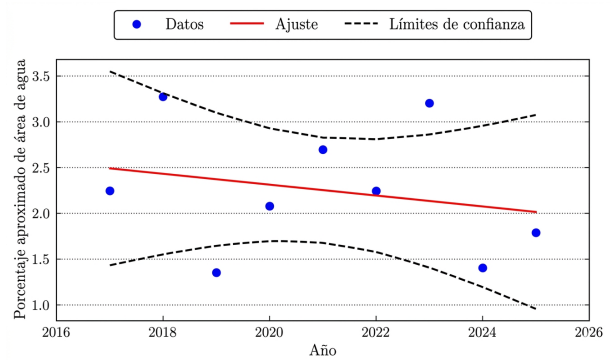


(a) 2017.

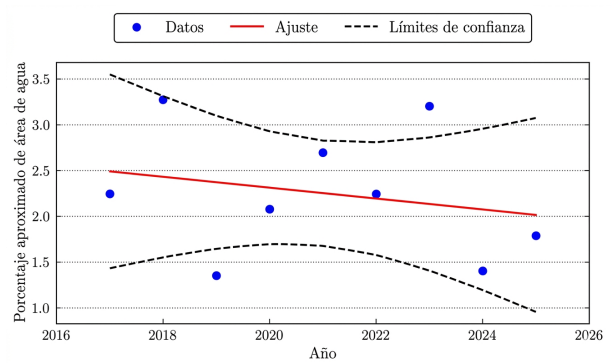


(b) 2025.

Fig. 5: Productos generados a partir del procesamiento de imágenes de satélite de la Laguna de Santa Olalla de Doñana (Huelva, España) con el sistema propuesto.



(a) Regresión lineal.



(b) Regresión de punto de salto.

Fig. 6: Porcentaje estimado de recurso hídrico respecto a la superficie total de imágenes LandSat8/9 para la Laguna de Santa Olalla de Doñana.

lar tendencias. Aunque algunos años –como 2021 y 2023– muestran una recuperación parcial, se observa una fuerte caída en 2024, con un modesto repunte a principios de 2025.

Estos resultados demuestran la utilidad del SOT diseñado con el sistema propuesto para monitorizar tendencias de escasez hídrica, un problema crítico en múltiples regiones del mundo.

VI. Conclusiones

Este artículo ha presentado un marco para el diseño de sistemas de observación de la tierra (SOT) que sigue los principios de la computación serverless. Este automatiza el despliegue, el escalado, la monitorización y la ejecución de funciones y aplicaciones en infraestructuras diversas. Su diseño orientado al continuum y el modelo orbital soportan una orquestación de servicios flexible en entornos de edge, fog y cloud.

Los contenedores de datos superan en rendimiento a soluciones centralizadas como servidores FTP multiplexados. Comparados con sistemas P2P como IPFS, los contenedores de datos incurren en cierta sobrecarga, pero ofrecen mayor disponibilidad gracias a la replicación de datos en múltiples contenedores. La evaluación experimental demuestra que esta implementación reduce el tiempo de procesamiento en casi un 85 % respecto a una configuración secuencial y supera en un 20 % al sistema de referencia Parsl + Globus Compute.

El caso de estudio sobre los recursos hídricos de la Laguna Santa Olalla pone de manifiesto la utilidad práctica de este trabajo para estudios ambientales a gran escala. Los productos derivados de NDWI revelan tendencias de reducción hídrica que podrían escalar a regiones más amplias para apoyar políticas de gestión del agua. Como trabajo futuro, se planea extender los estudios ambientales para incluir la detección de cambios en vegetación y zonas urbanas. También se explorarán la fusión de datos meteorológicos de estaciones terrestres con índices derivados de imágenes satelitales para un análisis enriquecedor, así como la validación de este trabajo mediante despliegues de larga duración y su integración con canalizaciones de observación de la Tierra en producción.

Agradecimientos

El presente trabajo ha sido parcialmente financiado por la Agencia Española de Investigación a través del proyecto “Nuevas técnicas escalables de E/S para cargas de trabajo híbridas de HPC e intensivas en datos (SCIOT)” con referencia PID2022-138050NB-I00 y la Comunidad de Madrid con el programa “Ayudas para la contratación de personal investigador predoctoral en formación para el año 2022” con referencia PIPF-2022/COM-25578..

Referencias

[1] A. Ullah, T. Kiss, J. Kovács, F. Tusa, J. Deslauriers, H. Dagdeviren, R. Arjun, and H. Hamzeh, “Orchestration in the cloud-to-things compute continuum: taxonomy, survey and future directions,” *Journal of Cloud Computing*, vol. 12, no. 1, pp. 135, 2023.

[2] A. Maia, A. Boutouchent, Y. Kardjadja, M. Gherari, E. G. Soyak, M. Saqib, et al., “A survey on integrated computing, caching, and communication in the cloud-to-edge continuum,” *Computer Communications*, 2024.

[3] T. C. Vance, T. Huang, and K. A. Butler, "Big data in earth science: Emerging practice and promise," *Science*, vol. 383, no. 6688, pp. eadh9607, 2024.

[4] A. Karatosun, C. Pisa, T. Kaprol, V. Baousis, and M. Albughdadi, "A replicable multi-cloud automation architecture for earth observation," in *EGU General Assembly Conference Abstracts*, 2024, p. 1857.

[5] R. Wilkinson, M. Mleczko, R. Brewin, K. Gaston, M. Mueller, J. Shutler, X. Yan, and K. Anderson, "Environmental impacts of earth observation data in the constellation and cloud computing era," *Science of The Total Environment*, vol. 909, pp. 168584, 2024.

[6] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, and O. Rana, "The internet of things, fog and cloud continuum: Integration and challenges," *Internet of Things*, vol. 3, pp. 134–155, 2018.

[7] E. Deelman, R. F. da Silva, K. Vahi, M. Rynge, R. Mayani, R. Tanaka, W. Whitcup, and M. Livny, "The pegasus workflow management system: translational computer science in practice," *Journal of Computational Science*, vol. 52, pp. 101200, 2021.

[8] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, "Nextflow enables reproducible computational workflows," *Nature Biotechnology*, vol. 35, no. 4, pp. 316–319, 2017.

[9] S. Kounev, N. Herbst, C. L. Abad, A. Iosup, I. Foster, P. Shenoy, O. Rana, and A. A. Chien, "Serverless computing: What it is, and what it is not?," *Communications of the ACM*, vol. 66, no. 9, pp. 80–92, 2023.

[10] M. P. Ueckermann, J. Bieszczad, D. Entekhabi, M. L. Shapiro, D. R. Callendar, D. Sullivan, and J. Milloy, "PODPAC: open-source python software for enabling harmonized, plug-and-play processing of disparate earth observation data sets and seamless transition onto the serverless cloud," *Earth Science Informatics*, vol. 13, pp. 1507–1521, 2020.

[11] Ian Foster, "Globus online: Accelerating and democratizing science through cloud-based services," *IEEE Internet Computing*, vol. 15, no. 3, pp. 70–73, 2011.

[12] R. Chard, Y. Babuji, Z. Li, T. Skluzacek, A. Woodard, B. Blaiszik, I. Foster, and K. Chard, "funcX: A federated function serving fabric for science," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, 2020, pp. 65–76.

[13] D.-N. Le, S. Pal, and P. K. Pattnaik, "OpenFaaS," in *Cloud Computing Solutions: Architecture, Data Storage, Implementation and Security*, pp. 287–303. John Wiley & Sons, 2022.

[14] Amazon Web Services, "AWS Lambda," `https://aws.amazon.com/lambda/`, Accedido: 2025.

[15] Google Cloud, "Cloud functions," `https://cloud.google.com/functions`, Accedido: 2025.

[16] A. Kurniawan and W. Lau, "Introduction to Azure functions," in *Practical Azure Functions: A Guide to Web, Mobile, and IoT Applications*, pp. 1–21. Packt Publishing Ltd, 2019.

[17] Y. Li, R. Chard, Y. Babuji, K. Chard, I. Foster, and Z. Li, "UniFaaS: Programming across distributed cyberinfrastructure with federated function serving," in *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2024, pp. 217–229.

[18] Apache Software Foundation, "Apache OpenWhisk," `https://openwhisk.apache.org/`, Accedido: 2025.

[19] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. M. Wozniak, I. Foster, et al., "Parsl: Pervasive parallel programming in Python," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, 2019, pp. 25–36.

[20] C. Chen, W. Sun, Z. Yang, G. Yang, M. Jia, Z. Zhang, et al., "Tracking dynamics characteristics of tidal flats using Landsat time series and Google Earth Engine cloud platform," *Resources, Conservation and Recycling*, vol. 209, pp. 107751, 2024.

[21] M. Abughazalah, W. Alsaggaf, S. Saifuddin, and S. Sarhan, "Centralized vs. decentralized cloud computing in healthcare," *Applied Sciences*, vol. 14, no. 17, pp. 7765, 2024.

[22] D. Kaiser, B. Dovhan, A. Bauer, and S. Kounev, "Towards splitting monolithic workflows into serverless functions and estimating their run-time in the earth observation domain (poster)," in *SSP*, 2021, pp. 1–2.

[23] T. Pfandzelter and D. Bermbach, "Komet: A serverless platform for low-earth orbit edge services," in *Proceedings of the 2024 ACM Symposium on Cloud Computing*, 2024, pp. 866–882.

[24] Christian Sicari, Lorenzo Carnevale, Antonino Galletta, and Massimo Villari, "Openwolf: A serverless workflow engine for native cloud-edge continuum," in *2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, 2022, pp. 1–8.

[25] Dante Domizzi Sanchez-Gallegos, Diana Di Luccio, Sokol Kosta, JL Gonzalez-Compean, and Raffaele Montella, "An efficient pattern-based approach for workflow supporting large-scale science: The dagonstar experience," *Future Generation Computer Systems*, vol. 122, pp. 187–203, 2021.

[26] Sebastian Henningsen, Martin Florian, Sebastian Rust, and Björn Scheuermann, "Mapping the interplanetary filesystem," in *2020 IFIP Networking Conference (Networking)*. IEEE, 2020, pp. 289–297.

[27] D. D. Sánchez-Gallegos, D. Carrizales-Espinoza, A. Zequeira, C. Torres-Charles, J. L. Gonzalez-Compean, and J. Carretero, "GeoNimbus: A serverless framework to build earth observation and environmental services," in *Euro-Par 2024*. 2025, vol. 15386 of *Lecture Notes in Computer Science*, pp. 1–12, Springer Nature Switzerland AG.

[28] D. Yan, X. Wang, X. Zhu, C. Huang, and W. Li, "Analysis of the use of NDWIgreen and NDWIred for inland water mapping in the Yellow River Basin using Landsat-8 OLI imagery," *Remote Sensing Letters*, vol. 8, no. 10, pp. 996–1005, 2017.

Clasificación y segmentación de tumores de hueso largo en tomografía computarizada

Máximo Rodríguez Herrero[1], Dante Sánchez Gallegos [1] y Jesús Carretero [1]

Resumen— **En este artículo se presenta una herramienta de apoyo al diagnóstico de cáncer de hueso largo mediante técnicas de aprendizaje profundo, en colaboración con el Instituto Mexicano de Rehabilitación Luis Guillermo Ibarra Ibarra. El cáncer de hueso largo es menos frecuente que otros tumores, como el de pulmón, pero tiene un impacto significativo en la población. La solución propuesta se centra en el diseño, optimización y evaluación de modelos de aprendizaje profundo capaces de analizar tomografías computarizadas, identificar posibles regiones afectadas por tumores en huesos largos y generar una segmentación preliminar de las regiones con tumor. Para ello se emplean redes neuronales convolucionales y técnicas como el aprendizaje por transferencia, con el fin de optimizar el rendimiento del sistema. Asimismo, se realiza un análisis exhaustivo de los resultados para la segmentación y clasificación de pacientes a partir de tomografías (TAC), destacando fortalezas y limitaciones del enfoque propuesto.**

Palabras clave— **Aprendizaje Profundo, Aprendizaje por Transferencia, Osteosarcoma, Tomografía Computarizada, Reconstrucción 3D.**

I. Introducción

En los últimos años, los avances en inteligencia artificial (IA) han potenciado significativamente la automatización de diversos procesos dentro de los sistemas de *eHealth*. Las principales aplicaciones de la IA en imagen médica asisten a médicos y especialistas en la identificación de regiones de interés, reduciendo así el tiempo requerido para el diagnóstico y pronóstico [1], [2]. En consecuencia, los sistemas impulsados por IA tienen el potencial de habilitar tratamientos adaptativos y personalizados, monitoreo en tiempo real y un soporte diagnóstico mejorado [3].

La segmentación y clasificación de imágenes basada en aprendizaje profundo, liderada por arquitecturas como U-Net y sus variantes [4], ha dominado los principales *benchmarks* de segmentación médica en los últimos años. A pesar de este éxito, la viabilidad clínica sigue siendo un desafío debido a la escasez de datos etiquetados, las estrictas regulaciones de protección de datos que limitan el intercambio de conjuntos de datos, y la complejidad técnica asociada con la configuración, entrenamiento y mantenimiento de estos modelos en entornos hospitalarios.

La segmentación de imágenes en entornos de *eHealth* requiere abordar varias necesidades adicionales, tales como *i)* proporcionar un mecanismo intuitivo de visualización de datos, *ii)* la creación de un sistema confiable, seguro y privado para el intercambio y gestión de datos, *iii)* flujos de trabajo eficientes para la gestión, procesamiento y etiquetado de datos, *iv)* soporte para la explicabilidad de los modelos y confianza del usuario, y *v)* ofrecer pipelines accesibles para la implementación y mantenimiento [5]. Asimismo, la adopción de soluciones de aprendizaje profundo sigue estando limitada principalmente a instituciones con recursos suficientes. Clínicas más pequeñas o con recursos restringidos a menudo carecen de la infraestructura computacional, volumen de datos o experiencia técnica necesaria para desarrollar y desplegar estos modelos de manera efectiva más allá de proyectos piloto [6], [7]. Aunque existen soluciones *cloud* que han democratizado el acceso a recursos en diversas industrias y áreas de investigación, su uso en entornos sanitarios puede generar preocupaciones de privacidad, complicando la adopción.

Considerando estos desafíos, en el grupo ARCOS estamos desarrollando **OsteoCAD**, un marco de *eHealth* diseñado específicamente para entornos clínicos con recursos limitados. OsteoCAD combina una arquitectura modular basada en contenedores con ejecución remota segura mediante GPU, permitiendo a los hospitales construir, entrenar y desplegar modelos de segmentación sin necesidad de *hardware* local de alto rendimiento ni personal especializado.

OsteoCAD está orientado a asistir en el marcado automático y ayudas al diagnóstico de cáncer en hueso largo, proporcionando flujos de trabajo clínicos de extremo a extremo, desde la creación del conjunto de datos y anotación con intervención humana hasta el despliegue de modelos de segmentación dentro de la práctica hospitalaria rutinaria. Además, OsteoCAD es agnóstico del dominio objetivo, por lo que puede ser utilizado por múltiples equipos con diferentes objetivos de segmentación, ya sea compartiendo el mismo conjunto de pacientes con anotaciones distintas o trabajando con conjuntos de datos completamente separados.

El modelo de trabajo propuesto se está validando mediante un caso de uso real enfocado en la detección de tumores de huesos largos en exploraciones por tomografía computarizada (TAC) en una clínica sin **ningún conjunto de datos existente** y **sin implementación previa de IA** en la rutina clínica. Esta evaluación abarca: (i) la construcción incremental de un conjunto de datos anotado que incluye **61 pacientes** aprobado bajo protocolos de investigación institucional, (ii) la evaluación del desempeño del modelo utilizando métricas estándar de imagen médica, alcanzando un coeficiente de Dice de **0,84 ± 0,02** en una cohorte inicial de 16 pacientes, y (iii) la demostración de la capacidad del marco para abordar un escenario completo de implementación

[1]Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: maxrodri@inf.uc3m.es

clínica en el mundo real.

En resumen, las contribuciones de este trabajo son:

1. El diseño e implementación de un marco modular de aprendizaje profundo que permite flujos de trabajo de segmentación remota acelerados por GPU en hospitales con recursos limitados.
2. La implementación clínica de un modelo de segmentación de tumores de huesos largos, permitiendo la expansión iterativa del conjunto de datos mediante intervención humana y la evaluación del desempeño en un caso de uso real.

El resto del documento se organiza de la siguiente manera: la Sección II revisa trabajos relacionados. La Sección III describe el proceso de construcción de un conjunto de datos para la clasificación y segmentación de tumores en huesos largos. La Sección IV muestra cómo se ha abordado la tarea crucial de preparar y evaluar los módulos de clasificación y segmentación. La Sección V muestra el marco experimental y las pruebas llevadas a cabo en el artículo. Finalmente, la Sección VI proporciona una discusión breve y las conclusiones.

II. Estado de la cuestión

El uso del aprendizaje automático (ML, Machine Learning) en el ámbito de la imagen médica se remonta a la década de 1960, con los primeros sistemas basados en características diseñadas manualmente y extraídas de radiografías [8]. Posteriormente, enfoques más avanzados basados en aprendizaje profundo (DL) comenzaron a surgir en la década de 1990, permitiendo abordar tareas como la detección de nódulos pulmonares [9], [10] y la clasificación de tejido mamario [11]. Estas soluciones superaron el enfoque de simples indicadores o visualizaciones, al ofrecer la posibilidad de desarrollar *pipelines* de diagnóstico completamente automatizados que asisten a los clínicos en los procesos de toma de decisiones.

En la última década, el aprendizaje profundo (DL) ha logrado avances significativos en la resolución de aplicaciones clásicas en imagen médica:

- **Clasificación:** Determinación de si una imagen médica pertenece a una categoría diagnóstica específica (por ejemplo, tumor frente a tejido sano) [1].
- **Segmentación:** Delimitación de estructuras anatómicas o regiones patológicas dentro de una imagen (por ejemplo, los contornos tumorales). Se considera la tarea más compleja debido a su alta dependencia de datos anotados, lo que requiere una supervisión intensiva [12]. Las tareas de segmentación pueden dividirse, de manera general, en dos categorías: segmentación de estructuras bien definidas, como órganos o huesos, y segmentación de lesiones. Esta última resulta más compleja, ya que las lesiones presentan una elevada variabilidad en forma, tamaño y apariencia, factores que incrementan el impacto de la escasez de datos etiquetados.
- **Detección:** Identificación de la presencia y localización de anomalías específicas [1].
- **Registro:** Alineación de imágenes obtenidas en distintos momentos temporales o mediante diferentes modalidades hacia un mismo marco de referencia [1].

Este artículo se centra específicamente en dos de estas tareas: clasificación y segmentación. Ambas se ven significativamente afectadas por la limitada disponibilidad de grandes conjuntos de datos anotados, lo que restringe el entrenamiento de redes neuronales profundas desde cero. Para superar esta limitación, el aprendizaje por transferencia se ha consolidado como la estrategia predominante [12]. En este enfoque, los modelos se preentrenan sobre grandes conjuntos de datos de imágenes naturales, como ImageNet [13], y posteriormente se ajustan (*fine-tuning*) sobre conjuntos de datos médicos, como se detallará en las secciones siguientes.

A. Detección y análisis de tumores en huesos largos

La solución propuesta en este artículo integra modelos de clasificación y segmentación para la detección y el análisis de tumores en huesos largos (e.g., osteosarcoma), los cuales se encuentran incorporados en un *pipeline* automatizado diseñado para asistir a los clínicos.

Diversos estudios han abordado la **clasificación y detección** de tumores en huesos largos mediante técnicas de aprendizaje automático. Mahore et al. [14] emplearon un clasificador *Random Forest* sobre imágenes de biopsias histopatológicas. Si bien la histopatología proporciona un alto nivel de detalle celular, depende de procedimientos invasivos, lo cual limita su aplicabilidad, especialmente en países de ingresos bajos y medios.

Por otro lado, la tomografía computarizada (TAC) constituye una alternativa no invasiva adecuada para el estudio de tumores en huesos largos, ya que permite capturar con precisión las estructuras óseas. Sampath et al. [15] llevaron a cabo un estudio comparativo utilizando diversas arquitecturas de aprendizaje profundo para la detección de cáncer óseo, destaca el uso de AlexNet sobre un conjunto de datos compuesto por 1,141 imágenes de TAC, donde reportaron una precisión del 100 % en el conjunto de prueba. No obstante, este resultado sugiere la posible presencia de sobreajuste o fuga de datos (imágenes de un mismo paciente en distintas particiones del *dataset*), particularmente considerando el tamaño relativamente limitado del conjunto de datos.

La **segmentación** constituye una de las tareas más complejas en el análisis de tumores en huesos largos debido a la heterogeneidad tumoral y a su presencia tanto en tejido óseo como en tejido blando. Dos contribuciones relevantes en este ámbito son las propuestas por Huang et al. [16] y Zhang et al. [17], ambos pertenecientes al mismo grupo de investigación y basados en un conjunto de datos compartido de 2,305 imágenes de TAC anotadas. En [16], los autores proponen una red convolucional autosupervisada (MSFCN), alcanzando un coeficiente de similitud

de Dice (DSC) del 87,80 %. Su enfoque se basa en la fusión de las salidas de tres cabezas de segmentación y el cálculo conjunto de la función de pérdida. Posteriormente, Zhang et al. [17] mejoraron este resultado (DSC de 89,22 %) mediante la integración de ResNet-50 como arquitectura base, con el objetivo de mitigar el problema de los gradientes desvanecientes, incorporando las cabezas de segmentación después de los bloques residuales. Cabe destacar que en ambos trabajos no se especifica claramente la separación de los datos a nivel de paciente entre los conjuntos de entrenamiento y prueba, lo que podría dar lugar a fuga de datos y a una sobreestimación del rendimiento de los modelos.

En el marco propuesto en este trabajo, se crea un pipeline en el que un modelo de clasificación filtra inicialmente las exploraciones de TAC con alta probabilidad de contener osteosarcoma. Posteriormente, dichas imágenes son procesadas mediante un módulo de segmentación, el cual permite reconstruir un modelo tridimensional de la región esquelética afectada, facilitando la interpretación clínica y la visualización de estructuras anómalas. Con el fin de proporcionar un mayor contexto espacial, se aplica un modelo de segmentación específico sobre las regiones identificadas, delimitando el tumor en cada corte de TAC. Este procedimiento proporciona información preliminar sobre la forma, el volumen y la extensión del tumor, contribuyendo a una toma de decisiones clínicas más fundamentada.

Mientras que otros estudios se centran principalmente en presentar arquitecturas novedosas, este artículo adopta un enfoque complementario basado en el aprovechamiento de arquitecturas consolidadas en el estado del arte, como U-Net++ y Residual U-Net, junto con la incorporación de técnicas de aprendizaje por transferencia para abordar la escasez de datos anotados. A diferencia del enfoque de Primakov et al. [18], en el que los modelos se entrenan desde cero, en este trabajo se emplean técnicas de aprendizaje por transferencia [19], lo que permite mejorar significativamente la eficiencia en el uso de los datos y el rendimiento de los modelos.

III. Análisis y preparación del conjunto de datos

En esta sección se describe el proceso de preparación de los conjuntos de datos utilizados en este trabajo para desarrollar aplicaciones de aprendizaje profundo (DL). Esto incluye un *pipeline* que transforma los datos médicos crudos en conjuntos de datos estructurados. Los datos médicos crudos están compuestos por imágenes de tomografía axial computarizada (TAC).

Este trabajo se basa en datos de imagen médica obtenidos del PACS-INR (Picture Archiving and Communication System) del Instituto Nacional de Rehabilitación "Luis Guillermo Ibarra Ibarra"[1]. Estas imágenes fueron compartidas con fines de investigación. El INR LGII proporcionó exploraciones por

[1]https://www.inr.gob.mx/

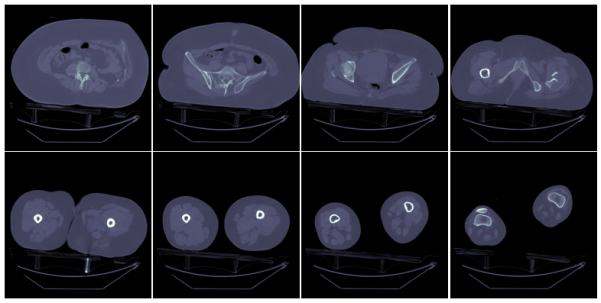


Fig. 1: Conjunto de imágenes iniciales de TAC en plano axial de un único paciente. La secuencia representa cortes axiales ordenados desde la cadera hasta la parte inferior de la pierna.

TAC adquiridas entre 2015 y 2024 de una cohorte de 61 pacientes diagnosticados con tumores en huesos largos [20]. Los pacientes tenían edades comprendidas entre 10 y 76 años. De ellos, el 53,5 % eran mujeres, y la lateralidad del miembro afectado se distribuyó de manera equitativa entre el lado izquierdo y derecho. Originalmente adquiridas en formato DICOM, las imágenes fueron anonimizadas y convertidas a JPEG de acuerdo con las pautas éticas del centro. Oncólogos expertos del centro anotaron las imágenes antes de compartirlas con fines de investigación. El conjunto de datos consiste en imágenes secuenciales de TAC en plano axial para cada paciente, con una resolución espacial de 512×512 píxeles por corte. El campo de visión abarca tanto el miembro sano como el afectado. La Figura 1 muestra un subconjunto de imágenes crudas de un paciente representativo, ilustrando la progresión anatómica desde la cadera hasta las extremidades inferiores.

El conjunto de datos incluye dos tipos de anotaciones: cajas (*bounding boxes*) y máscaras poligonales (*polygon masks*). La disponibilidad limitada de datos de imagen médica anotados es un desafío conocido en este dominio. Este conjunto de datos refleja dicha problemática, ya que contiene anotaciones para únicamente 45 pacientes con cajas y 16 pacientes con máscaras poligonales. La Tabla I resume la composición del conjunto de datos. Cabe destacar que, en ambas modalidades de anotación, menos del 20 % de los cortes de TAC totales están realmente etiquetados, lo que evidencia el problema de la escasez de anotaciones.

A. Preprocesamiento de Datos

En esta sección, se describe el *pipeline* de preprocesamiento automático diseñado para procesar tanto el conjunto de datos crudo como las exploraciones TAC de nuevos pacientes en tiempo real, previo a la inferencia del modelo. El pipeline consta de tres etapas principales: (1) eliminación de artefactos y segmentación del primer plano (*foreground*), (2) recorte guiado en 3D, y (3) extracción de la región de interés (ROI). Estos pasos aseguran la consistencia espacial entre cortes, facilitan el entrenamiento y la inferencia de los modelos, y las transformaciones reversibles permiten reconstrucciones 3D sencillas a partir del escaneo TAC original.

Primero, cada corte de TAC se limpia para elimi-

Tabla I: Resumen del conjunto de datos proporcionado.

Tipo de Anotación	# Pacientes	# Cortes de TC	# Cortes Anotados	Cobertura de Anotación (%)
Caja delimitadora (*Bounding Box*)	45	6708	1221	18,2 %
Polígono (*Polygon*)	16	2742	291	10,6 %

nar artefactos de imagen, como la mesa del escáner, y se extrae una máscara binaria del primer plano para aislar las estructuras anatómicas relevantes. Específicamente, nos enfocamos en segmentar la región de la pierna del fondo, sin necesidad de una delineación detallada.

En segundo lugar, el volumen de TAC se recorta considerando una vista global del conjunto 3D. En lugar de procesar cada corte de forma independiente, se calculan índices de recorte globales ($x_{\min}$, x_{mid}, $x_{\max}$, $y_{\min}$, $y_{\max}$) a partir de las máscaras del primer plano para aislar las regiones de pierna izquierda y derecha. Esta estrategia asegura continuidad espacial entre cortes, facilitando reconstrucciones 3D posteriores.

Finalmente, las secciones extraídas de las piernas izquierda y derecha se preprocesan individualmente, aislando las ROI y aplicando operaciones generales como cambio de tamaño (*resizing*) y normalización de intensidad.

A.1 Eliminación de Artefactos y Segmentación del Primer Plano

El conjunto de datos crudo consiste en exploraciones TAC que contienen ambas piernas o secciones completas de cadera. Un artefacto común en los escaneos sin procesar es la mesa del paciente, que aparece de forma prominente en muchos cortes. El objetivo de esta etapa es eliminar dichos artefactos, reducir el ruido y extraer una máscara limpia del primer plano que incluya las dos regiones de pierna (es decir, dos componentes conectados laterales grandes correspondientes a las extremidades inferiores).

Esta parte del pipeline, representada en la Figura 2, opera únicamente a nivel de corte 2D; cada corte axial se procesa de forma independiente del volumen 3D. Comenzamos aplicando una operación de apertura morfológica (*morphological opening*) utilizando un elemento estructurante en forma de disco con radio de 10 píxeles. Este paso rompe efectivamente conexiones pequeñas, suaviza el ruido y separa la mesa de la anatomía.

A continuación, aplicamos el método de Otsu [21] (algoritmo clásico de umbralización no supervisada) para generar una máscara binaria del primer plano. A partir de esta máscara, se extraen los dos componentes conectados más grandes ubicados en la parte superior del corte, correspondientes a las piernas del paciente.

A.2 Mejora de Contraste y Normalización de Imagen

Una posible extensión del pipeline de preprocesamiento incluye la mejora de contraste y la normalización de intensidad, para mejorar la visibilidad de estructuras relevantes y estandarizar la entrada para la red neuronal.

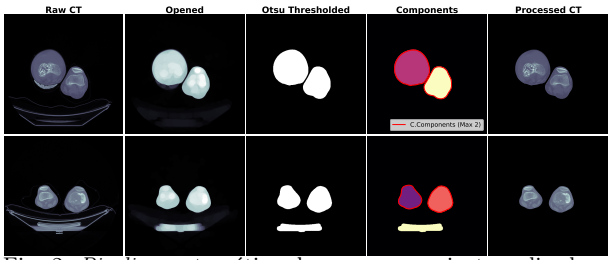


Fig. 2: *Pipeline* automático de preprocesamiento aplicado a cada exploración TAC de un paciente. De izquierda a derecha: (1) imagen TAC cruda con artefactos de la mesa del escáner, (2) resultado de la apertura morfológica utilizando un elemento estructural en forma de disco con radio de 10 px, (3) máscara binaria obtenida mediante el método de Otsu, y (4) máscara final conservando únicamente los dos componentes conectados más grandes correspondientes a las regiones de las piernas.

La técnica más frecuente es CLAHE (*Contrast Limited Adaptive Histogram Equalization*). Funciona realzando localmente el contraste en pequeñas regiones (*tiles*) mientras previene la amplificación de ruido al recortar los picos extremos del histograma. Esto es particularmente útil para cortes TAC médicos, donde los tejidos blandos pueden aparecer con contraste muy bajo respecto a los huesos.

Después de la mejora de contraste, la normalización para escalar las intensidades de píxel a $[0, 1]$ y la estandarización a media cero y varianza unitaria son operaciones comunes para homogeneizar la entrada del modelo.

La Figura 3 muestra un ejemplo del pipeline de preprocesamiento extendida con estas operaciones. En comparación con la imagen cruda, la versión mejorada resalta estructuras óseas y otros detalles de alto contraste que resultan más informativos para la inspección visual.

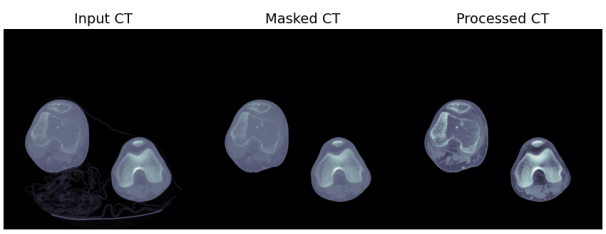


Fig. 3: Ejemplo del pipeline de preprocesamiento extendida con CLAHE, normalización y estandarización.

Finalmente, la Figura 4 muestra un corte TAC procesado junto con su histograma. En comparación con el histograma crudo, dominado por píxeles de fondo, el histograma mejorado separa claramente las regiones más brillantes correspondientes al hueso.

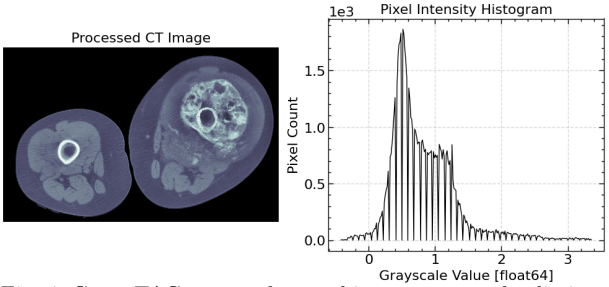


Fig. 4: Corte TAC procesado y su histograma tras la eliminación del fondo, mejora de contraste (CLAHE) y estandarización de intensidad.

B. *Recorte Espacialmente Alineado Basado en Proyecciones de Intensidad*

En esta sección se describe un método para calcular índices de corte que dividen espacialmente un escaneo TAC 3D en dos cajas delimitadoras, cada una conteniendo una pierna. El enfoque se basa en una colocación decúbito supino sobre la mesa de escaneo, con ambas piernas extendidas a lo largo del eje frontal (x).

La Figura 5 ilustra el proceso para calcular los índices de corte. Primero, dado un volumen TAC 3D $\mathrm{CT}(z, y, x)$, se proyecta a lo largo del eje axial (z) para obtener un mapa de intensidad 2D:

$$D_z(y, x) = \sum_{z=1}^{N_z} \mathrm{CT}(z, y, x).$$

A partir de esta proyección 2D, se calculan dos perfiles de intensidad 1D: uno a lo largo del eje sagital (y) y otro a lo largo del eje frontal (x):

$$D_{yz}(y) = \sum_{x=1}^{512} D_z(y, x) \tag{1}$$

$$D_{xz}(x) = \sum_{y=1}^{512} D_z(y, x) \tag{2}$$

Estos perfiles 1D capturan la distribución general de la intensidad TAC a lo largo de cada eje. En el perfil de proyección frontal $D_{xz}(x)$, típicamente se observan dos picos de alta intensidad correspondientes a las piernas izquierda y derecha. Se definen $x_{\min}$ y $x_{\max}$ como las primeras entradas no nulas en cada extremo, y x_{mid} como el mínimo local entre los picos, indicando el punto medio (por ejemplo, la región de la cadera).

De manera similar, en el perfil sagital $D_{yz}(y)$ se identifican $y_{\min}$ y $y_{\max}$ como los límites del paciente en el eje vertical, correspondientes a los primeros y últimos valores no nulos del perfil.

El pipeline de recorte completo se ilustra en la Figura 6, donde se muestra la pierna aislada final. Este método de recorte es específico para cada paciente, lo que significa que las imágenes resultantes pueden variar en tamaño y forma según la anatomía individual. En la siguiente sección se describe cómo se extrae una ROI consistente a partir de estos volúmenes recortados y cómo se estandariza cada imagen para que coincida con las dimensiones de entrada esperadas por el modelo.

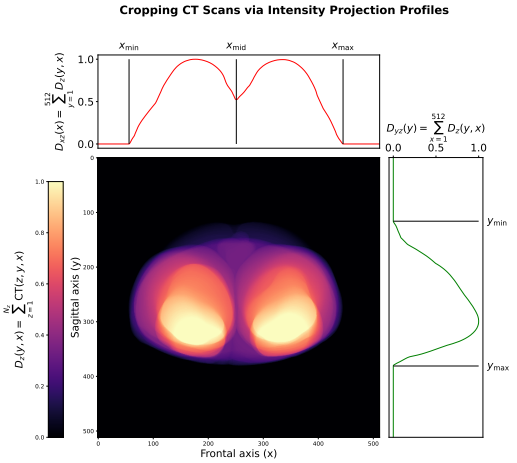


Fig. 5: Recorte del escaneo TAC basado en perfiles de proyección de intensidad. Los índices de corte ($x_{\min}$, x_{mid}, $x_{\max}$, $y_{\min}$, $y_{\max}$) se derivan de mínimos locales y límites de soporte en los perfiles de intensidad 1D $D_{xz}(x)$ y $D_{yz}(y)$. Estos perfiles se calculan primero proyectando el volumen TAC 3D $\mathrm{CT}(z, y, x)$ a lo largo del eje axial (z) para obtener $D_z(y, x)$, y luego sumando filas y columnas para obtener $D_{xz}(x)$ y $D_{yz}(y)$, respectivamente.

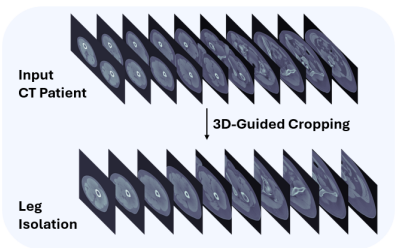


Fig. 6: Representación esquemática del método de recorte guiado en 3D. La fila superior muestra el volumen TAC 3D completo de un paciente, mientras que la fila inferior muestra los cortes recortados resultantes, aislando una sola pierna.

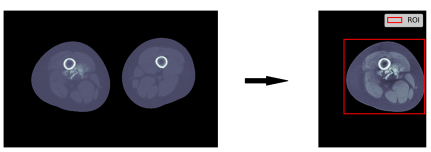


Fig. 7: ROI final extraída del TAC utilizada como entrada para el modelo.

C. *Extracción de la Región de Interés (ROI)*

Tras estos pasos, las imágenes originales de 512×512 píxeles se recortan en dos ROI principales, cada una representando una pierna. Ahora, las imágenes se normalizan por intensidad corte a corte utilizando la fórmula $I_N = (I - \mathrm{mín}(I))/(\mathrm{máx}(I) - \mathrm{mín}(I))$ y se redimensionan a 256×256 píxeles mediante interpolación por vecino más cercano (*nearest-neighbor*). Un ejemplo de ROI resultante después de los pasos de preprocesamiento y recorte se muestra en la Figura 7.

El uso de interpolación por vecino más cercano se prefiere para preservar la naturaleza booleana de las máscaras de segmentación sin requerir pasos de procesamiento adicionales.

IV. Clasificación y segmentación de imágenes

En esta sección se muestra cómo se aborda la tarea crucial de preparar y evaluar los módulos de clasifi-

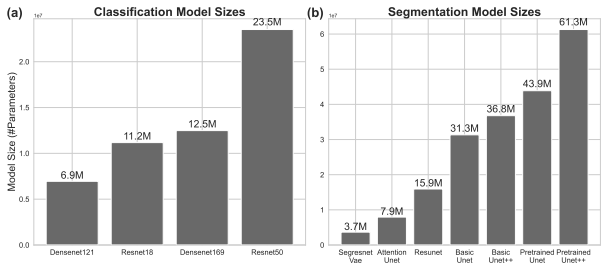


Fig. 8: Comparación de los tamaños de los modelos candidatos para la tarea de clasificación (a) y de segmentación (b).

cación y segmentación. Para ello, se crearon dos conjuntos de datos dedicados en colaboración con INR LGII: uno para clasificación (2.910 imágenes) y otro para segmentación (291 imágenes). A continuación, se implementó un pipeline en dos etapas: un modelo DenseNet para la clasificación inicial de los cortes de TAC, seguido de un modelo UNet++ para segmentar los tumores en las imágenes identificadas como positivas. Estas arquitecturas fueron seleccionadas tras un análisis cuidadoso de los avances recientes y fueron entrenadas utilizando una estrategia de ajuste gradual (*fine-tuning*) desarrollada específicamente para este proyecto. El desempeño de los modelos se evaluó mediante métricas estándar de evaluación, garantizando tanto la transparencia como la relevancia clínica.

La implementación del software se basó principalmente en dos bibliotecas de Python:

1. MONAI[2] (Medical Open Network for Artificial Intelligence) [22], un framework de código abierto basado en PyTorch desarrollado para aplicaciones de imagen médica, y
2. SMP[3] (Segmentation Models PyTorch) [23], una biblioteca ampliamente utilizada para tareas de segmentación semántica, comúnmente empleada en competiciones de segmentación.

Para facilitar el aprendizaje por transferencia y acelerar la convergencia, se inicializaron las redes con pesos preentrenados en ImageNet, disponibles a través del repositorio oficial de modelos de PyTorch[4] y compatibles con MONAI y SMP.

Para la tarea de **clasificación**, se emplearon cuatro arquitecturas CNN bien establecidas: ResNet18, ResNet50, DenseNet121 y DenseNet169. Los cuatro modelos se obtuvieron de la biblioteca MONAI, utilizando las configuraciones por defecto, salvo que se indique lo contrario. Estos modelos se adaptan a imágenes médicas 2D en escala de grises, empleando una dimensionalidad espacial de 2 y un único canal de entrada. La Figura 8a presenta los tamaños correspondientes de los modelos de clasificación en términos de parámetros entrenables, y las comparaciones de configuración se muestran en las Tablas II y III.

Las arquitecturas seleccionadas para la tarea de **segmentación** son proporcionadas directamente por MONAI. Sin embargo, para explorar de manera sistemática estrategias de ajuste fino (*fine-tuning*) basadas en el codificador, se consideraron adicionalmente Pretrained UNet y Pretrained UNet++, ambos disponibles en Segmentation Models PyTorch (SMP).

Tabla II: Configuración de ResNet18 y ResNet50.

Característica	**ResNet18**	**ResNet50**
Tipo de bloque	Básico	Bottleneck
Shortcut type	Identity	1×1 + Batch Norm
Bias in downsample	Verdadero	Falso
Bloues totales	8 (2 por grupo)	16 (3/4/6/3 por grupo)
Capas convolucionales por bloque	2	3
Capas entrenables	18	50

Tabla III: Configuracion de DenseNet121 y DenseNet169.

Característica	**DenseNet121**	**DenseNet169**
Filtros convolucionales iniciales	64	64
Tasa de aprendizaje	32	32
Configuración de los bloques densos	[6, 12, 24, 16]	[6, 12, 32, 32]
Capas convolucionales por unidad	2	2
Capas de transición	3	3
Capas entrenables	121	169

Como se ilustra en la Figura 8b, los modelos de segmentación suelen ser más grandes debido a las capas adicionales del decodificador. Los modelos SMP con codificadores ResNet50 preentrenados incrementan aún más el tamaño del modelo: Pretrained UNet es aproximadamente un 40 % más grande que UNet básico, mientras que Pretrained UNet++ es aproximadamente un 67 % más grande que UNet++ básico. Salvo que se indique lo contrario, todas las configuraciones siguen los ajustes por defecto de MONAI o SMP. Todos los modelos se adaptan a imágenes médicas 2D en escala de grises, con una dimensionalidad espacial de dos y un único canal de entrada.

Para resumir los **resultados obtenidos**, la tarea de clasificación alcanzó un AUC de 0,95, con una sensibilidad de 0,83 y una especificidad de 0,95. Para la tarea de segmentación, se obtuvo un DSC medio de 0,85, muy cercano al 0,89 reportado por Zhang et al. [17], a pesar de la diferencia significativa en el tamaño de los conjuntos de datos (2.305 imágenes frente a nuestras 291). Esto demuestra que nuestras estrategias de aprendizaje por transferencia y ajuste fino (*fine-tuning*) permiten aproximarse al rendimiento de última generación en el diagnóstico de tumores de hueso largo, incluso en condiciones de escasez de datos. Es importante enfatizar que estos resultados numéricos por sí solos no son totalmente significativos sin una adecuada contextualización. Por ello, realizamos un análisis cualitativo del comportamiento de los modelos, incluyendo ejemplos visuales de las predicciones generadas por nuestros modelos finales ajustados: DenseNet121-FT para clasificación y U-Net++-LP-FT para segmentación.

En cuanto a la segmentación, se lograron resultados sólidos, como se ilustra en las Figuras 9, 10 y 11. En estos ejemplos, las máscaras predichas se asemejan estrechamente a las delimitaciones de referencia (*ground-truth*) del tumor, y los mapas de confianza indican claramente que el modelo evita con éxito el

[2] `https://monai.io/`
[3] `https://github.com/qubvel-org/segmentation_models.pytorch`
[4] `https://pytorch.org/vision/main/models.html`

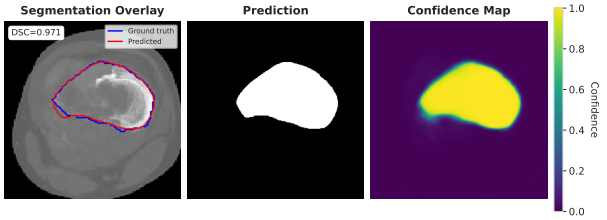


Fig. 9: Resultados de segmentación y mapa de confianza correspondiente para la muestra con mejor desempeño.

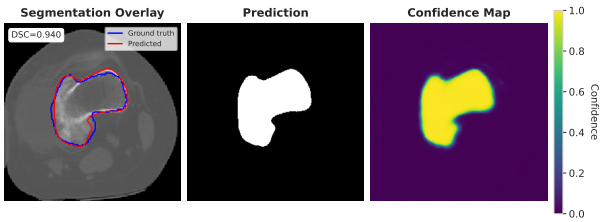


Fig. 10: Resultados de segmentación y mapa de confianza correspondiente para la muestra con desempeño medio.

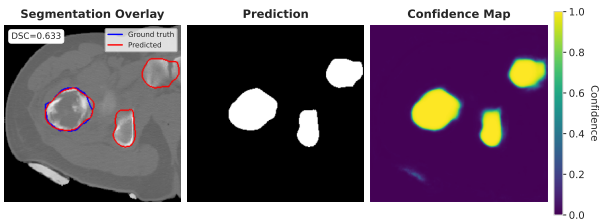


Fig. 11: Resultados de segmentación y mapa de confianza correspondiente para la muestra con peor desempeño.

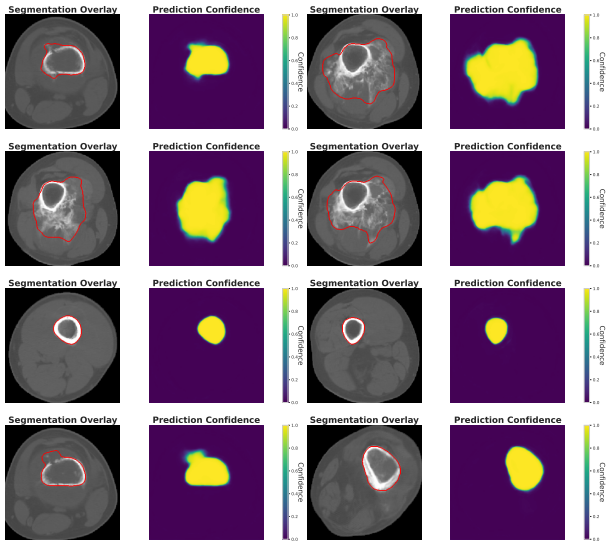


Fig. 12: Predicciones del modelo de segmentación y mapas de confianza correspondientes sobre el conjunto de prueba anotado para clasificación.

Tabla IV: Características del nodo `compute-17-3` del clúster Tucan de ARCOS.

Característica	Valor
ID del nodo	compute-17-3
Procesador	Intel(R) Xeon(R) Silver 4516Y+ @ 2.20GHz
Núcleos físicos	48
CPUs lógicas	96
Hilos por núcleo	2
Memoria RAM total	251.56 GB
GPU	NVIDIA L40S
Memoria GPU	46.07 GB

tejido no tumoral en la imagen. De manera positiva, el modelo ignoró correctamente un artefacto de imagen en la parte inferior de la pierna, como se evidencia por la mínima atención que recibe en el mapa de confianza. Los peores casos se dan en la cadera por falta de datos.

Se realizó una evaluación adicional utilizando el conjunto de datos de clasificación, que contiene imágenes anotadas para diagnóstico pero no para segmentación. Este análisis ofrece dos ventajas clave: (1) evita completamente cualquier riesgo de *data leakage*, dado que los conjuntos de datos son independientes, y (2) proporciona una prueba de generalización más desafiante, ya que las imágenes provienen de distintas fechas de adquisición y potencialmente de diferentes condiciones de máquina. Como se muestra en la Figura 12, el modelo de segmentación demuestra una localización del tumor sorprendentemente precisa, incluso cuando se evalúa sobre este conjunto de datos externo.

V. Evaluación

Para establecer una comparación justa y consistente entre los modelos candidatos para las tareas de clasificación y segmentación, se diseñó un marco experimental unificado para cada experimento principal, asegurando la consistencia mediante la conservación del entorno experimental sin modificaciones.

Todos los experimentos emplearon los conjuntos de datos descritos en la Sección III, siguiendo una estrategia de validación cruzada de 5 etapas (*5-fold cross-validation*) con separación por paciente para prevenir filtración de datos y garantizar una evaluación robusta.

A. Entorno experimental

El entrenamiento y la evaluación de modelos de aprendizaje profundo requieren acceso a recursos computacionales de alto rendimiento, como GPUs. Para este trabajo, se proporcionó acceso al clúster *Tucan* del Departamento de Informática del grupo ARCOS[5] de la UC3M. Para garantizar la reproducibilidad y una comparación justa de los resultados de los modelos, todos los experimentos se ejecutaron de manera consistente en el mismo nodo de cómputo, cuyas especificaciones se resumen en la Tabla IV.

B. Selección de la arquitectura

Dado que no es práctico entrenar, evaluar y optimizar todas las arquitecturas de vanguardia, es necesario identificar primero el modelo más adecuado para las tareas en cuestión. En esta sección, se describe una configuración común de entrenamiento y evaluación, aplicada tanto a clasificación como a segmentación, para permitir una comparación justa entre arquitecturas candidatas y tomar una decisión informada sobre qué modelo optimizar en experimentos posteriores de transferencia de aprendizaje.

Las configuraciones de clasificación y segmentación comparten los siguientes supuestos centrales: Para cada variante de modelo, se utiliza la implementa-

[5]`https://www.uc3m.es/research-groups/arcos`

Tabla V: Configuración de los experimentos de selección de modelo para tareas de clasificación y segmentación.

Parámetro	Clasificación	Segmentación
Épocas	15	50
Función de pérdida	CE	$0{,}25 \cdot \mathcal{L}_{\text{Dice}} + 0{,}75 \cdot \text{CE}$
Optimizador	Adam	Adam
Tasa de aprendizaje	10^{-5}	10^{-5}
Métrica de validación	ROC AUC	DSC
Función de activación	Softmax (2 clases)	Sigmoid (umbral 0.5)

ción original proporcionada en la publicación correspondiente; Los modelos se entrenan desde cero, salvo que se especifique explícitamente; Las imágenes se redimensionan uniformemente a 256×256 píxeles y se normalizan al rango $[0, 1]$; No se aplican aumentos de datos (*data augmentation*) en esta etapa, para centrarse en el desempeño intrínseco del entrenamiento sin fuentes adicionales de variabilidad; Los parámetros de validación cruzada se mantienen consistentes entre experimentos y las semillas aleatorias se inicializan de manera controlada para garantizar reproducibilidad; Por último, el tamaño de lote se fija en 16 para todos los experimentos.

Para identificar la **arquitectura de clasificación**, cada modelo candidato se entrenó durante 15 épocas utilizando el optimizador Adam con tasa de aprendizaje 10^{-5}. Las métricas de validación se calcularon tras cada época para monitorear el desempeño. Se seleccionó el ROC AUC como métrica principal de evaluación durante el entrenamiento. Para cada pliegue, se preservó el *checkpoint* que alcanzó el AUC de validación más alto y posteriormente se evaluó en el conjunto de prueba. La configuración resumida se muestra en la Tabla V.

El enfoque para seleccionar la **arquitectura de segmentación** sigue los mismos principios fundamentales que la tarea de clasificación, con algunas diferencias clave. Primero, la función de pérdida combina la entropía cruzada (CE) y la pérdida de Dice ($\mathcal{L}_{\text{Dice}}$) con ponderaciones 0,75 y 0,25 respectivamente, para garantizar tanto la precisión general de la segmentación como la identificación efectiva de las regiones tumorales, abordando particularmente el desbalance de clases entre fondo y primer plano. Segundo, los modelos se entrenaron durante 50 épocas. Tercero, se utilizó el DSC como métrica principal de validación. La configuración resumida se presenta en la Tabla V.

C. Entrenamiento para Rendimiento Óptimo en la Tarea

En esta sección se describen los experimentos diseñados para identificar las mejores configuraciones de entrenamiento y estrategias de fine-tuning para las tareas de clasificación y segmentación. Salvo indicación contraria, todos los parámetros conservan las configuraciones detalladas en la sección anterior.

Se emplearon dos planificadores de tasa de aprendizaje (*learning rate schedulers*):

- Para la cabeza de clasificación o segmentación, se utilizó `ReduceLROnPlateau` (ROP), que depende de la métrica de validación: si no se observa mejora durante un número predefinido de épocas, la tasa de aprendizaje se reduce a la mitad, con un umbral mínimo de 10^{-6}.
- Para la columna vertebral (*backbone*; codificador-decodificador en el caso de segmentación), se aplicó `CosineAnnealingLR` (CA), con $T_{max} = 50$ y tasa mínima de 10^{-6}.

Aunque estas estrategias se introdujeron en el Capítulo de Estado del Arte II, aquí se aplican a ambos tipos de modelos:

1. FT (*Full Fine-Tuning*): Entrenamiento sobre toda la red utilizando un solo optimizador y el planificador CA. La métrica de validación se emplea únicamente para *early stopping* basado en un límite de pacientes por época.
2. G-LF (*Gradual Last-to-First*): El entrenamiento inicia con solo la cabeza de clasificación o segmentación descongelada. Tras N épocas consecutivas sin mejora en la métrica de validación, se descongela el último bloque del *backbone*. Este proceso se repite iterativamente: cada vez que la métrica de validación se estanca durante N épocas, se descongela el siguiente bloque anterior. El entrenamiento termina cuando todos los bloques están descongelados y no se observa mejora adicional según la paciencia de *early stopping* o cuando se alcanza el número máximo de épocas.
3. LP-FT (*Linear Probing seguido de Full Fine-Tuning*): Este enfoque comienza como G-LF, entrenando inicialmente solo la cabeza. Una vez que la métrica de validación se estanca durante N épocas, todos los bloques del *backbone* se descongelan simultáneamente, y el entrenamiento continúa con ambos planificadores y optimizadores activos.

A continuación se llevó a cabo el *Fine-Tuning* de los modelos de clasificación y segmentación. En el caso de clasificación, se evaluaron diferentes combinaciones de parámetros disponibles, incluyendo variaciones en la función de pérdida, el uso de aumentos de datos y el entrenamiento sobre el conjunto de datos curado mediante embeddings. Cada planificador de tasa de aprendizaje está asociado con su propio optimizador Adam. Una actividad similar se llevó a cabo con los modelos de segmentación para evaluar si las estrategias de fine-tuning podían mejorar el desempeño, la robustez o la estabilidad del entrenamiento en comparación con los modelos entrenados de manera estándar. Además, solo se incluyeron aumentos de datos como factor adicional para intentar mejorar el desempeño de las estrategias FT y LP-FT. Cabe mencionar que la estrategia G-LF no se implementó en segmentación, dado que abordar las relaciones complejas entre bloques de codificador y decodificador conectados mediante *skip connections* excede el alcance de este artículo. La Tabla VI muestra una comparación del desempeño de las estrategias de fine-tuning para el modelo DenseNet-121. Se presentan la media y el intervalo de confianza del 95 % [IC 95 %] calculados a partir de las cinco par-

ticiones de validación en el conjunto de prueba. En negrita se indica la mejor media por clase y experimento, y se resalta el mejor valor de métrica considerando todos los experimentos.

VI. Conclusiones

En este artículo se ha presentado OsteoCAD, un marco para la clasificación y segmentación de tumores de hueso largo usando aprendizaje profundo. Desde la perspectiva de los datos, este artículo arroja varias contribuciones clave: Un pipeline de preprocesamiento robusto, consistente y transparente es tan importante como los modelos de DL mismos; La escasez de datos, un desafío recurrente en imágenes médicas, afecta fuertemente el resultado, aunque uso de *transfer learning*, que demostró ser útil para mitigar la falta de datos anotados; la *data augmentation* también resultó invaluable, mejorando consistentemente la generalización y robustez de los modelos; en cuanto a la manipulación de datos, es siempre mejor priorizar la transparencia y evitar cualquier práctica que pudiera inflar métricas (por ejemplo, se impusieron divisiones por paciente para garantizar estimaciones realistas de desempeño). Este enfoque evidencia que algunos resultados publicados que reportan precisiones cercanas al 100 % sin tales precauciones deben interpretarse con cautela.

Sobre la aplicabilidad del DL en el ámbito médico, nuestros experimentos ilustran que, aunque el campo puede parecer complejo, sus ideas más efectivas son elegantemente simples: Garantizar un flujo de gradiente suave a través de conexiones residuales o densas sigue siendo fundamental para entrenar redes profundas, mientras que la segmentación se beneficia adicionalmente de la transferencia de información espacial de bajo nivel desde el codificador al decodificador (*skip connections*); Las estrategias de *transfer learning* se exploraron extensivamente y, consistentes con la literatura, los enfoques más simples a menudo fueron los más efectivos: el *full fine-tuning* para clasificación y el *linear probing seguido de full fine-tuning* para segmentación superaron estrategias más elaboradas, que tendieron a sobreajustarse;

Los resultados finales alcanzados son alentadores. La tarea de clasificación se resolvió usando DenseNet121 con un ROC AUC de 0.95, sensibilidad de 0.83 y especificidad de 0.95. La tarea de segmentación, basada en UNet++ con codificador ResNet-50, alcanzó un DSC de 0.85.

De cara al futuro, este trabajo abre varias líneas de investigación. Primero, la validación externa en conjuntos de datos independientes sigue siendo un paso crucial para garantizar la generalización más allá de nuestra cohorte local. Segundo, un modelo unificado capaz de realizar clasificación y segmentación sería una evolución natural, ya sea extendiendo un modelo tipo UNet con una cabeza de clasificación o usando nuestro DenseNet entrenado como codificador para un decodificador de segmentación dedicado. Tercero, pasar de un enfoque 2D por corte a modelos volumétricos 3D podría aprovechar la continuidad espacial, capturando el contexto anatómico específico de cada paciente que actualmente está subutilizado. Otra dirección prometedora es la explicabilidad del modelo: generar salidas interpretables, como mapas de activación de clases o visualizaciones de sensibilidad basadas en oclusión, puede fortalecer la confianza y trazabilidad clínica.

Agradecimientos

El presente trabajo ha sido parcialmente financiado por la Comunidad Autónoma de Madrid, bajo el PROGRAMA DE JÓVENES INVESTIGADORES 2025, referencia 09-PIN1-00005.4/2025, por la Agencia Española de Investigación a través del proyecto "Nuevas técnicas escalables de E/S para cargas de trabajo híbridas de HPC e intensivas en datos (SCIOT)" con referencia PID2022-138050NB-I00, y por la Universidad Carlos III de Madrid a través del proyecto "Nuevas técnicas para la gestión de fábricas de datos en el cómputo continuo" (2025/00822/001 - Proyectos JÓVENES PPIT2025).

Referencias

[1] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A.W.M. van der Laak, Bram van Ginneken, and Clara I. Sánchez, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60–88, 12 2017.

[2] Berkman Sahiner, Aria Pezeshk, Lubomir M. Hadjiiski, Xiaosong Wang, Karen Drukker, Kenny H. Cha, Ronald M. Summers, and Maryellen L. Giger, "Deep learning in medical imaging and radiation therapy," *Medical Physics*, vol. 46, no. 1, pp. e1–e36, 1 2019.

[3] Hiroshi Fujita, "AI-based computer-aided diagnosis (AI-CAD): the latest review to read first," *Radiological Physics and Technology*, vol. 13, no. 1, pp. 6–19, 3 2020.

[4] Nahian Siddique, Sidike Paheding, Colin P. Elkin, and Vijay Devabhaktuni, "U-Net and Its Variants for Medical Image Segmentation: A Review of Theory and Applications," *IEEE Access*, vol. 9, pp. 82031–82057, 2021.

[5] Masooma Hassan, Andre Kushniruk, and Elizabeth Borycki, "Barriers to and Facilitators of Artificial Intelligence Adoption in Health Care: Scoping Review.," *JMIR human factors*, vol. 11, no. 1, pp. e48633, 8 2024.

[6] Musitapa Maimaitiaili, Yiershatijiang Jiamaliding, Guangle Dai, Hui Xiao, Warisijiang Kuerbanjiang, and Yuexiong Yi, "Artificial Intelligence Platform Architecture for Hospital Systems: Systematic Review Corresponding Author," .

[7] Zosia Kmietowicz, "Radiologist shortage leaves patient care at risk, warns royal college," 8 2017.

[8] P. H. MEYERS, C. M. NICE, H. C. BECKER, W. J. NETTLETON, J. W. SWEENEY, and G. R. MECKSTROTH, "Automated Computer Analysis of Radiographic Images," *https://doi.org/10.1148/83.6.1029*, vol. 83, pp. 1029–1034, 12 1964.

[9] Shih Chung B. Lo, Shyh Liang A. Lou, Minze V. Chien, and Seong K. Mun, "Artificial Convolution Neural Network Techniques and Applications for Lung Nodule Detection," *IEEE Transactions on Medical Imaging*, vol. 14, no. 4, pp. 711–718, 1995.

[10] K. Kanazawa, Y. Kawata, N. Niki, H. Satoh, H. Ohmatsu, R. Kakinuma, M. Kaneko, N. Moriyama, and K. Eguchi, "Computer-aided diagnosis for pulmonary nodules based on helical CT images," *Computerized Medical Imaging and Graphics*, vol. 22, no. 2, pp. 157–167, 3 1998.

[11] Berkman Sahiner, Heang Ping Chan, Nicholas Petrick, Datong Wei, Mark A. Helvie, Dorit D. Adler, and Mitchell M. Goodsitt, "Classification of mass and normal breast tissue: A convolution neural network classifier with spatial domain and texture images," *IEEE Transactions on Medical Imaging*, vol. 15, no. 5, pp. 598–610, 1996.

Tabla VI: Comparación del desempeño de las estrategias de fine-tuning para el modelo DenseNet-121

Exp. ID	Class	Strategy	Accuracy	Precision	Recall	F1-Score
C	Healthy	FT	**0.785 [0.757, 0.813]**	0.736 [0.712, 0.759]	**0.889 [0.859, 0.919]**	**0.805 [0.780, 0.830]**
		LP-FT	0.781 [0.715, 0.848]	**0.751 [0.658, 0.844]**	0.858 [0.812, 0.903]	0.798 [0.750, 0.847]
		G-LF	0.749 [0.696, 0.802]	0.708 [0.644, 0.771]	0.859 [0.769, 0.950]	0.774 [0.727, 0.820]
	Tumor	FT	**0.785 [0.757, 0.813]**	**0.860 [0.823, 0.896]**	0.681 [0.651, 0.710]	**0.760 [0.728, 0.791]**
		LP-FT	0.781 [0.715, 0.848]	0.833 [0.795, 0.870]	**0.705 [0.555, 0.856]**	0.759 [0.665, 0.853]
		G-LF	0.749 [0.696, 0.802]	0.826 [0.750, 0.902]	0.638 [0.514, 0.763]	0.715 [0.639, 0.790]
CA	Healthy	FT	**0.864 [0.807, 0.921]**	**0.808 [0.727, 0.889]**	**0.964 [0.953, 0.974]**	**0.878 [0.833, 0.922]**
		LP-FT	0.822 [0.727, 0.918]	0.798 [0.748, 0.847]	0.863 [0.646, 1.081]	0.822 [0.696, 0.948]
		G-LF	0.825 [0.791, 0.860]	0.770 [0.723, 0.817]	0.932 [0.876, 0.989]	0.842 [0.813, 0.871]
	Tumor	FT	**0.864 [0.807, 0.921]**	**0.955 [0.947, 0.963]**	0.764 [0.640, 0.888]	**0.846 [0.770, 0.922]**
		LP-FT	0.822 [0.727, 0.918]	0.874 [0.714, 1.033]	**0.781 [0.705, 0.858]**	0.818 [0.744, 0.892]
		G-LF	0.825 [0.791, 0.860]	0.918 [0.859, 0.976]	0.718 [0.634, 0.802]	0.803 [0.757, 0.850]
CE	Healthy	FT	**0.800 [0.770, 0.829]**	**0.745 [0.720, 0.771]**	**0.911 [0.871, 0.952]**	**0.820 [0.792, 0.847]**
		LP-FT	0.762 [0.711, 0.813]	0.725 [0.653, 0.796]	0.857 [0.792, 0.922]	0.783 [0.744, 0.822]
		G-LF	0.744 [0.714, 0.775]	0.695 [0.665, 0.726]	0.872 [0.828, 0.917]	0.773 [0.747, 0.800]
	Tumor	FT	**0.800 [0.770, 0.829]**	**0.887 [0.841, 0.932]**	**0.688 [0.650, 0.726]**	**0.774 [0.741, 0.808]**
		LP-FT	0.762 [0.711, 0.813]	0.826 [0.776, 0.877]	0.667 [0.542, 0.792]	0.734 [0.661, 0.807]
		G-LF	0.744 [0.714, 0.775]	0.830 [0.781, 0.879]	0.617 [0.560, 0.674]	0.706 [0.666, 0.747]
CAE	Healthy	FT	**0.889 [0.879, 0.899]**	**0.849 [0.832, 0.867]**	**0.946 [0.914, 0.977]**	**0.895 [0.884, 0.906]**
		LP-FT	0.856 [0.824, 0.887]	0.805 [0.766, 0.843]	0.941 [0.904, 0.978]	0.867 [0.839, 0.895]
		G-LF	0.843 [0.786, 0.901]	0.798 [0.744, 0.852]	0.921 [0.857, 0.985]	0.855 [0.801, 0.908]
	Tumor	FT	**0.889 [0.879, 0.899]**	**0.940 [0.909, 0.971]**	**0.832 [0.805, 0.859]**	**0.882 [0.872, 0.893]**
		LP-FT	0.856 [0.824, 0.887]	0.930 [0.890, 0.970]	0.770 [0.714, 0.826]	0.842 [0.805, 0.879]
		G-LF	0.843 [0.786, 0.901]	0.907 [0.839, 0.976]	0.766 [0.694, 0.838]	0.830 [0.767, 0.893]
F	Healthy	FT	0.770 [0.715, 0.825]	0.760 [0.713, 0.807]	0.795 [0.647, 0.943]	0.772 [0.700, 0.845]
		LP-FT	**0.785 [0.751, 0.819]**	**0.764 [0.721, 0.808]**	0.830 [0.722, 0.938]	**0.793 [0.748, 0.837]**
		G-LF	0.776 [0.749, 0.803]	0.744 [0.732, 0.755]	**0.842 [0.784, 0.900]**	0.789 [0.758, 0.820]
	Tumor	FT	0.770 [0.715, 0.825]	0.796 [0.691, 0.901]	**0.745 [0.647, 0.843]**	0.764 [0.713, 0.815]
		LP-FT	**0.785 [0.751, 0.819]**	0.820 [0.741, 0.899]	0.740 [0.662, 0.818]	**0.775 [0.742, 0.807]**
		G-LF	0.776 [0.749, 0.803]	**0.820 [0.766, 0.874]**	0.711 [0.698, 0.723]	0.761 [0.739, 0.782]
FAE	Healthy	FT	**0.875 [0.823, 0.928]**	**0.878 [0.799, 0.957]**	0.876 [0.834, 0.918]	**0.876 [0.827, 0.925]**
		LP-FT	0.851 [0.819, 0.883]	0.863 [0.833, 0.892]	0.838 [0.729, 0.946]	0.847 [0.803, 0.891]
		G-LF	0.857 [0.824, 0.889]	0.825 [0.769, 0.881]	**0.911 [0.854, 0.967]**	0.864 [0.837, 0.891]
	Tumor	FT	**0.875 [0.823, 0.928]**	0.876 [0.837, 0.914]	**0.874 [0.781, 0.967]**	**0.874 [0.817, 0.931]**
		LP-FT	0.851 [0.819, 0.883]	0.849 [0.764, 0.933]	0.864 [0.813, 0.915]	0.853 [0.832, 0.874]
		G-LF	0.857 [0.824, 0.889]	**0.903 [0.855, 0.950]**	0.803 [0.716, 0.889]	0.847 [0.805, 0.890]

[12] Xuxin Chen, Ximin Wang, Ke Zhang, Kar-Ming Fung, Theresa C Thai, Kathleen Moore, Robert S Mannel, Hong Liu, Bin Zheng, and Yuchen Qiu, "Recent advances and clinical applications of deep learning in medical image analysis," *Medical Image Analysis*, vol. 79, pp. 4, 2022.

[13] Jia Deng, Wei Dong, Richard Socher, Li Jia Li, Kai Li, and Li Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," *2009 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009*, pp. 248–255, 2009.

[14] S Mahore, K Bhole, S Rathod 2021 8th International, and undefined 2021, "Machine Learning approach to classify and predict different Osteosarcoma types," *ieeexplore.ieee.orgS Mahore, K Bhole, S Rathod2021 8th International Conference on Signal Processing and, 2021•ieeexplore.ieee.org*.

[15] K Sampath, S Rajagopal, A Chintanpalli Scientific Reports, and undefined 2024, "A comparative analysis of CNN-based deep learning architectures for early diagnosis of bone cancer using CT images," *nature.comK Sampath, S Rajagopal, A ChintanpalliScientific Reports, 2024•nature.com*, vol. 14, pp. 2144.

[16] Lin Huang, Wei Xia, Bo Zhang, Bensheng Qiu, and Xin Gao, "MSFCN-multiple supervised fully convolutional networks for the osteosarcoma segmentation of CT images," *Computer Methods and Programs in Biomedicine*, vol. 143, pp. 67–74, 5 2017.

[17] Rui Zhang, Lin Huang, Wei Xia, Bo Zhang, Bensheng Qiu, and Xin Gao, "Multiple supervised residual network for osteosarcoma segmentation in CT images," *Computerized Medical Imaging and Graphics*, vol. 63, pp. 1–8, 1 2018.

[18] Sergey P. Primakov and et al., "Automated detection and segmentation of non-small cell lung cancer computed tomography images," *Nature Communications*, vol. 13, no. 1, 12 2022.

[19] Hee E. Kim, Alejandro Cosa-Linan, Nandhini Santhanam, Mahboubeh Jannesari, Mate E. Maros, and Thomas Ganslandt, "Transfer learning for medical image classification: a literature review," *BMC Medical Imaging 2022 22:1*, vol. 22, no. 1, pp. 1–13, 4 2022.

[20] Marco Antonio Núñez-Gaona and et al., "A product-based technology transfer model for medical image management systems: the PACS-INR case," *Investigación en Discapacidad*, vol. 10, no. 3, pp. 151–161, 2024.

[21] Nobuyuki Otsu, "Threshold Selection Method From Gray-Level Histograms.," *IEEE Trans Syst Man Cybern*, vol. SMC-9, no. 1, pp. 62–66, 1979.

[22] et al. Jorge Cardoso, "MONAI: An open-source framework for deep learning in healthcare," 2022.

[23] Pavel Iakubovskii, "Segmentation Models Pytorch," 2019.

K8S-sims: Framework de Pruebas para Simuladores de Kubernetes

Matias Medina, Vitor da Silva, Rosana Tomas, Fernando Guirado y Josep Lluis Lérida[1]

Resumen— **Experimentar con clústeres de Kubernetes para investigación supone un desafío considerable debido a la inversión de tiempo y recursos necesaria, así como a la dificultad de garantizar la reproducibilidad de los experimentos. Aunque existen diversos simuladores que reducen la necesidad de infraestructura real, cada uno emplea formatos de entrada, procesos de configuración y mecanismos de recolección de métricas distintos, lo que dificulta su uso sistemático. En este trabajo se presenta *k8s-sims*, un framework modular y extensible de benchmarking para simuladores de Kubernetes que automatiza el ciclo completo de experimentación: generación de cargas de trabajo adaptadas a cada simulador, ejecución controlada de las simulaciones, recolección de métricas y visualización de resultados. Para validar el framework, se ha ejecutado un caso de uso sobre cinco simuladores empleando trazas reales de producción del programa Alibaba Cluster Trace [1]. Los resultados demuestran que *k8s-sims* permite realizar experimentos reproducibles sobre múltiples simuladores con un esfuerzo mínimo de configuración. El código fuente se encuentra disponible como proyecto de código abierto.**

Palabras clave— **Kubernetes, simuladores, benchmarking, contenedores, framework, automatización, reproducibilidad, k8s-sims.**

I. Introducción

EX perimentar con clústeres de Kubernetes para investigación supone un desafío considerable, debido a la inversión de tiempo y recursos que requiere la configuración de la infraestructura, la ejecución de cargas de trabajo y la posterior recolección y análisis de resultados. A esto se suma la dificultad de garantizar la reproducibilidad de los experimentos, aspecto fundamental en cualquier proceso de investigación.

Aunque existen diversos simuladores de Kubernetes que reducen la necesidad de desplegar infraestructura real, cada uno emplea formatos de entrada, procesos de configuración y mecanismos de recolección de métricas distintos. Esta falta de estandarización dificulta la comparación objetiva entre simuladores y obliga al investigador a invertir un esfuerzo significativo en tareas de adaptación manual.

Para abordar estos problemas, se ha desarrollado *k8s-sims*, un framework modular y extensible de benchmarking para simuladores de Kubernetes que automatiza el ciclo completo de experimentación, desde la generación de cargas de trabajo hasta la visualización de resultados, permitiendo realizar experimentos reproducibles con un esfuerzo mínimo de configuración.

El resto del artículo se organiza de la siguiente manera: la Sección II presenta el contexto tecnológico y los simuladores existentes, la Sección III describe la arquitectura del framework *k8s-sims*, la Sección IV muestra un caso de uso que valida el funcionamiento del framework y, por último, la Sección V presenta las conclusiones y líneas de trabajo futuro.

[1]Departamento de Ingeniera y Diseño Digital, Universidad de Lleida, e-mail: mmj14@alumnes.udl.cat.

II. Estado del Arte

En las arquitecturas de software modernas, los contenedores constituyen el pilar fundamental sobre el que se sustenta el paradigma de microservicios. Un contenedor encapsula una aplicación junto con todas sus dependencias (bibliotecas, configuraciones y entorno de ejecución), lo que garantiza la portabilidad y la reproducibilidad del software independientemente de la infraestructura subyacente [2].

No obstante, a medida que el número de contenedores crece, su gestión manual se vuelve inviable. Coordinar el despliegue, el escalado, la recuperación ante fallos y la comunicación entre cientos o miles de contenedores requiere herramientas de orquestación automatizada. En este contexto, Kubernetes [3], [4] se ha consolidado como el estándar *de facto* en la industria. Se trata de una plataforma de código abierto que permite administrar cargas de trabajo contenerizadas mediante una configuración declarativa, proporcionando mecanismos de autoescalado, balanceo de carga, descubrimiento de servicios y autoreparación de forma nativa.

Sin embargo, cuando se desea desarrollar nuevos algoritmos de planificación o experimentar con clústeres para observar cómo reaccionan ante determinadas cargas de trabajo, el proceso se vuelve costoso: es necesario montar un clúster completo y desplegar toda la infraestructura antes de poder llevar a cabo el experimento. Esto supone una inversión de tiempo y dinero de la que no siempre se dispone. Afortunadamente, existen simuladores de Kubernetes que permiten realizar estos experimentos en un entorno controlado y por una fracción del coste. Cada simulador se centra en un aspecto concreto de Kubernetes. Entre los más relevantes se encuentran los siguientes.

Kubemark [5] es un simulador integrado en el propio repositorio de Kubernetes que permite crear nodos *hollow*, los cuales emulan el comportamiento de nodos reales sin llegar a ejecutar las cargas de trabajo. KWOK [6] adopta un enfoque más ligero: simula el ciclo de vida de los nodos y los *pods* directamente en el plano de control, lo que permite escalar a miles de nodos con un consumo mínimo de recursos. OpenSimulator [7] es un simulador independiente desarrollado por Alibaba que se ejecuta como un binario autónomo sin dependencias externas, utilizando la implementación *fake* de la clase `ClientSet` de la biblioteca de Kubernetes para Go [8] para represen-

tar todos los objetos en memoria. SimKube [9] es un simulador de tipo *record-and-replay* construido sobre KWOK que permite grabar trazas de un clúster en producción y reproducirlas en un entorno simulado. Por último, Kubernetes Scheduler Simulator [10] es un proyecto de Kubernetes SIGs enfocado en inspeccionar las decisiones del planificador, desplegándose sobre KWOK y añadiendo un *wrapper* que registra cada paso del proceso de planificación.

Más allá de los simuladores individuales descritos, en la literatura existen diversas propuestas que abordan la experimentación con entornos Kubernetes y *cloud-native* desde perspectivas complementarias.

En el ámbito de la simulación, K8sSim [11] es un simulador orientado a la evaluación de algoritmos de planificación de Kubernetes y Volcano que, al igual que el presente trabajo, utiliza trazas reales de Alibaba para generar cargas de trabajo representativas. No obstante, K8sSim se centra en la optimización del *scheduling* dentro de un único entorno de simulación propio y, cabe destacar que su código fuente para Kubernetes ha dejado de estar disponible públicamente, lo que limita su reproducibilidad. CloudNativeSim [12] propone un *toolkit* de simulación para aplicaciones *cloud-native* con arquitectura de microservicios, extendiendo CloudSim con interfaces de políticas de planificación y modelado de aplicaciones basadas en microservicios. Straesser et al. [13] plantean un enfoque denominado *Kubernetes-in-the-Loop* que integra el plano de control real de Kubernetes en un entorno de simulación ofreciendo mayor fidelidad.

En el ámbito de la emulación y la experimentación sobre infraestructura real o virtualizada, Q8S [14] propone la emulación de clústeres Kubernetes heterogéneos (x86_64/ARM64) mediante QEMU sobre OpenStack, argumentando que las simulaciones como CloudSim o K8sSim ofrecen un nivel de detalle limitado frente a la emulación. iDynamics [15] ofrece un *framework* configurable para emular dinámicas *cloud-edge* ejecutando microservicios reales sobre Kubernetes e inyectando condiciones de red controladas. Estas propuestas comparten con k8s-sims la motivación de reducir la barrera de entrada a la experimentación, pero se centran en la ejecución de cargas reales o emuladas, no en la automatización del proceso de experimentación sobre los propios simuladores.

Morrison [16] argumenta que la simulación es una herramienta infrautilizada en el ámbito de los sistemas distribuidos y demuestra su utilidad mediante SimKube, un simulador construido sobre KWOK. Su trabajo evidencia tanto el potencial de la simulación como la necesidad de apoyarse en múltiples herramientas (KWOK, kind, controladores específicos) para construir un entorno experimental completo.

No obstante, la diversidad de simuladores existentes (KWOK, Kubemark, SimKube, OpenSimulator, entre otros) supone en la práctica un obstáculo para el investigador que necesita seleccionar el más adecuado como base para sus algoritmos, ya que cada uno impone formatos de entrada, procesos de configuración y mecanismos de recogida de métricas distintos. Ninguna de las propuestas mencionadas aborda este problema de forma integral. k8s-sims nace precisamente de la necesidad de automatizar el ciclo completo de experimentación sobre múltiples simuladores de Kubernetes, desde la generación de cargas de trabajo hasta la visualización de resultados, proporcionando una capa de estandarización que permite al investigador centrarse en el análisis de los resultados en lugar de en las particularidades de cada simulador.

III. Arquitectura del framework

El framework *k8s-sims* automatiza el ciclo completo de experimentación para los simuladores soportados (Kubemark, KWOK, SimKube, OpenSimulator y Kubernetes Scheduler Simulator), ofreciendo además un sistema modular que permite incorporar nuevos simuladores con un esfuerzo mínimo de desarrollo. El toolkit se distribuye como una aplicación contenerizada mediante Docker, lo que elimina las dependencias con el sistema anfitrión. La Figura 1 muestra el flujo general del framework. El proceso parte de manifiestos YAML genéricos que definen los nodos y pods del clúster. KubeGen toma estos ficheros y genera los datasets adaptados al formato específico de cada simulador. A continuación, el usuario configura la lista de simuladores a evaluar y lanza la experimentación mediante KubeDirector, que se encarga de preparar la configuración necesaria para cada simulador. Para cada ejecución, KubeDirector invoca KubeRun, que despliega las cargas de trabajo, recopila métricas de rendimiento y limpia el entorno al finalizar. Los resultados se almacenan en ficheros CSV que KubePlot utiliza para generar gráficos comparativos de forma automática.

A. KubeGen: generación de datasets

Uno de los principales obstáculos detectados durante la experimentación es que cada simulador requiere un formato de entrada distinto para definir las cargas de trabajo. KubeGen resuelve este problema tomando como entrada dos manifiestos Kubernetes genéricos (uno de nodos y otro de pods) y generando automáticamente los ficheros en el formato específico de cada simulador. Esto elimina la necesidad de editar manualmente ficheros que, en cargas de trabajo reales, pueden contener miles de definiciones.

Además, KubeGen incorpora un mecanismo de partición incremental: divide un dataset grande en subconjuntos de tamaño creciente (por ejemplo, 100, 200, 300 nodos), de modo que cada partición es un superconjunto de la anterior. Esto permite estudiar cómo escala cada simulador sin necesidad de regenerar los datos desde cero, ya que los elementos de iteraciones anteriores no necesitan ser recalculados. Para maximizar la carga en cada partición, KubeGen emplea un pseudo planificador basado en un algoritmo voraz que, para cada nodo seleccionado, asigna pods mientras los recursos disponibles (CPU y memoria) sean suficientes para satisfacer las peticiones

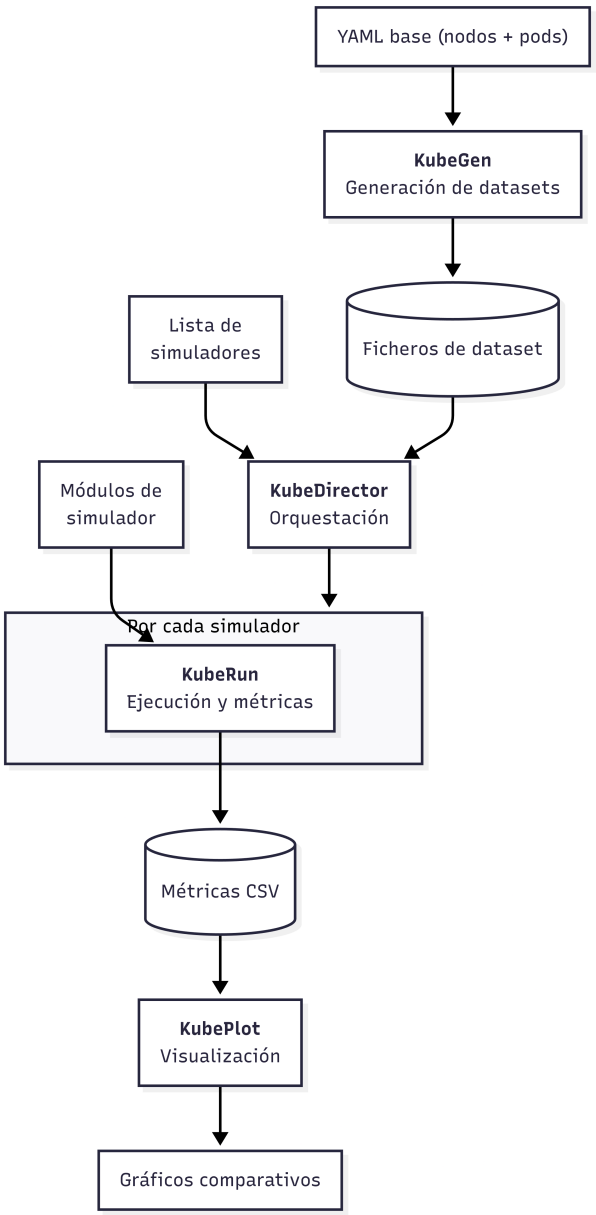


Fig. 1: Flujo general del framework *k8s-sims*.

del siguiente pod en la cola. De este modo, cada partición contiene una aproximación de la máxima carga de trabajo posible para el conjunto de nodos definido, lo que garantiza que los simuladores se evalúan en condiciones de saturación progresiva.

B. KubeRun: ejecución y recolección de métricas

Uno de los mayores obstáculos a la hora de experimentar con simuladores de Kubernetes es la cantidad de acciones que requieren la atención del usuario y que, en su mayoría, son procesos repetitivos susceptibles de ser automatizados: crear el clúster, desplegar las cargas de trabajo, monitorizar la ejecución, recopilar resultados y preparar el entorno para la siguiente ejecución. El problema se agrava cuando este ciclo debe repetirse decenas de veces con distintas configuraciones y simuladores, ya que el esfuerzo manual se convierte en un cuello de botella que compromete tanto la productividad de la investigación como la fiabilidad de los resultados.

KubeRun nace precisamente para eliminar este cuello de botella. Constituye el núcleo del framework y gestiona el ciclo completo de cada experimento de forma totalmente autónoma, sin requerir intervención del usuario en ninguna fase. Como muestra la Figura 2, KubeRun comienza parseando los argumentos de entrada y cargando el módulo correspondiente al simulador seleccionado. Dicho módulo sobrescribe las funciones genéricas del núcleo con la lógica específica del simulador. A continuación, el bucle principal de ejecución se encarga de crear el clúster simulado, desplegar los nodos y pods definidos en el dataset y monitorizar el consumo de recursos en tiempo real a través de la interfaz del simulador y del sistema de recolección de métricas. Al finalizar, registra los resultados en un fichero CSV y limpia el entorno, dejándolo preparado para la siguiente ejecución. Además, incorpora mecanismos de seguridad que permiten establecer límites de tiempo y memoria, deteniendo automáticamente cualquier simulación que supere los umbrales configurados.

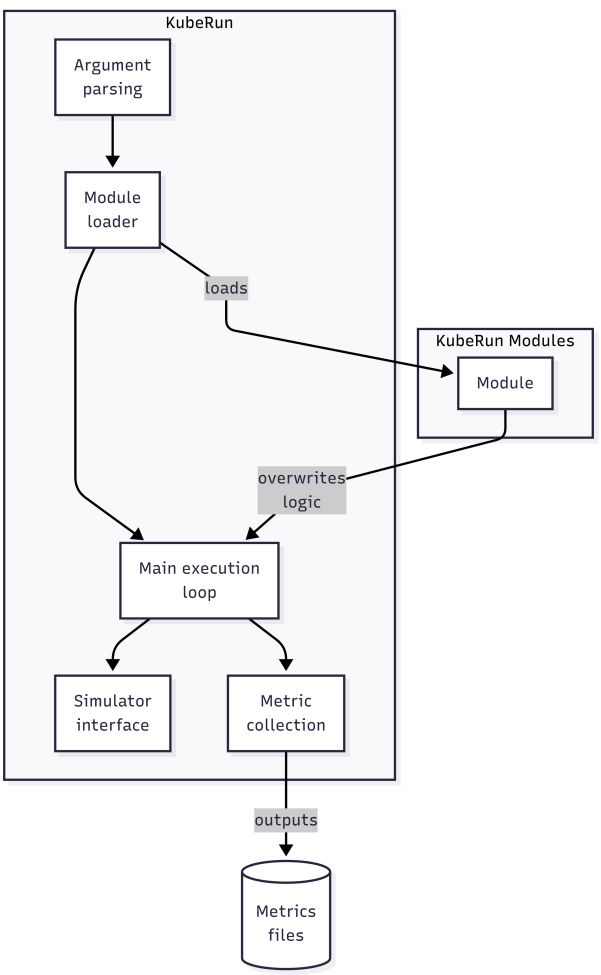


Fig. 2: Arquitectura de KubeRun. El módulo del simulador sobrescribe la lógica genérica del núcleo, permitiendo añadir nuevos simuladores sin modificar el framework.

Este diseño modular es un aspecto clave de KubeRun. Incorporar soporte para un nuevo simulador se reduce a implementar únicamente las funciones propias de dicho simulador, sin modificar el núcleo del framework. De este modo, KubeRun no solo resuelve el problema actual, sino que sienta las bases para futuras ampliaciones conforme aparezcan nuevos si-

muladores en el ecosistema.

El Listing 1 muestra la estructura mínima de un módulo. Cada módulo se ubica en un directorio y es cargado dinámicamente por KubeRun. Las funciones cubren dos aspectos: el ciclo de vida del clúster (creación, configuración, despliegue y limpieza) y la recolección de métricas. El módulo también puede sobrescribir variables globales del núcleo para adaptar el comportamiento del framework. Aquellas funciones cuyo comportamiento por defecto sea adecuado pueden omitirse, ya que KubeRun conserva su implementación genérica. Para activar el nuevo módulo, basta con añadir su nombre al fichero `SIM_MODULES`.

```
#!/bin/bash
# modules/<nuevo-simulador>/module.sh

# Variables globales que el modulo puede sobrescribir
# CLUSTER_NAME, CONTAINERS_TO_WATCH,
# FILE_PATTERN, NAMESPACE

# Ciclo de vida del cluster
create_cluster()    { :; } # Crear el cluster
cluster_setup()     { :; } # Configuracion post-creacion
deploy_objects()    { :; } # Desplegar nodos ($1) y pods ($2)
cleanup_cluster()   { :; } # Destruir cluster y limpiar

# Recoleccion de metricas
get_container_ids() { :; } # IDs de contenedores
get_memory_usage()  { :; } # Uso de memoria en bytes
get_cpu_usage()     { :; } # CPU total, user y system
metric_collector()  { :; } # Delega segun $1 (memory|cpu)
```

Listing 1: Plantilla de módulo.

C. KubeDirector: orquestación de experimentos

Ejecutar un benchmark sobre múltiples simuladores de forma consecutiva es un proceso que, realizado manualmente, resulta tedioso y propenso a errores: cada simulador tiene sus propias dependencias, su propio formato de dataset y sus propios requisitos de entorno. Una configuración incorrecta o una limpieza incompleta entre ejecuciones puede contaminar los resultados del siguiente simulador, invalidando todo el experimento.

KubeDirector resuelve este problema actuando como orquestador central del proceso de experimentación. Como se observa en la Figura 3, el proceso comienza cargando la lista de simuladores activos desde un fichero de configuración. A continuación, para cada módulo, KubeDirector localiza los datasets generados por KubeGen e invoca KubeRun con la configuración adecuada, gestionando el ciclo de setup y cleanup entre ejecuciones. Además, incorpora un sistema de limpieza que se activa automáticamente si la experimentación se interrumpe de forma inesperada, garantizando la consistencia del entorno.

El resultado es que un investigador puede lanzar una batería completa de experimentos con una única instrucción y obtener los resultados de todos los simuladores sin haber intervenido en ningún momento del proceso. Esto asegura que todos los simuladores se evalúan bajo las mismas condiciones, garantizando la reproducibilidad de los resultados.

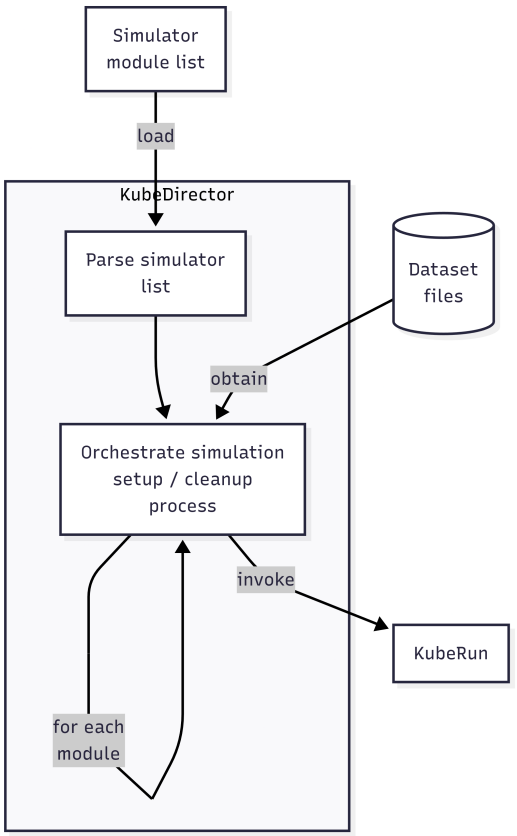


Fig. 3: Arquitectura de KubeDirector. Lee la lista de simuladores, obtiene los datasets y orquesta la ejecución secuencial de KubeRun para cada módulo.

D. KubePlot: visualización de resultados

Una vez finalizados los experimentos, el investigador se enfrenta a ficheros CSV con miles de registros que deben ser interpretados y comparados entre simuladores. Realizar este análisis de forma manual no solo consume tiempo, sino que es propenso a errores de interpretación que pueden llevar a conclusiones incorrectas.

KubePlot automatiza este proceso tomando como entrada los ficheros CSV generados por KubeRun y produciendo gráficos comparativos de forma inmediata.

De este modo, el investigador obtiene una visión clara y visual del comportamiento de cada simulador sin necesidad de manipular los datos en bruto, pudiendo identificar tendencias, cuellos de botella y diferencias de rendimiento de un solo vistazo.

El toolkit se distribuye como una aplicación contenerizada mediante Docker [2], lo que elimina las dependencias con el sistema anfitrión y resuelve los problemas de permisos y compatibilidad. De este modo, reproducir la batería completa de experimentos se reduce a ejecutar un único comando, independientemente del sistema operativo o la configuración del entorno.

IV. Caso de uso

Para validar el correcto funcionamiento del framework, se ha ejecutado un benchmark completo sobre cinco simuladores de Kubernetes. La Tabla I resume sus características principales. Los cinco simuladores, descritos en la Sección II, abarcan desde enfoques ligeros como KWOK hasta simulaciones de mayor fidelidad como Kubemark, lo que permite evaluar el framework en un rango amplio de niveles de abstracción y consumo de recursos.

Tabla I: Simuladores de Kubernetes evaluados.

Simulador	Base	Enfoque	Recursos
KWOK [6]	—	Nodos/pods fake	Bajo
Kubemark [5]	—	Nodos hollow	Alto
OpenSimulator [7]	—	Fake clientset	Bajo
SimKube [9]	KWOK	Record-and-replay	Medio
KSS [10]	KWOK	Depurar scheduler	Medio

A. Generación del dataset

Las trazas utilizadas provienen del programa Alibaba Cluster Trace [1] y contienen información sobre nodos y pods desplegados en clústeres reales de Alibaba, lo que permite reproducir escenarios de producción en los simuladores. A partir de estas trazas, KubeGen genera los ficheros de entrada en el formato específico de cada simulador, eliminando la necesidad de adaptar manualmente las cargas de trabajo.

Además, KubeGen particiona el dataset original en subconjuntos incrementales: cada partición contiene todos los elementos de la anterior más un incremento adicional de nodos y pods. De este modo se puede observar cómo evoluciona el comportamiento del simulador a medida que crece la carga, partiendo de un escenario base y saturándolo progresivamente. En concreto, para este caso de uso se han generado particiones en incrementos de 100 nodos, desde 100 hasta 1 213 nodos con un máximo de 7 064 pods. Este enfoque incremental también reduce el coste de generación, ya que los elementos de iteraciones anteriores no necesitan ser recalculados.

B. Ejecución de los experimentos

Los experimentos se han ejecutado sobre un servidor con 2×Intel Xeon Silver 4210R (40 núcleos) a 3,20 GHz y 128 GB de memoria DDR4.

Una vez generados los datasets, el usuario configura los simuladores que desea evaluar. A partir de esta configuración, KubeDirector lee la lista de simuladores activos y, para cada uno, invoca KubeRun con el dataset correspondiente.

KubeRun gestiona el ciclo completo de cada experimento de forma autónoma. Prepara el entorno con los requisitos específicos del simulador seleccionado, despliega los nodos y pods definidos en el dataset y monitoriza la ejecución mediante *cgroups* de Linux [17]. Una vez que todos los pods han sido planificados o se alcanza un límite de tiempo o de memoria previamente establecido, KubeRun registra las métricas recopiladas en un fichero CSV y procede a la limpieza del entorno, dejándolo preparado para la siguiente ejecución. Para garantizar la reproducibilidad, cada configuración se ha ejecutado tres veces.

Cabe señalar que el coste computacional del propio framework es despreciable frente al de las simulaciones. KubeGen realiza operaciones de transformación de texto y KubePlot procesa ficheros CSV para producir gráficos, tareas cuya ejecución se completa en el orden de segundos incluso para el dataset completo de 1 213 nodos y 7 064 *pods*. En comparación, los tiempos medios de CPU de los simuladores oscilan entre los 193,08 s de KWOK y los 7 439,00 s de Kubemark (véase Tabla II). De este modo, el *overhead* introducido por k8s-sims se limita a las fases de preparación y visualización, sin impacto significativo en el tiempo total de experimentación, que está dominado por la ejecución de los propios simuladores.

C. Visualización de resultados

Tras la ejecución, KubePlot toma los ficheros CSV generados por KubeRun y produce gráficos comparativos de forma automática. La Tabla II resume las métricas clave obtenidas durante las simulaciones. Es importante destacar que estas métricas reflejan el consumo de recursos del propio simulador, no de la orquestación de contenedores que se está simulando.

Tabla II: Resultados promedio de los simuladores evaluados. En **negrita** el mejor valor y subrayado el peor para cada métrica.

Métrica	OSim	KWOK	SKube	KSS	KMark
CPU min (s)	**6,33**	38,33	174,67	64,33	317,00
CPU media (s)	260,47	**193,08**	698,14	825,67	7 439,00
CPU max (s)	631,67	**293,67**	1 492,67	1 244,00	14 387,33
RAM min (GB)	**0,11**	0,52	5,96	0,90	9,08
RAM media (GB)	2,62	**1,15**	8,69	5,01	15,54
RAM max (GB)	6,30	**1,51**	11,20	8,28	18,47

OSim: OpenSimulator. SKube: SimKube. KSS: Kubernetes Scheduler Simulator. KMark: Kubemark.

Los resultados muestran cómo se comportan los simuladores a la hora de simular cargas de trabajo. KWOK es el simulador con menor huella de recursos, manteniendo un consumo de memoria por debajo de 1,51 GB y un tiempo medio de CPU de 193,08 s incluso en las configuraciones más exigentes. La segunda opción más ligera es OpenSimulator, que destaca especialmente en escenarios pequeños con un tiempo mínimo de CPU de apenas 6,33 s y un pico de memoria de 0,11 GB, aunque su consumo crece de forma más pronunciada a medida que aumenta la carga, alcanzando los 6,30 GB y 631,67 s en las configuraciones más grandes.

En el extremo opuesto, Kubemark obtiene el peor rendimiento en todas las métricas, con picos de memoria de hasta 18,47 GB y tiempos de CPU que alcanzan las 14 387,33 s. Esto evidencia que un mayor grado de realismo en la simulación conlleva un coste significativo en recursos, ya que replicar los componentes internos del nodo (kubelet y kube-proxy) introduce una sobrecarga considerable cuando se trabaja con miles de pods.

También podemos observar que SimKube y Kubernetes Scheduler Simulator ofrecen un consumo moderado de recursos, con picos de memoria de

11,20 GB y 8,28 GB respectivamente. Esto se debe a que ambos ejecutan KWOK internamente y añaden capas de funcionalidad sobre esta base, lo que incrementa el consumo respecto a KWOK pero se mantiene lejos de los valores de Kubemark.

La Figura 4 muestra la evolución del pico de memoria a medida que crece el número de nodos, y la Figura 5 presenta la tasa de planificación para cada simulador. Estas visualizaciones permiten identificar rápidamente tendencias y cuellos de botella sin necesidad de analizar los datos en bruto.

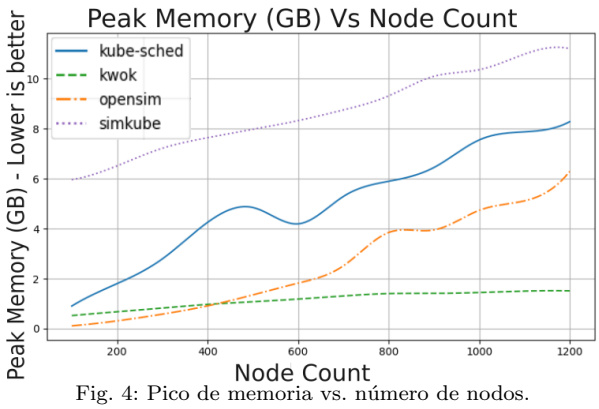


Fig. 4: Pico de memoria vs. número de nodos.

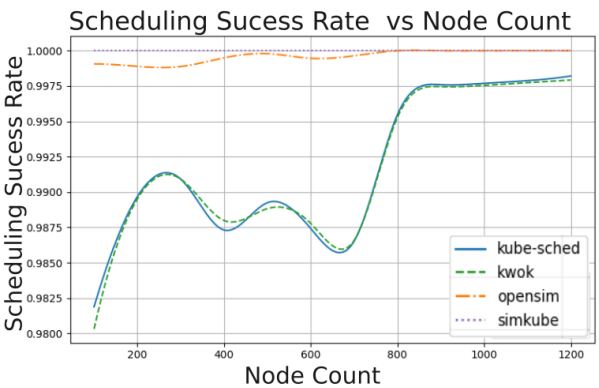


Fig. 5: Tasa de planificación vs. número de nodos.

El caso de uso demuestra que *k8s-sims* permite ejecutar un benchmark completo sobre múltiples simuladores de forma automatizada y reproducible, abstrayendo las particularidades de cada uno y reduciendo el esfuerzo manual a la configuración inicial.

V. Conclusiones y trabajo futuro

En este artículo se ha presentado *k8s-sims*, un framework modular y extensible para la automatización de experimentos sobre simuladores de Kubernetes. El toolkit aborda los principales problemas que dificultan la experimentación en este ámbito: la heterogeneidad de formatos entre simuladores, el carácter manual y repetitivo del proceso de experimentación, y la dificultad de recopilar y visualizar métricas de forma sistemática.

Para dar respuesta a cada uno de estos problemas se ha desarrollado un conjunto de herramientas que cubren el ciclo completo de experimentación, desde la generación de cargas de trabajo hasta la visualización de resultados. El diseño modular del framework permite incorporar nuevos simuladores con facilidad, sin necesidad de modificar el núcleo de la aplicación.

Como validación, se ha ejecutado un caso de uso sobre cinco simuladores utilizando trazas reales de producción. Los resultados generados por el framework han permitido identificar diferencias significativas en el consumo de recursos y en la capacidad de planificación de los simuladores evaluados, demostrando la utilidad de *k8s-sims* como herramienta de apoyo a la investigación en este campo.

El código fuente se encuentra disponible como proyecto de código abierto [18], con el objetivo de facilitar su adopción y extensión por parte de la comunidad investigadora.

Como trabajo futuro se contemplan varias líneas de mejora. En primer lugar, la ampliación de KubeGen para soportar tipos de despliegue adicionales de Kubernetes, como *Deployments* y *StatefulSets*, así como la generación e integración de trazas procedentes de otros proveedores cloud. En segundo lugar, la incorporación de sistemas de monitorización como Prometheus para enriquecer la recolección de métricas durante las simulaciones. Por último, la mejora del sistema de módulos y del proceso de conversión de formatos, con el objetivo de simplificar la incorporación de nuevos simuladores al framework.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades (MICIU) AEI/10.13039/501100011033 bajo el contrato PID2023-146193OB-I00.

Referencias

[1] Alibaba, "Kubernetes cluster traces," `https://github.com/alibaba/clusterdata`.
[2] Docker, "What is a container?," `https://www.docker.com/resources/what-container/`.
[3] C. Carrión, "Kubernetes as a standard container orchestrator – a bibliometric analysis," *Journal of Grid Computing*, vol. 20, no. 4, pp. 42, 2022.
[4] Kubernetes, "Kubernetes documentation – overview," `https://kubernetes.io/docs/concepts/overview/`.
[5] Kubernetes SIGs, "Kubemark," `https://github.com/kubernetes/kubernetes/tree/master/test/kubemark`.
[6] Kubernetes SIGs, "Kubernetes without kubelet," `https://kwok.sigs.k8s.io/`.
[7] Alibaba, "Opensimulator: Kubernetes cluster simulator," `https://github.com/alibaba/open-simulator`.
[8] Kubernetes, "fake – client-go/kubernetes/fake," `https://pkg.go.dev/k8s.io/client-go/kubernetes/fake`.
[9] SimKube, "Simkube documentation," `https://simkube.dev/documentation/`.
[10] Kubernetes SIGs, "Kubernetes scheduler simulator," `https://github.com/kubernetes-sigs/kube-scheduler-simulator`.
[11] Shilin Wen, Rui Han, Ke Qiu, Xiaoxin Ma, Zeqing Li, Hongjie Deng, and Chi Harold Liu, "K8ssim: A simulation tool for kubernetes schedulers and its applications in scheduling algorithm optimization," *Micromachines*, vol. 14, no. 3, 2023.
[12] Jingfeng Wu, Minxian Xu, Yiyuan He, Kejiang Ye, and Chengzhong Xu, "Cloudnativesim: A toolkit for modeling and simulation of cloud-native applications," *Software: Practice and Experience*, vol. 55, no. 7, pp. 1185–1208, 2025.
[13] Martin Straesser, Patrick Haas, Sebastian Frank, Alireza Hakamian, André van Hoorn, and Samuel Kounev, "Kubernetes-in-the-loop: Enriching microservice simulation through authentic container orchestration," in *Performance Evaluation Methodologies and Tools*, Evangelia

Kalyvianaki and Marco Paolieri, Eds., Cham, 2024, pp. 82–98, Springer Nature Switzerland.

[14] Jonathan Decker, Vincent Florens Hasse, and Julian Kunkel, "Q8s: Emulation of heterogeneous kubernetes clusters using qemu," *Algorithms*, vol. 18, no. 6, 2025.

[15] Ming Chen, Muhammed Tawfiqul Islam, Maria Rodriguez Read, and Rajkumar Buyya, "idynamics: A configurable emulation framework for evaluating microservice scheduling policies under controllable cloud-edge dynamics," 2025.

[16] David R. Morrison, "Simulation: An underutilized tool in distributed systems," *ACM Queue*, vol. 22, no. 6, pp. 40–59, 2024, `https://queue.acm.org/detail.cfm?id=3711677`.

[17] Linux man-pages project, "cgroups(7) – linux manual page," `https://man7.org/linux/man-pages/man7/cgroups.7.html`.

[18] Matias Medina, Vitor Da Silva, Rosana Tomas, Fernando Guirado, and Josep Lluís Lérida, "k8s-sims: Benchmarking toolkit for kubernetes simulators," `https://github.com/TheSmuks/k8s-sims`, 2025.

HDCLite: A Compact Encoding Scheme for Memory-Efficient HyperDimensional Computing

Víctor Ortega, Soledad Escolar, Fernando Rincón, Jesús Barba, Julián Caba and Juan Carlos López[1]

Abstract— **Hyperdimensional Computing has demonstrated its effectiveness for anomaly classification, addressing several limitations of traditional Machine Learning-based approaches. However, achieving high classification accuracy typically requires processing large volumes of training data to learn normal behavior, which can hinder its deployment in resource-constrained embedded systems. To address this limitation, we propose HDCLite, a memory-efficient encoding scheme that eliminates the need to store level hypervectors in memory, which are required to construct the class hypervector further used for sample classification. Instead, HDCLite stores a compact representation of the level hypervectors along with a seed that enables the generation of the remaining hypervectors on-the-fly. The proposed approach is evaluated through a real-world case study based on data collected from industrial emergency lighting devices. We assess performance in terms of accuracy, memory footprint, and execution time. Results show that HDCLite achieves up to a 142,151× reduction in memory usage compared to a non-optimized encoding strategy, while maintaining comparable classification accuracy.**

Keywords— **Hyperdimensional Computing; Anomaly Detection; Predictive Maintenance; Internet of Things; Machine Learning.**

I. Introduction

INTERNET of Things (IoT) has significantly transformed industrial environments, enabling enhanced monitoring and automation capabilities and fostering the concept of Industry 4.0. However, the increasing connectivity and complexity of IoT devices also introduce a higher risk of unexpected failures, which may lead to significant economic losses and operational disruptions. In this context, the early detection of anomalies has become a key enabler for predictive maintenance strategies. By identifying deviations from normal behaviour at an early stage, it is possible to anticipate potential faults, reduce unplanned downtime, and minimize maintenance costs. Consequently, anomaly detection plays a crucial role in improving the reliability, efficiency, and safety of modern industrial systems.

An anomaly can be defined as a pattern or a set of observations in the data that does not conform to the notion of *normal* behaviour [1]. Typically, anomalies correspond to data points or regions that significantly deviate from those where the majority of the data is concentrated. Such deviations may arise from various causes, including malicious activities, system failures, or unusual behaviours, and are therefore of particular interest to analysts due to their potential relevance within the context of the problem under study. Anomaly detection has been widely explored in the literature using a variety of approaches [2], including statistical methods, clustering techniques, nearest neighbor-based approaches, information-theoretic models, spectral methods, and Bayesian networks [3]. Among these, Machine Learning (ML)-based techniques have received significant attention due to their ability to model complex patterns in data [4].

Hyperdimensional Computing (HDC) [5] has emerged as a promising alternative, demonstrating strong performance in classification tasks [6]. Although HDC models typically require a smaller memory footprint than dense neural networks, effective classification still demands the processing of large volumes of data to accurately capture normal behaviour and subsequently detect anomalies. This requirement poses significant challenges for the deployment of HDC in resource-constrained embedded systems and remains an important limitation for its widespread adoption. With this motivation, we propose HDCLite, a lightweight encoding scheme based on a quantization approach that drastically reduces the memory footprint in the encoding stage of a typical HDC pipeline. HDCLite achieves this by storing only a minimal base hypervector for each feature, from which all level hypervectors are generated on-the-fly, thereby avoiding the need to store the full set of hypervectors in memory.

The remainder of this paper is organized as follows. Section II reviews the existing literature and Section III summarizes the fundamental operations of HDC. In Section IV we present the system model and in Section V we describe the HDCLite implementation. Section VI evaluates our approach through one real case study and finally, in Section VII we summarize the main conclusions of this work and outlines directions for future research.

II. Related Works

HDC is a computational paradigm inspired by the human brain and designed to address classification problems by mimicking aspects of cognitive processing. In HDC, data are encoded as high-dimensional vectors, also known as hypervectors, which are capable of encoding, storing, and manipulating infor-

[1]School of Computer Science, University of Castilla-La Mancha, Ciudad Real, Spain. Corresponding author's e-mail: victor.ortegagomez@uclm.es

mation in a manner analogous to distributed representations in the human brain. In contrast to conventional ML-based approaches, HDC enables one-shot and computationally lightweight learning, leading to energy-efficient implementations that are particularly appealing for resource-constrained embedded systems. The high dimensionality of hypervectors provides inherent robustness to noise, as errors tend to be distributed across dimensions rather than concentrated in specific components. Additionally, the limited interpretability often associated with traditional ML models can be alleviated in HDC, as its prototype-based representation facilitates a more intuitive understanding of the relationships between inputs and outputs [7].

The suitability of HDC for anomaly detection has been demonstrated in a growing body of recent work. For instance, the ODHD framework proposed in [8] adopts a Positive-Unlabeled (P–U) learning paradigm, where the model is trained exclusively on normal samples and evaluated on unlabeled data that may contain anomalies. This approach outperforms classical methods such as One-Class SVM, Isolation Forest, and autoencoders across multiple datasets and evaluation metrics, including accuracy, F1-score, and ROC-AUC. HDC has also been successfully applied to anomaly detection in dynamic network environments. In [9], network states are modeled as sequences of directed and weighted graphs, where nodes and edges are encoded as hypervectors. Anomalies are identified through a scoring mechanism that captures both instantaneous deviations and historical behaviour, achieving high accuracy and computational efficiency on both synthetic and real-world datasets. Additional applications further highlight the versatility of HDC in this domain. The HDAD framework introduced in [10] targets anomaly detection in autonomous vehicle sensor data by encoding sensor readings into hypervectors and reconstructing the original feature space. Anomalies are detected based on the deviation between reconstructed and observed values, achieving near-perfect detection performance on the evaluated dataset. In the context of IoT security, HDC has been employed for network intrusion detection using the NSL-KDD dataset [11], reporting an accuracy of 91.55% on the KDDTrain+ subset and outperforming traditional machine learning approaches. In [12] is studied and demonstrated the effectiveness of using HDC in modeling physiological and behavioural patterns in wearable devices, where iterative learning achieves training times up to 35.8× faster than deep neural networks while maintaining comparable accuracy. In a previous work [13], we investigated the feasibility of applying HDC to anomaly classification in industrial emergency lighting systems. In contrast, this paper focuses on reducing the amount of data required during the quantization process in the encoding stage, without compromising classification accuracy. This is particularly relevant in scenarios with a large number of features and high quantization levels, where the memory requirements can become prohibitive and may negatively impact classification performance.

III. Background on HDC

In HDC each data point from the input space obtained from the real world is encoded as a high-dimensional vector, commonly referred to as a hypervector. These hypervectors typically have a dimensionality D in the order of thousands, with $D =$ 10,000 being a commonly adopted value. Hypervectors can represent different types of data. In general, they are classified according to the type of elements they contain, such as binary, bipolar, integer, real-valued, or character-based representations.

Let hv_1 and hv_2 be two bipolar hypervectors, i.e., $hv_i \in \{-1, 1\}^D$. These hypervectors can be combined using a set of fundamental operations [5]:

1. **Bundling ($\oplus$)** is used to combine multiple hypervectors into a single representative hypervector. It is typically implemented as a component-wise sum of the input hypervectors, i.e., $hv_1 \oplus hv_2 = hv_1[i] + hv_2[i]$, where $hv_1[i]$ and $hv_2[i]$ denote the i-th components of hv_1 and hv_2, respectively. The resulting hypervector may require normalization; common strategies include majority voting or random tie-breaking.
2. **Binding ($\otimes$)** is used to encode associations or relationships between hypervectors. It is implemented as a component-wise multiplication of the input hypervectors, producing a new hypervector that is approximately orthogonal to the original ones while remaining invertible under certain conditions. The binding operation between hv_1 and hv_2 is defined as: $hv_1 \otimes hv_2 = hv_1[i] \cdot hv_2[i]$, where $hv_1[i]$ and $hv_2[i]$ denote the i-th components of the hypervectors. For bipolar hypervectors, this operation is equivalent to the logical XOR operation when using binary representations.
3. **Permutation (Π)** is mainly used to encode sequential or positional information by reordering the components of a hypervector according to a fixed deterministic rule, such as a cyclic shift. This operation preserves the content of the hypervector while encoding contextual information. The permutation of a hypervector hv_1 produces a new hypervector defined as: $\Pi(hv_1) = [hv_1[\pi(i)]]_{i=1}^{D}$, where π is a bijective permutation function.
4. A **similarity function** is used to evaluate the degree of similarity between two hypervectors. One of the most commonly used measures is cosine similarity, defined as:

$$\cos(hv_1, hv_2) = \frac{hv_1 \cdot hv_2}{\|hv_1\| \, \|hv_2\|} \tag{1}$$

where $\cdot$ denotes the dot product and $\|hv_i\|$ represents the Euclidean norm of hv_i.

IV. System Model

Let us consider a set of n networked IoT devices $d_1, d_2, \ldots, d_n$, each acting as a monitoring device that observes environmental or operational conditions of the deployment scenario. These devices are equipped with dedicated hardware components (e.g., MCU, sensors, and battery) and communication capabilities that enable them to periodically transmit data to a special node d_0, referred to as the sink, which forwards the collected data to an external network for further analysis and processing. Data is relayed towards the sink through neighbouring nodes selected according to routing metrics such as link quality, forming a dynamic multi-hop communication path within the local network.

At each sampling period, each device d_i ($i \in [1, n]$) collects a set of k features $\mathcal{F}=\{f_1, f_2, \ldots, f_k\}$, representing observable parameters and status values that are unique to each device. These features may describe the device's measurements (e.g., identifiers, failure counts) or its internal state (e.g., next hop, travel time, link quality, battery level). The values that a feature takes may fluctuate within a normal range specific to the device (e.g., battery voltage in [2.5, 3.3] V). Note, however, that abnormal events such as hardware failures, network issues, or security incidents may produce atypical values or outliers, which differ slightly from the values considered normal. At each transmission period, the values collected for each feature f_j by each device d_i are first aggregated into a message and then transmitted to the sink d_0. In turn, the sink d_0 aggregates the messages that it receives from the devices into a single data space denoted as $\mathcal{S}$, which contains all observations transmitted by the n devices along the time. From this set may be easily derived the following subset of data:

- $S_i \in \mathcal{S}$ is the subset of the samples (i.e. values of the features) sent by d_i, i.e., $\mathcal{S}=\{S_1, S_2, \ldots, S_n\}$.
- $S_{i,j} \in S_i$ is the subset that contains the values of feature f_j sent by device d_i,
- $\overrightarrow{s_{i,x}}$ is a vector that contains the values of all features corresponding to the xth-sample of device d_i, i.e. $\overrightarrow{s_{i,x}} = [s^1_{i,x}, s^2_{i,x}, \ldots, s^k_{i,x}]$, with $1 \leq i \leq n$ and $1 \leq j \leq k$.

Each sample is encoded into a hypervector via a device-specific encoding function. Using the training set, the HDC framework derives a prototype hypervector representing the statistical regularities of normal device behaviour. The main objective of the proposed HDC framework is to classify, if a new sample constitutes either a normal or anomalous behaviour, by evaluating their distance to the learned signature. The following subsections describe in detail the proposed HDC model.

A. HDC pipeline

In the field of HDC, we understand a pipeline to be a data processing flow in which data is transformed, combined, and then compared for decision-making purposes. A typical HDC pipeline for classification consists of three following stages: encoding, training, and inference.

A.1 Encoding

We define an encoding function $\varphi : S_i \rightarrow \mathcal{HV}$, that maps each input vector $\overrightarrow{s_{i,x}}$, representing the xth-sample collected by device d_i, to a hypervector $hv_x \in \mathcal{HV}$, with $hv_x = \{-1, 1\}^D$. This encoding function φ must be carefully designed to generate hypervectors that preserve the correlation structure of the input data. In this work, we implement φ using a quantization-based approach [14], which discretizes the input space to generate consistent and meaningful hypervector representations.

We denote by $s^j_{i,\min}$ and $s^j_{i,\max} \in S_{i,j}$ the minimum and maximum values of feature f_j for device d_i, respectively. We consider a fixed number of quantization levels M, common to all features $f_j \in \mathcal{F}$ and devices. Therefore, the range of values for feature f_j in device d_i is given by $[s^j_{i,\min}, s^j_{i,\max}]$, while the number of samples lies in the interval $[0, |S_{i,j}|]$. Assuming a uniform distribution across quantization levels, the width of each quantization interval is defined as: $\sigma^j_i = s^j_{i,\max} - s^j_{i,\min}/M$.

The set of level hypervectors associated with feature f_j for device d_i is denoted as $Q^j_i = \{q^j_{i,0}, q^j_{i,1}, \ldots, q^j_{i,M-1}\}$, where $q^j_{i,m}$ represents the mth level hypervector. The first level hypervector $q^j_{i,0}$ is randomly initialized, while the remaining ones $\{q^j_{i,1}, \ldots, q^j_{i,M-1}\}$ are generated incrementally from their predecessors. In particular, each $q^j_{i,m}$ differs from $q^j_{i,m-1}$ ($1 \leq m < M$) by flipping exactly $b = \frac{D}{2(M-1)}$ components, ensuring a gradual and uniform increase in dissimilarity across levels. Note that, in doing so, we generate a matrix of size $M \times D$ for each feature and a three-dimensional tensor of size $k \times M \times D$ that stores the level hypervectors associated with device d_i. Then, we define the encoding function $\varphi : S_i \rightarrow \mathcal{Q}^j_i$ that returns, for each item of the xth sample given by the input vector $\overrightarrow{s_{i,x}}$, the level hypervector corresponding as:

$$\varphi(s^j_{i,x}) = \begin{cases} q^j_{i,0} & s^j_{i,x} = s^j_{i,\min} \\ q^j_{i,m} & \sigma^j_i m \leq s^j_{i,x} \leq \sigma^j_i(m+1), m \in [1, M) \\ q^j_{i,M-1} & s^j_{i,x} = s^j_{i,\max} \end{cases} \tag{2}$$

A bipolar hypervector of dimension D, $hv_x \in \mathcal{HV}$, with $hv_x = \{-1, 1\}^D$, exhibits two important characteristics that directly affect the resources requirements: 1) the density of the hypervector, denoted by ρ, is defined as the fraction of components equal to 1 with respect to the total dimension D, i.e. $\rho(hv_x) = \frac{|\{i | hv_x[i]=1\}|}{D}$; 2) the sparsity degree of the hypervector, denoted by Δ, describes how the components of the hypervector are distributed across its dimension, capturing the degree to which equal values tend to appear clustered or dispersed within the vector. To formalize this notion, we represent the

hypervector as a sequence of u contiguous blocks of identical values, each block having a size of v components, such that $u \times v = D$. The sparsity degree is therefore defined as $\Delta(hv_x) = \frac{D}{u} = v$, which corresponds to the length of the contiguous blocks of identical values in the hypervector. While the density $\rho(hv_x)$ determines the proportion of 1's in the vector, multiple combinations of u and v may satisfy the same density while producing very different structural patterns. Note that, a randomly generated hypervector, obeys to a density $\rho = 0.5$ and a configuration with $u = D$ and $v = 1$, corresponding to a highly alternating hypervector, where the values change at every dimension. In contrast, smaller values of u and, therefore, larger block sizes v produce hypervectors with longer contiguous segments of identical values.

A.2 Training

In the training stage, a class hypervector, also known as a prototype hypervector, is generated for each class used in the classification task. For our classification purposes, each class corresponds to the normal operating behaviour of each specific device d_i. Thus, we have a number of n classes used for classification, and a set of classes $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n\}$, where $\mathcal{C}_i$ is the class hypervector corresponding to d_i, with $i \in [1, n]$, that is computed in two steps: 1) the encoding of all sample input vectors $\overrightarrow{s_{i,x}}$ of d_i by using the φ function. For each item of the input vector $\overrightarrow{s_{i,x}} = [s^1_{i,x}, s^2_{i,x}, \ldots, s^k_{i,x}]$, the encoding function φ produces its corresponding hypervector, denoted as hv^x_i, by adding up their level hypervectors:

$$hv^x_i = \varphi(s^1_{i,x}) \oplus \varphi(s^2_{i,x}) \oplus \ldots \oplus \varphi(s^k_{i,x}) \forall x \in S_i \quad (3)$$

and 2) the bundling of the hypervectors obtained in the previous step as follows:

$$C_i = \bigoplus hv^x_i \quad \forall x \in S_i \quad (4)$$

where $\mathcal{C}_i$ is constructed by applying the bundling operation to the hypervectors of all samples in the data space associated with device d_i.

A.3 Inference

Let $\overrightarrow{s_{i,y}}$ be the vector corresponding to the sample y collected by device d_i defined as $\overrightarrow{s_{i,y}} = \{s^1_{i,y}, s^2_{i,y}, \ldots, s^k_{i,y}\}$ with $y \neq x \quad \forall x$. This means that, the vector of sample y, $\overrightarrow{s_{i,y}}$, has not been used in the computation of $\mathcal{C}_i$. We classify the sample $\overrightarrow{s_{i,y}}$ as an outlier of d_i if the cosine similarity function between its hypervector hv^y_i and the class hypervector $\mathcal{C}_i$, computed as:

$$cos(hv^y_i, C_i) = \frac{hv^y_i \times C_i}{\|hv^y_i\| \times \|C_i\|} \quad (5)$$

is below a certain threshold $\mathcal{B}_i$, which represents a decision boundary established for a specific device d_i to determine whether a new sample $\overrightarrow{s_{i,y}}$ corresponds to a normal behaviour or to an anomaly. It is worth noting that the smaller the difference between hv^y_i and $\mathcal{C}_i$, the higher the resulting cosine similarity. Moreover, the selection of the threshold value $\mathcal{B}_i$ is critical to system performance, as it directly influences the classification precision.

B. Threshold computation

Let $\mathcal{B}_i$ be the decision threshold for device d_i. According to [15] we compute it as follows:

$$B_i = \mu_i + 2\sigma_i \quad (6)$$

where μ_i is the average similarity between the samples used in the training process to build the prototypical hypervector $\mathcal{C}_i$, and σ_i is the standard deviation associated with these similarities. Note that, this computation captures the inherent variability of normal operation while remaining robust to noise and minor fluctuations in the input data.

V. HDCLite: A memory-efficient pipeline implementation

The quantization-based encoding approach described in Section IV is well suited for classification tasks where the input data are represented through features. However, as discussed, the memory required to store the level hypervectors grows linearly with D, M and k and it may become prohibitive for deployments on resource-constrained embedded systems. Motivated by this limitation, we propose HDCLite, an HDC framework that implements a memory-efficient quantization-based encoding scheme aimed at drastically reducing the memory footprint of the HDC pipeline. The key idea behind HDCLite is to exploit the structural sparsity of hypervectors to avoid storing the full set of level hypervectors. Instead, HDCLite stores the minimal representation that is required for deterministically generate M level hypervectors for each feature f_j consisting of: 1) a single base hypervector denoted as HV^j_{base} encoded as a byte pattern; and 2) a vector of seeds of length k, denoted as $\overrightarrow{seeds}$, where $seeds[j]$ corresponds to the seed of the feature f_j. In following subsections we describe in depth the processing details.

A. Memory-Efficient Encoding Scheme

HDCLite defines, for each feature f_j, a base hypervector denoted as HV^j_{base}, which constitutes the minimal representation required to generate its corresponding set of M level hypervectors. The base hypervector is designed to minimize memory usage, requiring only 1 byte, where each bit encodes a binary value (0 or 1). Each bit in this compact representation is expanded into a contiguous block of identical values, resulting in a full-dimensional hypervector of size $D = u \times v$, where $u = 8$ is the number of blocks (i.e., the number of bits in the base representation) and v is the block length. This construction ensures that all components within a block share the same value, while preserving a predefined density ρ of the resulting hypervector. Note that for generating bipolar hypervectors during the expansion we translate

the value as -1. Note also that this representation requires the constraint $D \bmod 8 = 0$ to hold. For instance, when $v = 256$, the resulting dimensionality is $D = 4,096$. Larger values of D can be obtained by increasing v, at the cost of a marginal increase in memory requirements.

Since unique level-0 hypervectors must be assigned to each feature, HDCLite ensures uniqueness by generating base hypervectors with different bit patterns. The number of distinct patterns to be generated must match the number of features k. To this end, we define w as the minimum number of bits required to guarantee uniqueness, such that $2^w \geq k$ or equivalently, $w \geq \lceil \log_2 k \rceil$. For example, when $k = 32$ features, at least $w = 5$ bits are required to ensure unique representations. This procedure is formalized in Algorithm 1.

Algorithm 1 Generation of u blocks of HV^j_{base}

```
function GenerateBlocks(u, ρ, w)
    repeat
        blocks ← 0
        count_ones ← 0
        for bit = 0 → u do
            r ← rand()
            if r ≥ ρ then
                blocks ← blocks | (1 ≪ bit)
                count_ones ← count_ones + 1
            end if
        end for
    until threshold ≤ count_ones ≤ w
    return blocks
end function
```

Once the u bits of the base hypervector HV^j_{base} of the feature f_j has been generated, we derivate from it its first level hypervector. This procedure consists of shuffling the previously generated blocks and verifying whether the resulting combination has already been produced, thereby ensuring that each first level hypervector remains unique. If the combination is indeed unique, the process continues with the generation of further combinations following the same procedure. After obtaining the required set of unique combinations, they are assigned to a vector that stores the first level hypervector. Each of these hypervectors is represented and stored using a single byte. This procedure can be seen in Algorithm 2.

Algorithm 2 Generation of first level hypervector

```
function GenerateFirstLevelHDVs(k)
    block_comb ← []
    for i = 0 → k − 1 do
        repeat
            duplicate ← false
            candidate ← permute_bits(blocks)
            for j = 0 → i − 1 do
                if candidate = block_comb[j] then
                    duplicate ← true
                    break
                end if
            end for
        until duplicate = false
        block_comb[i] ← candidate
    end for
    return block_comb
end function
```

As stated previously, with $M > 0$, each consecutive level hypervector is obtained by flipping b components of the preceding level hypervector. In order to reduce memory consumption, the subsequent hypervectors are not stored explicitly. Instead, a collection of seeds is generated, one for each feature, which is then used to dynamically produce the indices of the components that must be modified when transitioning from one level hypervector to the next. In order to do that, we generate a vector $\overrightarrow{seeds}$ of k components, each one represents a unique integer for generating $M-1$ level hypervectors for each feature.

This whole process is depicted in Figure 1, using the combination of $u = 8$ blocks and $v = 256$. When the level hypervectors are required for computation, they are expanded in memory. Consequently, despite storing each level hypervector using only 1 byte, during both training and classification or inference, there will be at most three simultaneously expanded hypervectors held in memory: 1) the base hypervector HV^j_{base}; 2) the hypervector corresponding to the sample that is being processed $s^j_{i,x}$; and 3) the computed prototypical hypervector $\mathcal{C}_i$. This process is illustrated in Figure 2, where the first level hypervector of a feature is expanded, and the corresponding seed associated with that feature generates the required random indices used to flip the components of the subsequent level hypervectors.

B. Training and Testing Implementation

HDCLite implements the subsequent training stage, aimed at building a class or prototypical hypervector $\mathcal{C}_i$ from the training dataset, by using the encoding scheme described in the previous subsection. For the generation of the prototypical hypervector, the procedure followed is outlined in Algorithm 3.

Algorithm 3 Prototypical hypervector generation

```
function GeneratePrototypicalHypervector(S_i)
    hv_entry ← []; C_i ← []; counter ← 0
    if S_i is ∅ then return C_i
    end if
    foreach s⃗_{i,x} ∈ S_i
        for j = 0 → k − 1 do
            if hv_entry is None then hv_entry ← φ(s^j_{i,x})
            else hv_entry ← hv_entry ⊕ φ(s^j_{i,x})
            end if
        end for
        if C_i is None then C_i ← hv_entry
        else
            hv_entry ← normalize(hv_entry)
            if counter ≥ normalizationValue then
                C_i ← C_i ⊕ hv_entry
                C_i ← normalize(C_i)
                counter ← 0
            else
                C_i ← C_i ⊕ hv_entry
                counter ← counter + 1
            end if
        end if
        hv_entry ← []
    C_i ← normalize(C_i)
    return C_i
end function
```

For each device, the algorithm iterates over all the

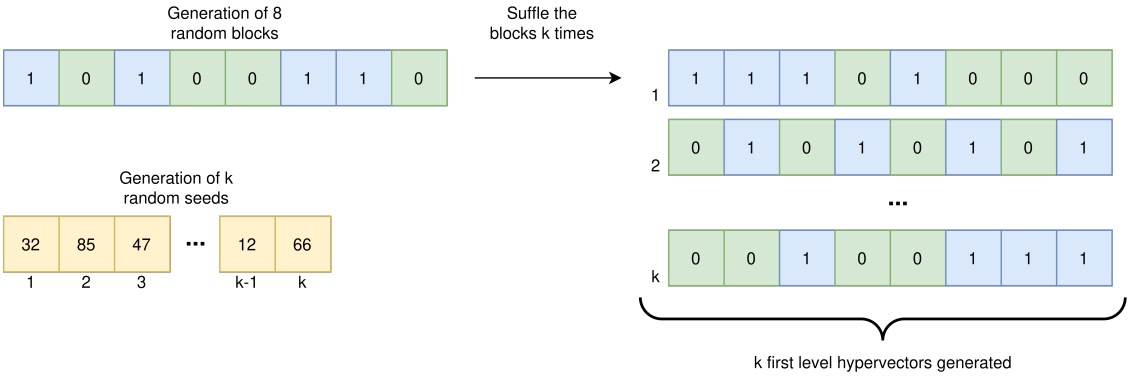


Fig. 1: Generation of the first level hypervectors for each feature k

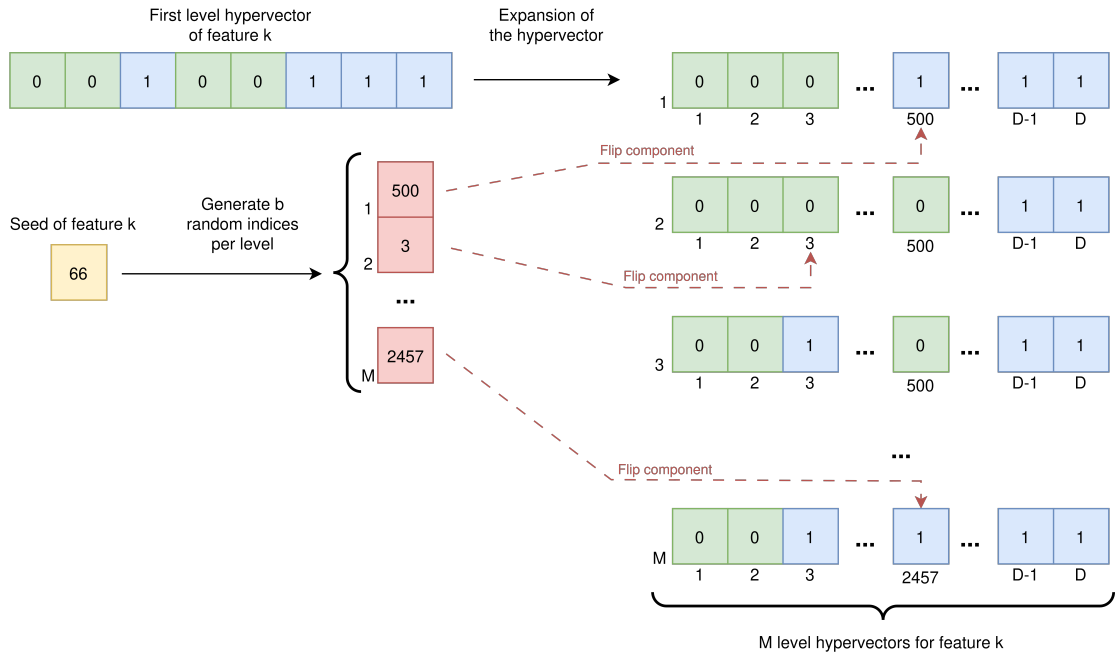


Fig. 2: Dynamic generation of the level hypervectors for each feature k

corresponding data entries and incrementally constructs the prototypical hypervector. Each entry is processed by examining every variable and retrieving the level hypervector associated with the value of the variable under consideration. These level hypervectors are subsequently combined through the bundling operation, thereby aggregating the information from all variables of the entry into a single intermediate hypervector. Once the hypervector corresponding to an entry has been produced, it is normalized and then bundled into the prototypical hypervector. In order to prevent numerical drift, the prototype is periodically normalized after a fixed number of bundling operations, defined by the parameter $normalizationValue$. This process is repeated for every entry in the dataset, ultimately yielding a final normalized hypervector that represents the prototypical, or class, hypervector of the emergency lighting device. Note that, unlike the traditional quantization approach, instead of storing M hypervector levels for each feature, HDCLite only stores three expanded hypervectors at most: 1) the base hypervector HV^j_{base}; 2) the hypervector corresponding to the sample that is being processed $s^j_{i,x}$; and 3) the computed prototypical hypervector $\mathcal{C}_i$.

In the final step of HDCLite, during classification or inference, there will be an input of data to be classified; this will be encoded into a hypervector and, with this, a similarity check will be performed with the stored prototypical hypervector. If the given hypervector falls under a certain threshold, it will be considered an anomaly. Algorithm 4 computes the similarity threshold $\mathcal{B}_i$ for each device d_i, as described in Section IV. For the classification of a new data entry, Algorithm 5 encodes the given entry into its corresponding hypervector and subsequently compares it against the prototypical hypervector of the emergency lighting device by means of the cosine similarity metric. When the resulting similarity value falls below the previously established threshold, the entry is classified as an anomaly. Otherwise, if the similarity exceeds or equals that threshold, the entry is considered to reflect the normal behaviour of the emergency lighting device.

VI. Evaluation

We validate our framework by means a case study targeting anomaly detection in emergency lighting devices was performed. The system involved a network of $n = 118$ emergency lighting units de-

Algorithm 4 Computation of Similarity Threshold

function ComputeThreshold(S_i, C_i)
  $hv_{entry} \leftarrow []$; $similarities \leftarrow []$
  foreach $\vec{s_{i,x}} \in S_i$
    for $j = 0 \rightarrow k-1$ **do**
      if hv_{entry} is None **then** $hv_{entry} \leftarrow \varphi(s^j_{i,x})$
      else $hv_{entry} \leftarrow hv_{entry} \oplus \varphi(s^j_{i,x})$
      end if
    end for
    $cosSim \leftarrow cos(C_i, hv_{entry})$
    $similarities \leftarrow similarities + cosSim$
  $\mu_i \leftarrow mean(similarities)$; $\sigma_i \leftarrow std(similarities)$
  $B_i \leftarrow \mu_i - 2\sigma_i$
  return B_i
end function

Algorithm 5 Classification of test entries

function Testing$(\vec{s_{i,y}}, C_i, B_i)$
  $anomaly \leftarrow false$
  $hv_{test} \leftarrow []$
  for $j = 0 \rightarrow k-1$ **do**
    if hv_{test} is None **then** $hv_{test} \leftarrow \varphi(s^j_{i,y})$
    else $hv_{test} \leftarrow hv_{test} \oplus \varphi(s^j_{i,y})$
    end if
  end for
  $cosSim \leftarrow cos(C_i, hv_{test})$
  if $cosSim < B_i$ **then** $anomaly \leftarrow true$
  end if
  return $anomaly$
end function

ployed throughout several neighbouring buildings. The hardware platform used in the study consists of SmartZ® emergency lighting devices [16], which operate as nodes within a wireless mesh architecture, that automatically activate whenever the primary electrical supply fails, such as during power outages or blackouts. To monitor the operational health of the system, the devices transmit two types of messages at different sampling rates: 1) *status* messages, which contain information related to the operating state of the device; and 2) *diagnosis* messages, which include diagnostic data. We implement HDCLite to learn the normal behavioural patterns of the devices. Deviations from these learned patterns are interpreted as anomalies, enabling the detection of potential failures affecting one or more lighting units.

The implementation of HDCLite was assessed on two different hardware platforms. The first platform corresponds to a laptop equipped with an 11th Gen Intel(R) Core(TM) i5-11400H processor running at 2.70 GHz, 16 GB of RAM, and an NVIDIA RTX 3050 GPU with 4 GB GDDR6 memory. The second platform consists of a Raspberry Pi Pico 2 (RPI), incorporating an ARM Cortex-M33 processor running at 150 MHz, along with 520 KB of on-chip SRAM. All the tests were evaluated for the datasets described in the following subsection, with a number of level hypervectors $M = 3,336$, for different values of dimension $D = \{2048, 4096, 8192, 16384\}$ and the number of blocks $u = 8$. This evaluation aims to examine the performance, computational efficiency, and resource utilization of HDCLite.

A. Experimental Dataset

During two periods of 6 and 24 days, respectively, from December 7 to 12, 2024, and from December 24, 2024, to January 16, 2025, both types of data (*status* and *diagnosis*) were collected from the n devices and merged into a single dataset. In this dataset, a sample is formed by combining one sample from the status data with another sample from the corresponding diagnostic data of the same device, where the transmission *timestamp* matches. The total number of samples collected was 3,021,920, each with 126 features, resulting in a dataset of size 7.7 GB.

This original dataset was divided into two subsets with the aim to simulate a real world scenario where the model is exposed to data it has never seen before, thus avoiding information leakage. Each subset contains 80% and 20% of the entries from the original dataset, respectively. The 80% subset was used for exploratory analysis and subsequent training, while the 20% subset was reserved for the validation or testing phase. This means that the 20% subset intended for validation was not subjected to any type of analysis or preprocessing. As a result of the analysis and preprocessing of the dataset, the number of relevant features is reduced to $k = 32$ and the number of noisy samples is greatly reduced. The characteristics of the training and testing datasets can be observed in the first two rows in Table I.

To properly validate the HDCLite implementation, it was necessary to generate additional datasets to represent anomalous behaviour, since all samples in the original dataset correspond exclusively to *normal* operation. These anomalous datasets were constructed using synthetic data designed to simulate abnormal inputs. Specifically, each feature value in the original testing dataset was iteratively perturbed by injecting heteroscedastic noise ε. This approach enables the generation of anomalies with varying intensity, thus reflecting more realistic deviations from normal behaviour. A description of the resulting synthetic datasets can be observed in the last 3 rows of Table I.

Dataset	**Samples**	**Features** (k)	**Noise** (α)	**Size** (MB)
Training dataset	380,688	32	NA	506.7
Testing dataset	288,598	32	NA	462.9
Synth. dataset 1	288,598	32	0.05	462.9
Synth. dataset 2	288,598	32	0.2	462.9
Synth. dataset 3	288,598	32	0.5	462.9

Tabla I: Summary of the datasets used in the evaluation

B. Machine Learning Metrics

For the evaluation of machine learning models, metrics that quantify how well the model generalizes to unseen data are generally used. In classification tasks, commonly used metrics include accuracy, recall, and the F1-score, each providing a different perspective on model performance.

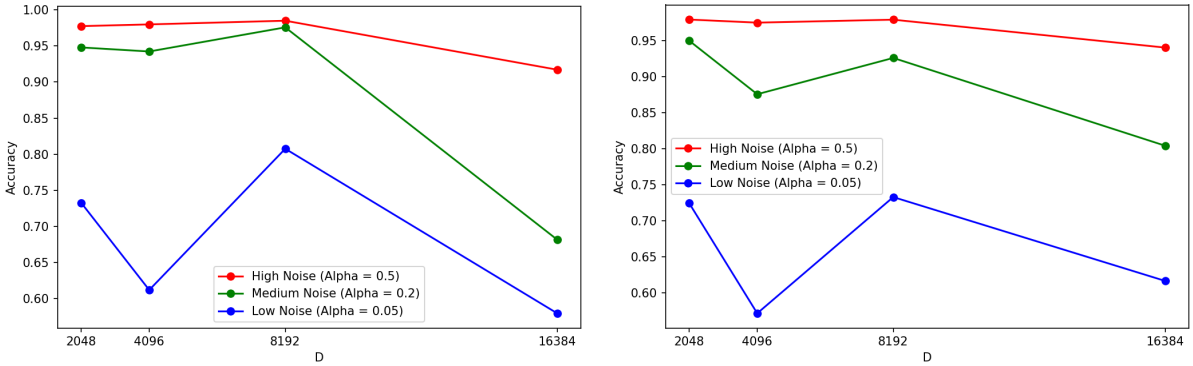


Fig. 3: Accuracy for different values of D and $M = 3,336$ for the PC (left side) and for the RPI (right side)

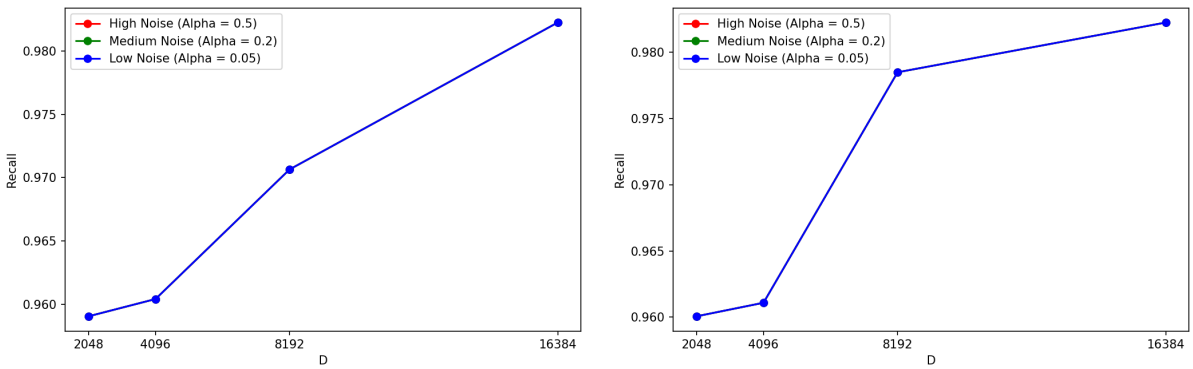


Fig. 4: Recall for different values of D and $M = 3,336$ for the PC (left side) and for the RPI (right side)

Figure 3 presents the accuracy obtained on the PC and RPI platforms, for different values of D and under varying noise levels. As can be observed, both platforms exhibit very high performance under medium and high noise conditions. However, a performance drop appears at low noise levels. This behaviour is due to the fact that the noise becomes so small that the testing samples are extremely similar to the original ones, which in turn leads to a degradation in classification performance. It can also be observed that the performance decreases when reaching $D = 16,384$. Note that the typical dimensionality of the hypervectors is $10,000$. This effect arises from maintaining the same number of level hypervectors M across all tests. For this value of D, the selected M does not properly adjust, since the number of changes between hypervectors b shifts from 1 to 2. At present, the implemented system does not account for this scenario. When the configuration is properly adjusted by setting $M = 8,193$ (recall $b = \frac{D}{2(M-1)}$), the accuracy obtained for $D = 16,384$ increases to 0.985. The evaluation further reveals a slight decrease at $D = 4{,}096$. This behaviour can be attributed to the inherent randomness characteristic of HDC, suggesting that improved results might be achieved under identical conditions by employing a different seed.

Figure 4 shows the recall obtained on the PC and RPI platforms. In contrast to accuracy and F1-score, recall remains consistently high across all noise levels and values of D, including at low noise. This is explained by the fact that the classification threshold is computed from the training data, which tends to be set low enough to capture most positive samples regardless of the noise level, thus minimizing false negatives at the expense of increasing false positives.

C. Memory Usage and Execution Times

HDCLite has been proposed with the aim of outperforming the traditional quantization encoding technique. In particular, with this approach, the hypervector dimensionality D, the number of input features k, and the number of level hypervectors M have a significant influence on memory consumption, with the storage cost scaling as $O(k \times M \times D)$. In HDCLite, this storage consumption changes due to the dynamic generation of the level hypervectors when required, and the first level hypervectors are stored using a 1 byte representation. Consequently, the factor that most strongly impacts memory usage is the hypervector dimensionality D during the moments in which hypervectors must be expanded in order to perform the required computations.

In our implementation, memory consumption has been computed as the sum of 32 bytes required to store the first level hypervectors, 32 bytes allocated for the seeds (each seed being a integer of 8 bits), and the hypervector dimensionality D corresponding to the 3 hypervectors (sample hypervector, level hypervector and prototypical hypervector) that, at most, need to be expanded simultaneously. As shown in Figure 5, memory usage increases with larger values of D. Nevertheless, even for relatively high dimensionalities such as $D = 16,384$, the overall memory footprint of HDCLite remains very low, with a memory consumption of 48.06 KB, ensuring that it can

be efficiently deployed in an embedded system. In comparison with a non-optimized quantization implementation of HDC, HDCLite demonstrates the most significant reduction in memory usage. Specifically, HDCLite achieves up to a 142,151× reduction in memory usage compared to a non-optimized quantization strategy that represents each hypervector component as a 32-bit integer.

The time metrics obtained for both platforms correspond to the total training time required by each platform and the testing time for the entire testing dataset, rather than the inference time for a single data sample. Figure 6 presents the training time (left side) for the PC platform (in blue) and for the RPI (in green). As expected, a resource constrained platform such as the RPI requires several orders of magnitude more time than the PC platform, which benefits from significantly more powerful hardware. Furthermore, the training time increases proportionally with D, since larger hypervectors require a greater number of operations. In the worst case scenario, the training time reaches 5.85 seconds for the PC and 5,993.96 seconds for the RPI. Figure 6 also shows the testing time results (right side) for both platforms using the same color scheme. As it can be observed, the same trends appear: the RPI requires several orders of magnitude more time than the PC, and larger values of D increase the execution time for both platforms. In the worst-case scenario, the testing time reaches 2.93 seconds for the PC and 8,808.40 seconds for the RPI.

D. HDCLite Profiling with Codasip

Codasip [17] is a technology company focused on processor design, aimed at enabling system-on-chip (SoC) developers to achieve product differentiation. The company concentrates on RISC-V based processor solutions and provides Codasip Studio, a comprehensive toolchain for processor development based on the CodAL architecture description language. This framework provides a set of preconfigured RISC-V processor cores spanning multiple architectural variants, including a vector core implementation. In this work, the vector processor has been adopted as the reference platform to conduct the experimental evaluation and comparison of HDCLite against a previous implementation of ours. The assessment has been carried out using the profiling tool integrated within Codasip Studio, which enable detailed performance and resource usage analysis at a cycle level of detail. As shown in Figure 7, the most frequently executed instructions correspond to those involved in the computation of the bundling operation, as well as the memory access through data load and store operations. Furthermore, it can be observed that HDCLite not only reduces the memory footprint, but also lowers the number of cycles and computational resources required by the HDC pipeline, with an improvement factor of 2.74× with regard to the quantization-based HDC method.

Although a RISC-V vector core was employed, no vector-specific optimizations were enabled; consequently, the most frequently executed instructions are not vector operations. This observation indicates that there remains significant potential for further optimization in terms of computational efficiency.

VII. Conclusions

In this work, we introduce HDCLite, a technique for compressing bipolar hypervectors within quantized HDC frameworks, aimed at enabling efficient deployment on embedded platforms. By refining the encoding and storage strategy of level hypervectors, the proposed technique achieves a substantial reduction in memory footprint while preserving classification performance. This improvement makes it feasible for resource constrained devices to perform advanced learning and inference tasks that would otherwise surpass their hardware limitations, thus facilitating the adoption of the HDC framework in embedded system architectures. Additionally, the system has been evaluated using Codasip in order to identify and analyze low-level operations susceptible to further optimization.

The proposed approach has been validated through the implementation of an anomaly detection system based on HDC, assessed on two distinct platforms: a PC/Linux environment and a Raspberry Pi Pico 2. The evaluation considers execution time, memory consumption, and ML metrics. Experimental results indicate that with HDCLite, HDC solutions can be effectively deployed across heterogeneous platforms while maintaining accuracy levels close to 98%.

As part of future work, we intend to extend HDCLite towards specialized hardware accelerators, with particular emphasis on improving the efficiency of vector instruction execution.

Acknowledgments

This research is partially funded by the Regional Government of Castilla-La Mancha (INNOCAM) and the ERDF through the RURALCARE (SBPLY/24/180225/000158) and REACH (SBPLY/24/180225/000105) projects, by the Spanish Ministry of Economy, Trade and Enterprise through the OASIS project (PID2023-148285OB-C41), the PERTE-Chip programme (TSI-069100-2023-0014), and the Horizon Europe programme through the MAYA project (GA No. 101213323).

References

[1] Sherenaz W Al-Haj Baddar, Alessio Merlo, Mauro Migliardi, et al., "Anomaly detection in computer networks: A state-of-the-art review.," *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, vol. 5, no. 4, pp. 29–64, 2014.

[2] Animesh Patcha and Jung-Min Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[3] Varun Chandola, Arindam Banerjee, and Vipin Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[4] Jose Edson de Albuquerque Filho, Laislla CP Brandao, Bruno José Torres Fernandes, and Alexandre MA Ma-

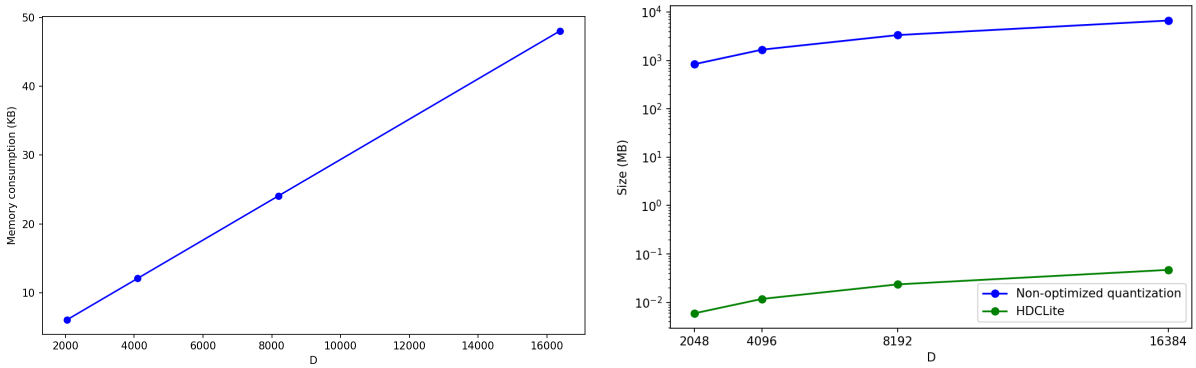


Fig. 5: Memory RAM consumption for different values of D in KB (left side) and memory RAM consumption comparison for different values of D between different HDC approaches in MB (right side)

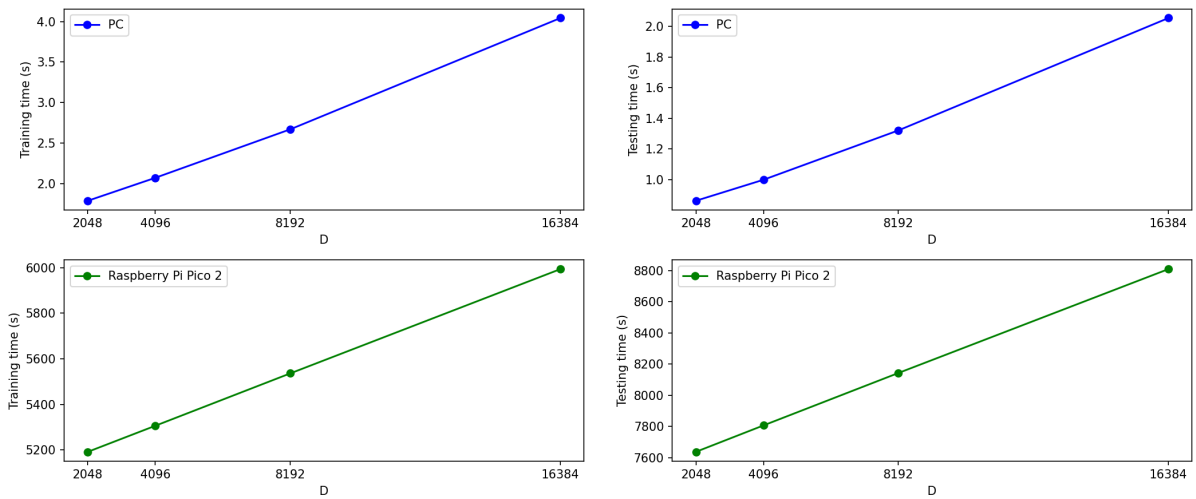


Fig. 6: Training time (left side) and testing time (right side) for different values of D in seconds

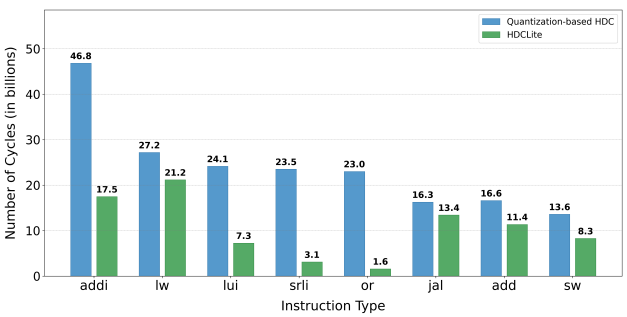


Fig. 7: Profiling comparison: HDCLite vs Quantization-based HDC

ciel, "A review of neural networks for anomaly detection," *IEEE Access*, vol. 10, pp. 112342–112367, 2022.

[5] Pentti Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, pp. 139–159, 2009.

[6] Denis Kleyko, Dmitri A Rachkovskij, Evgeny Osipov, and Abbas Rahimi, "A survey on hyperdimensional computing aka vector symbolic architectures, part i: Models and data transformations," *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–40, 2022.

[7] Pere Vergés, Mike Heddes, Igor Nunes, Denis Kleyko, Tony Givargis, and Alexandru Nicolau, "Classification using hyperdimensional computing: A review with comparative analysis," *Artificial Intelligence Review*, vol. 58, no. 6, pp. 173, 2025.

[8] Ruixuan Wang, Xun Jiao, and X. Sharon Hu, "Odhd: one-class brain-inspired hyperdimensional computing for outlier detection," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, New York, NY, USA, 2022, DAC '22, p. 43–48, Association for Computing Machinery.

[9] William Arliss, Andrew B Godbehere, and Graham Mueller, "Using hypervectors for efficient anomaly detection in graph streams," in *2024 IEEE 11th International Conference on Data Science and Advanced Analytics (DSAA)*, 2024, pp. 1–10.

[10] Ruixuan Wang, Fanxin Kong, Hasshi Sudler, and Xun Jiao, "Brief industry paper: Hdad: Hyperdimensional computing-based anomaly detection for automotive sensor attacks," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021, pp. 461–464.

[11] Ghazal Ghajari, Ashutosh Ghimire, Elaheh Ghajari, and Fathi Amsaad, "Network anomaly detection for iot using hyperdimensional computing on nsl-kdd," 2025.

[12] Sina Shahhosseini, Yang Ni, Emad Kasaeyan Naeini, Mohsen Imani, Amir M. Rahmani, and Nikil Dutt, "Flexible and personalized learning for wearable health applications using hyperdimensional computing," in *Proceedings of the Great Lakes Symposium on VLSI 2022*, New York, NY, USA, 2022, GLSVLSI '22, p. 357–360, Association for Computing Machinery.

[13] Víctor Ortega, Soledad Escolar, Fernando Rincón, Jesús Barba, and Julián Caba, "An efficient and scalable hyperdimensional computing framework for anomaly classification in industrial systems," in *Proceedings of the 22nd International Conference on Informatics in Control, Automation and Robotics - Volume 2: ICINCO*, 2025, pp. 429–436.

[14] Toygun Basaklar, Yigit Tuncel, Shruti Yadav Narayana, Suat Gumussoy, and Umit Y. Ogras, "Hypervector design for efficient hyperdimensional computing on edge devices," 2021.

[15] Ruixuan Wang, Fanxin Kong, Hasshi Sudler, and Xun Jiao, "Brief industry paper: Hdad: Hyperdimensional computing-based anomaly detection for automotive sen-

sor attacks," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 461–464.
[16] "Zemper smartz — brochure," 2026, Accessed on February 12, 2026.
[17] Codasip, "Codasip official website," `https://codasip.com/`, Accessed: 2026-03-20.
[18] Lulu Ge and Keshab K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE Circuits and Systems Magazine*, vol. 20, no. 2, pp. 30–47, 2020.

Docencia

NuCachis and NuTracer: Teaching Computer Organization through an Interactive RISC-V Memory Simulation Ecosystem

Aitor Echevarría[1], Borja Peréz[2], Esteban Stafford[2] and José Luis Bosque[2]

Resumen— **Teaching computer organization is challenging due to its complex structure and abstract concepts. Simulators and visual learning tools help students reinforce ideas, but are often inflexible, unintuitive, limited or hard to modify, reducing the effectiveness of hands-on sessions and increasing the effort required to create insightful assignments. This work presents NuCachis, an educational, open-source, interactive memory hierarchy simulator designed to be portable and easy to use and modify. It also introduces NuTracer, a complementary C-like pseudocode parser and RISC-V pseudo-assembly interpreter that helps generate input traces for NuCachis. Together, these tools have been successfully integrated into the Computer Organization course at the Universidad de Cantabria, where they have proven effective in helping students grasp otherwise difficult memory hierarchy concepts.**

Palabras clave— **RISC-V, NuCachis, NuTracer, Computer Organization, Memory Hierarchy, Cache Hierarchy, Simulation**

I. Introduction

The complexity and abstraction inherent in computer organization and architecture make these subjects difficult to teach and learn. As part of contents related to memory hierarchy of the syllabus, students are required to understand both the fundamentals and the hardware details of a complex, multi-level memory hierarchy, as well as how different memory access patterns affect it.

Conventional teaching approaches, such as lectures or textbooks, have proven to be insufficient, as understanding the details of caches, memory and how different architectural parameters affect behavior and performance is not straightforward. Students often struggle to piece together these abstract concepts and fully understand details due to the lack of visual representation and the limitations of traditional explanations.

To consolidate these concepts, additional tools are required. Interactive simulators and other visual learning tools are a great complement to courses, as they allow students to experiment and bridge the gap between theory and actual implementation. However, for such tools to be useful they need to be flexible, so the student can learn about different cache organizations and policies, user-friendly, to reduce the learning curve and, ideally, open-source, to allow the tool to be extended with further capabilities.

Computer architecture simulators are of two main types: Execution-driven and trace-driven. The former are designed to take an executable as input, process instructions as they run and dynamically simulate different aspects of the computer. These kinds tools are excellent for performance profiling and analysis, which require precise results, at the expense of having a steep learning curve and being more complex to use and modify. On the other hand, trace-driven simulators parse a trace that represents a sequence of events generated by the program. While this sometimes limits accuracy, it allows for simpler simulators and, in some cases, the ability to modify or use compact test traces.

Simulation tools have proven effective in illustrating various aspects of computer operation [1], [2], [3]. However, despite the wide range of simulation options currently available, none provide the level of visualization and flexibility necessary to effectively study the full complexity of the memory hierarchy in an educational environment. The aim of this work is to introduce "NuCachis" [4], a simple and interactive trace-based memory hierarchy simulator designed exclusively for educational purposes that focuses on user friendliness, portability and ease of modification, as current simulation tools are either heavily focused on professional use or have many of the shortcomings mentioned above. This approach allows students to try different memory and cache configurations, and traces to better understand how system performance and organization are influenced. To reinforce this workflow and allow for simpler experimentation, the complementary "NuTracer" [5] tool helps create complex traces from C-like pseudocode and can be used with ease by teachers and students familiar with the C programming language and RISC-V assembly.

Additionally, this article presents an evaluation and comparison of current simulators, both desktop and web-based, and their strengths and limitations in the context of education. This comparison highlights the need for a simulator that meets the aforementioned criteria.

Finally, we explain how the tools have been integrated into the computer organization course at Universidad de Cantabria. This has enabled us to confirm that the proposed methodology, based on specially crafted laboratory assignments and experi-

[1]Dpto. de Tecnología Electrónica e Ingeniería de Sistemas y Automática, Universidad de Cantabria, under contract ***Cátedra Chip Cantabria*** (VII) e-mail: echevarriafa@unican.es

[2]Dpto. de Ingeniería Informática y Electrónica, Universidad de Cantabria, e-mail: {borja.perezpavon, esteban.stafford, joseluis.bosque }@unican.es

mentation, is effective at helping the students understand the nuances of the memory hierarchy .

The remainder of this article is structured as follows. Section 2 reviews state of the art computer organization simulators, highlighting their strengths and limitations for didactic purposes. Section 3 defines the requisites and design choices a suitable simulator should comply with. Section 4 contains an overview of NuCachis and NuTracer, with configuration and trace examples. Section 5 presents laboratory assignments designed to illustrate cache behavior, replacement policies and field calculation. Section 6 presents results and alumni experience. Lastly, Section 7 provides conclusions and outlines possible directions for future work.

II. Simulator Tools for Teaching Computer Organization

A wide range of computer organization tools and simulators exist, each varying in terms of features, flexibility and ease of use. Focusing on their didacticity, we evaluate the strengths and weaknesses of each simulator and identify the need for a new one that meets all criteria. A rough feature comparison is shown in Table I.

Since computer architecture and memory hierarchy are widely researched topics and are taught at many universities throughout the world, there is an extensive selection of simulators to choose from. These range from light web-based tools with limited configurability to complex execution-driven, complete system emulators, both featuring different sets of capabilities.

Starting with web tools, ParaCache [6] is a reasonably featured simulator that includes prebuilt single-level cache associativity configurations and supports different replacement and write policies. Another alternative is the University of Michigan's cache simulator [7], featuring a pseudocode interpreter, though with a simpler interface and limited configuration options. Both of these simulators are excellent for quick demonstrations and light experimentation, but lack key features for more involved assignments, such as multi-level cache support, and are cumbersome to use, as their interfaces are basic and require manual configuration of each parameter on startup.

On the other hand, desktop applications typically bundle more complex behavior and features, such as multi-level cache hierarchy simulation. SMP Cache [8] is a teaching simulator that offers easy to select, varied configurations; however, it is multiprocessor-focused, features an outdated interface, and does not have publicly available source code. MARS [9] is a MIPS focused simulator that aims to teach several varied aspects of the processor. It is highly portable and intuitive, though the cache-specific simulation tools are limited and more focused on numeric calculation than visual representation.

Dinero IV [10] and Valgrind's Cachegrind [11] are designed for performance profiling and research purposes. Both are capable open-source and terminal-based alternatives that work with traces and executables respectively. They are easy to set up, configure, and use, but lack graphical interfaces, which represent a major hurdle when using the tool in the classroom. In the case of Valgrind's Cachegrind, modifications can also be difficult. Gem5 [12] is a fully configurable simulator capable of modeling entire processors and memory hierarchies with varying degrees of complexity and configurability. It is an intricate terminal-based tool used for research and might prove hard to use for beginners. Setup and configuration are involved processes, and modifications are not trivial due to its sophisticated design. Several other simulators of comparable detail to gem5 could also be considered in this comparison, but their complexity defeats the purpose of an easy to use tool for beginners.

Tabla I: Feature comparison of different simulators

	Usability	Modifiability	Configurability	Portability	Features
ParaCache	●	○	◐	●	◐
UMICH Cache Simulator	◐	○	○	●	○
SMP cache	◐	○	◐	◐	◐
MARS	●	◐	◐	●	○
Valgrind	◐	◐	●	●	◐
Dinero IV	◐	●	●	●	◐
gem5	○	○	●	◐	●

Legend: ●= provides property, ◐= partially provides property, ○ = does not provide property

As shown in the comparison, web tools often feature interactive graphical interfaces, but are of limited functionality and not easily modifiable. On the other hand, desktop simulators are more feature-packed and focused on research or accurate simulation, but lack easy to use graphical interfaces, and adapting their functionality and designing assignments is time consuming.

For teaching computer organization in the past, the authors have extensively used Cachegrind in combination with it's postprocessor `cg_annotate`, as well as the graphical MARS' cache simulation module for teaching computer organization. `cg_annotate`'s output pairs lines of code with cache statistics, allowing students to understand how different memory access patterns might perform and which ones are more cache aware. This approach, however, offered poor visual representation and little insight in understanding concepts related to replacement and mapping policies. In contrast, MARS offered slightly better academic results with a more visual, albeit simpler experience, as it is a general purpose architectural simulator.

Considering the previous comparison and personal experiences, the authors have determined that graphical representation yields better learning results than analytical tools. Hence, there is a need for a graphical simulator that combines portability, configurability, and a rich set of features suitable for teaching and experimentation.

III. Simulator Design and Requirements

In order to fulfill this need, all of the previous criteria were taken into consideration during the design phase. The NuCachis simulator meets a set of functional requirements that represent its capabilities and *features*. As a computer organization simulator, it supports the following:

- Multi-level cache hierarchies, as well as a cacheless configurations.
- Separate instruction and data caches.
- Direct, set associative and fully associative mapping policies.
- Basic line replacement policies (LRU, LFU, FIFO and random).
- Write-through and write-back policies.
- Simple DRAM modeling.
- Timing and line statistics, including burst access times for main memory.
- Trace-based simulation with breakpoints.

Additionally, in order to develop a tool that is well-suited for teaching and ensure that the remaining criteria are met, several additional non-functional requirements were also defined. These requirements represent characteristics of the simulator, instead of functionality:

An intuitive and interactive user interface was created to improve *usability*. The application is optimized for ease of use, requiring little training, and is designed to display useful information without excessive details, while showing the global state of the program and highlighting only relevant changes. The interface's layout is dynamic, resizable and able to adapt to various different resolutions.

Elements of the simulator were categorized into classes that represent real components of a memory system, such as a `Cache` or a `MemoryLine`, as seen on Figure 1. The use of inheritance allows the addition of new memory element types by simply extending the `MemoryElement` class. The object oriented paradigm allows for better maintainability and *modifiability*, as most simulator concepts are mapped to real objects. C++ was the chosen programming language for the project due to its extensive collection of libraries and robustness.

A feature present in most desktop-based simulators but missing from web-based ones is the ability to *easily customize most simulation parameters* and load configurations quickly, either with command line arguments or configuration files. NuCachis supports setting up all memory hierarchy parameters through a single configuration file that is independent from the trace. These can be selected either with a file picker or command line arguments for faster or more user-friendly interaction respectively. The use of separate files allows teachers to deliver per-exercise configurations and students to easily modify and experiment with different simulation parameters.

One of the key challenges in developing applications with graphical user interfaces is selecting an

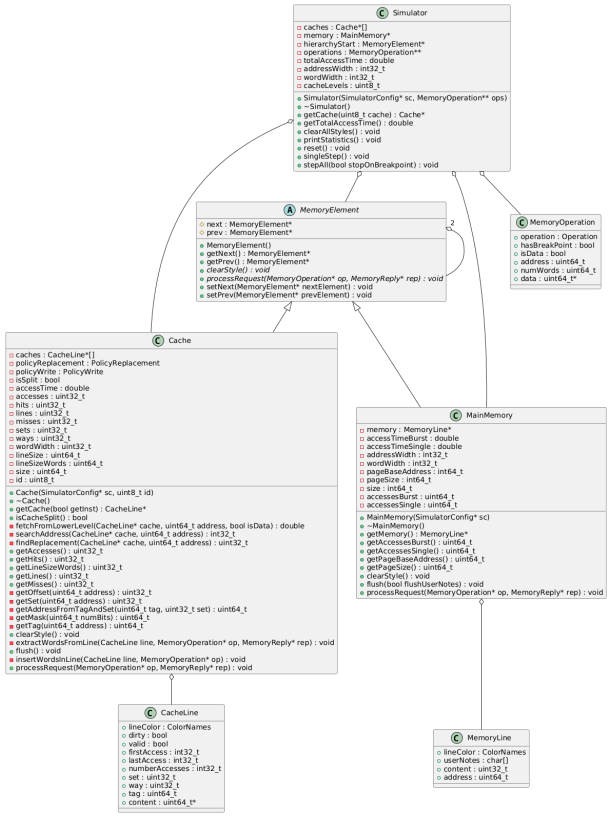


Fig. 1: Simplified Class Diagram

appropriate technological stack to build upon. While there is no one-fits-all solution, when considering *portability*, minimal dependencies and code simplicity, large frameworks such as QT or GTK are not feasible. For this reason, NuCachis was built using Dear ImGui[13], a minimal library that only requires OpenGL and SDL2 and can be compiled and run on different platforms and even different architectures.

Lastly, documentation, installation instructions and sample configuration (Figure 2) and trace (Figure 3) files are provided alongside the code.

IV. Ecosystem Overview

Upon starting NuCachis, the user will be presented with a window in which configuration and trace files can be selected. If desired, the files can be passed through the command line with the corresponding arguments. Additionally, the simulation can be run without a graphical user interface by using the `--nogui` command line flag.

After launching the simulation, the program will display 3 main windows, as seen on Figure 4.

The leftmost panel contains the instruction window that represents the set of memory accesses performed by the CPU, as described on the trace file, and the miscellaneous window. The instruction window displays the last access that was performed in gray, as well as information related to each memory access. Controls are present at the top of the window to step one access at a time (`Single Step`) or step until a breakpoint is reached or no more accesses are left (`Step All`), as well as a `Reset` button

```
[cpu]
address_width = 32
word_width = 32
rand_seed = 1234

[cache1]
line_size = 16
size = 256
associativity = 8
write_policy = wb
replacement_policy = lru
separated = yes
access_time = 25m

[cache2]
line_size = 16
size = 512
associativity = 16
write_policy = wt
replacement_policy = lfu
separated = no
access_time = 50m

[memory]
size = 2G
access_time_1 = 1
access_time_burst = 250m
page_base_address = 0x8000000
page_size = 1k
```

Fig. 2: Sample 2-level configuration

```
#Initial loads
L 0x08000040 D
L 0x08000060 D
L 0x08000080 D
L 0x080000A0 D
L 0x08000000 I
L 0x08000010 I
L 0x08000020 I
L 0x08000030 I

# L1 Stores
!S 0x080002A0 D          #This will store a 0
S 0x08000100 D 2345
S 0x08000124 D 2345

# L1 Overwrites that cause replacements
L 0x08000000 I
L 0x08000000 D
```

Fig. 3: Sample trace

to restart the simulation. Furthermore, stepping the simulation generates an informational trace of messages printed through standard output to indicate changes and operations performed on each level of the hierarchy. Below, on the miscellaneous window, two selectable tabs group the statistics of each level and a legend for the colors used in highlighting.

The central panel contains the cache hierarchy, grouped by levels. Information such as the tag, set, way, access times and dirty and valid bits are displayed per line in a compact manner. Contextual help about what each abbreviated field is gets shown to the user when hovering over the column header. Lines that have recently been accessed get highlighted in green or red if there has been a hit or a miss respectively. When a cache-less simulation is being run, this panel will not be rendered.

The right panel contains the main memory overview. Each line represents a 4 byte word with its address and content. For simplicity, only a segment of the total memory is displayed, which can be modified in the configuration file for a particular simulation. Additionally, a field for per-word user notes is included for brief annotations about memory locations.

For trace generation, NuTracer offers a similar user experience, featuring both GUI and CLI execution modes. Upon startup, the user will be prompted for a pseudocode file and a NuCachis configuration file. After providing both, the program will automatically detect all variables in the pseudocode, display the main interface (Figure 5), allow the user to set per-variable custom addresses and select how often each of them should appear on the trace to simulate the use of registries. Automatically generated comments can also be appended to the trace for better readability.

Once all variables are set up, NuTracer will parse the input files and generate a list of pseudo RISC-V assembly operations. An interpreter will simulate all operations and store each variable access to a buffer that can be saved to a trace file. To help with troubleshooting, a debug mode is present, and checks on the code are performed to detect infinite loops, syntax errors, and out-of-bounds accesses.

V. Course Integration and Laboratory Assignments

The simulator has been introduced in the Computer Organization course of Universidad de Cantabria, which is composed of theoretical lectures and practical laboratory assignments that involve understanding the effects of the memory wall, the purpose of caches and their design implications. By actively experimenting with different arrangements, configurations and policies, students are able to achieve a higher level of subject knowledge beyond theoretical proposals.

Previous assignments were redesigned to exploit the capabilities of the new simulator. Each assignment is designed to explore fundamentals in a step-by-step manner, gradually introducing new concepts and allowing students to freely experiment different combinations, even without extensive knowledge of the microarchitecture or other details required by other simulators.

A. Visual cache representation and policy analysis

The aim of this assignment is to familiarize students with how different cache geometries and policies affect data access patterns and performance, as well as the simulator and its workflow.

For this assignment, students are provided with a set of deliverables composed of configuration and trace files for the simulator, as well as a document with an introduction and an exercise sheet.

Step 1: Basic for-loop and tool usage. Students are provided with the trace representation of a simple for loop and a single-level cache hierarchy and are asked to identify the cache geometry, policies, and note hits and misses. The goal is for students to understand the hierarchy and visualize how a cache fetches and stores data. After understanding the basic concepts, a question is posed about how data presence in a cache is handled and how tags are calculated.

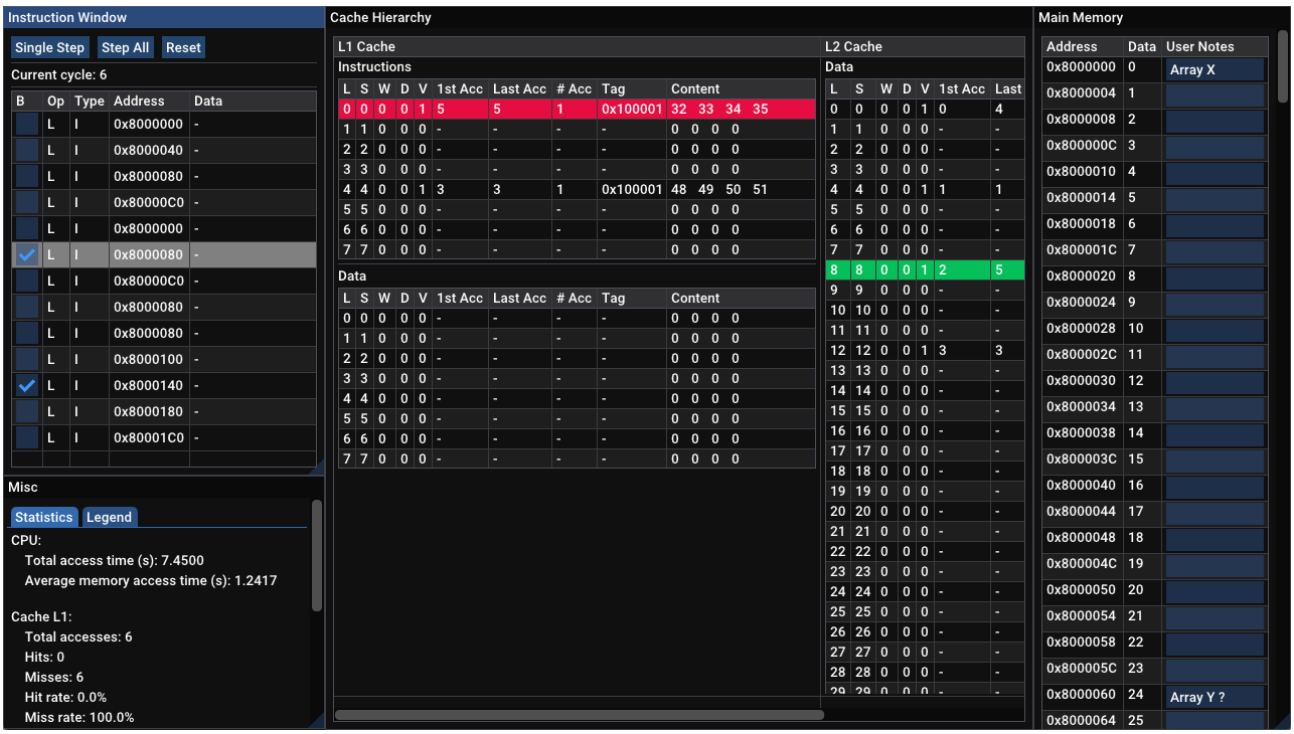


Fig. 4: Main NuCachis workspace

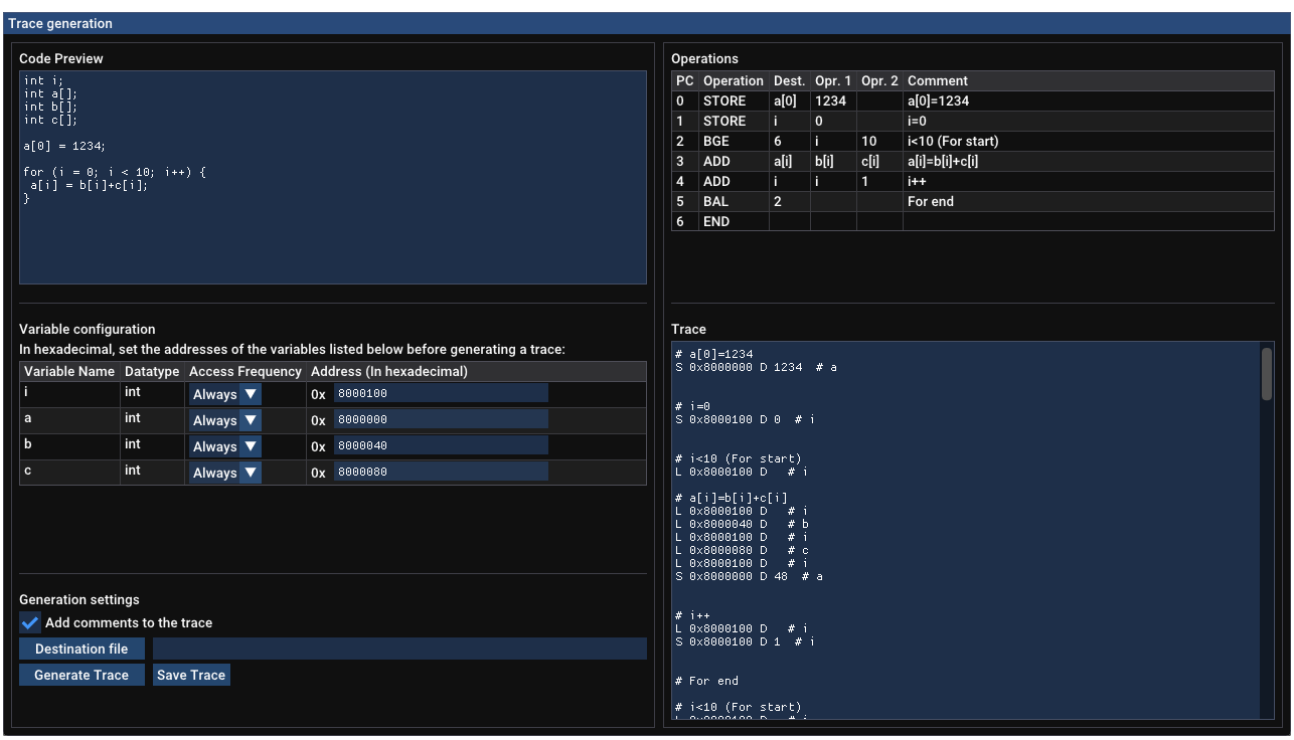


Fig. 5: Main NuTracer workspace

Step 2: Mapping policy comparison. Students are presented with a larger access pattern and 4 different configuration files featuring direct mapped, 2-way, 4-way and fully associative caches. Then, they are asked to compare the behavior and mapping flexibility across different policies and to provide their insights on how set calculation works.

Step 3: Effect of line width on different mapping policies. To better understand how conflicts work and why fully associative caches offer the most flexibility, a larger trace is provided and students are asked to experiment with line widths ranging between 8 and 32 Bytes. Results show that fully associative caches remain unaffected by collisions and have fewer misses as the line width increases. Other policies display a constant, but larger number of misses caused by collisions, and access times increase with line width as more data has to be fetched on each miss.

Step 4: Replacement policy comparison. To visualize how mapping policies affect the flow of data on caches, students are asked to compare the behavior of the LRU, LFU, FIFO and random replacement policies when processing a common for loop in a 2-way associative cache, as well as discussing why some may perform better than others on more practical cases.

Step 5: Stride effect on caches. Finally, to help visualize the importance of spatial locality in code, students are asked to generate a set of traces using a for loop, with the goal of yielding a miss rate of 25, 50 and 100%, by increasing the stride of the iterator. As the stride approaches the number of words stored in a line, the miss rate increases, reinforcing the concept and impact of line width on miss rates.

By following this approach, students better understand data flow in caches and the impact of seemingly unimportant code details on performance, reinforcing their understanding of the memory hierarchy as a whole. This process clearly displays the intricacies and details of caches, such as the different mapping or replacement policies. Such insights are very hard to convey without the kind of visual aid that NuCachis allows.

B. Set and tag calculation and replacement policy implementation

Replacement policies, such as FIFO or LRU, are commonly used across several fields of computer science and are of great importance. Students often face difficulties understanding how these policies integrate with caches, particularly how caches determine the line to replace and their integration with different mapping policies. Additionally, set and tag calculation, which are the foundation of mapping policies, are also challenging when first encountered by students.

To bolster these concepts, the second assignment was designed following a more implementation-focused methodology, as the authors have experienced better results with these kinds of techniques for more technical topics. Students are handed a document with an introduction, an exercise sheet and a C function and bit-operator appendix, as well as a partially incomplete implementation of the source code of NuCachis. The assignment is split into two main sections:

Step 1: Mapping policy implementation. The `getSet()` and `getTag()` functions are used to calculate the set and tag for a given address. Students must implement these functions by applying bit operations on the address, giving them insight into how these fields work.

Step 2: Replacement policy implementation. The `findReplacement()` function finds the line to be replaced upon an access to the cache. Based on the previously implemented set and tag functions, students must find the set to which the address belongs and iterate over all the ways in the set to find the most suitable candidate for a replacement. Each line contains information such as the first, last and total number of accesses to aid in implementing the FIFO, LRU and LFU policies respectively. After finishing the implementation, a comparison with the output of the original simulator allows for direct validation and eases the troubleshooting process, as the output of the simulator is deterministic.

With this assignment, students gain hands-on experience with address field calculation, and consolidate their knowledge about how mapping and replacement policies integrate in caches.

VI. Results and Experience

The NuCachis ecosystem has been successfully integrated into laboratory classes and regular courses. The tools were used by a total of 67 students split in 3 groups.

With the inclusion of optional sections in laboratory assignments, students across a range of different skill levels are able to progress consistently while still learning the core concepts. The portability of the simulator allows students to explore and work at their own pace, even from home, as the binaries can be easily compiled and run outside of laboratories.

The visual learning component of the assignments has proven effective and has helped students better understand and internalize the abstract and often troublesome concepts of computer organization. The programming assignment has also yielded positive results and improved students' awareness of mapping and replacement policies.

After laboratory assignments concluded, an anonymous poll was carried out to assess student satisfaction and identify possible future improvements. The following questions were presented, with satisfaction levels rated (Figure 6) on a scale from 0 (least satisfied) to 5 (most satisfied):

- Q1: Do you consider that the use of NuCachis has helped you to better understand how the memory hierarchy works (misses, replacement policies, write policies ...)?
- Q2: Do you consider that NuCachis facilitates a better visualization of memory processes than

traditional theoretical methods such as diagrams or tables?

- Q3: Do you think that the tool is useful for verifying and solving practical problems from the course in an autonomous way?
- Q4: What is your general level of satisfaction with the NuCachis tool (interface, ease of use and clarity)?

Question	Average	Std. Deviation
Q1	3.9	1.2
Q2	4.2	0.8
Q3	4.1	1.2
Q4	3.8	1.4

Fig. 6: Poll results.

VII. Conclusions and Future Work

This experience proves that visual and interactive learning tools play a crucial role in improving both learner understanding and teaching effectiveness. The ability to try different memory configurations and policies provides insightful hands-on experience and makes bridging the gap between theory and implementation simple. By integrating the NuCachis and NuTracer ecosystem into courses, students' performance and overall satisfaction have demonstrated improvement over time.

Future work for this project will include improvements to the NuCachis simulator, such as the inclusion of alternative write policies, management of byte, half word and double word accesses, and graphical plotting or statistics exporting capabilities. NuTracer syntax extensions to include more complex structures such as while loops or individual conditionals are also planned. Additionally, extending future assignments to take advantage of multi-level cache support could be interesting, as it depicts a more realistic scenario in current processor designs.

Acknowledgments

This work has been partially funded by Project TSI-069100-2023-0011 "Cátedra Chip Cantabria", funded by the Ministry for Digital Transformation and the Civil Service and by the European Union through the Next GenerationEU programme and the Spanish Science and Technology Commission under contract PID2022-136454NB-C21.

Referencias

[1] Morten B Petersen, "Ripes: A visual computer architecture simulator," in *2021 ACM/IEEE Workshop on Computer Architecture Education (WCAE)*. IEEE, 2021, pp. 1–8.

[2] Diego Camarmas-Alonso, Felix Garcia-Carballeira, Alejandro Calderon-Mateos, and Elias Del-Pozo-Puñal, "Creator: An educational integrated development environment for risc-v programming," *IEEE Access*, 2024.

[3] E. Stafford, B. Perez, and J.L. Bosque, "Learning computer architecture with a visual simulation of risc-v processors," in *RISC-V Summit Europe*, 2025.

[4] Aitor Echevarría, "Nucachis: A simple, interactive cache simulator," `https://github.com/aiechevarria/nucachis-public`, Accessed: 2026-04-08.

[5] Aitor Echevarría, "Nutracer: A simple c-like interpreter and trace generator for nucachis.," `https://github.com/aiechevarria/nutracer`, Accessed: 2026-04-08.

[6] Aryani Paramita, "Paracache," `https://www3.ntu.edu.sg/home/smitha/ParaCache/Paracache/start.html`, Accessed: 2026-03-05.

[7] Harrison Davis and Satish Narayanasamy, "University of michigan's cache simulator," `https://vhosts.eecs.umich.edu/370simulators/cache/simulator.html`, Accessed: 2026-03-05.

[8] Juan A. Gómez Pulido Miguel A. Vega-Rodríguez, Juan M. Sánchez Pérez, "Smpcache," `http://arco.unex.es/smpcache/`, Accessed: 2026-03-06.

[9] Ken Vollmar and Pete Sanderson, "A mips assembly language simulator designed for education," *Journal of Computing Sciences in Colleges*, vol. 21, no. 1, pp. 95–101, October 2005.

[10] Mark D. Hill Jan Edler, "dinero iv," `https://pages.cs.wisc.edu/~markhill/DineroIV/`, Accessed: 2026-03-06.

[11] Nicholas Nethercote and Valgrind Developers, "Cachegrind," `https://valgrind.org/`, Accessed: 2026-03-06.

[12] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[13] Omar Cornut, "Dear imgui: Bloat-free immediate mode graphical user interface for c++," `https://github.com/ocornut/imgui`.

Un intercomunicador de audio de baja latencia con compresión perceptual basada en paquetes wavelet para la docencia de tecnologías multimedia

Savíns Puertas-Martín, Juan José Moreno y Vicente González Ruiz[1]

Resumen— **En este trabajo se presenta InterCom, un intercomunicador de audio full-duplex y de baja latencia, implementado íntegramente en Python y diseñado como herramienta docente para la enseñanza de Tecnologías Multimedia mediante Aprendizaje Basado en Proyectos (ABP). InterCom emplea una arquitectura por capas donde cada módulo añade funcionalidad de forma incremental siguiendo una progresión pedagógica que permite a los estudiantes explorar de forma aislada cada etapa del problema tasa-distorsión. Se formalizan las propiedades de ortogonalidad de las transformadas utilizadas, se analiza la resolución frecuencial de las descomposiciones diádica, lineal y MDCT frente a la escala Bark, y se evalúa experimentalmente la contribución de cada capa a la reducción de la tasa binaria, discutiendo los compromisos entre latencia, tasa y calidad perceptual.**

Palabras clave— **Intercomunicador de audio, tiempo real, baja latencia, doncencia, aprendizaje basado en proyectos.**

I. Introducción

La comunicación de voz en tiempo real sobre redes IP es una tecnología ampliamente utilizada en aplicaciones como la telefonía sobre Internet (VoIP) [1], las videoconferencias y los sistemas de intercomunicación profesional. Estos sistemas deben satisfacer requisitos estrictos de baja latencia, típicamente por debajo de 150 ms de extremo a extremo [2], y robustez frente a las variaciones del retardo de red (jitter), al tiempo que minimizan la tasa binaria consumida. Desde la perspectiva de la teoría tasa-distorsión (Rate-Distortion, RD) [3], el diseño de un intercomunicador de audio se puede formular como un problema de optimización en el que se busca minimizar la distorsión de reconstrucción sujeta a una restricción de tasa, o equivalentemente, minimizar la tasa para una distorsión máxima admisible.

Los códecs de audio más extendidos, como Opus [4] o Speex [5], ofrecen excelentes prestaciones pero están implementados en C/C++ y presentan una barrera de entrada elevada para su modificación y experimentación académica. En el ámbito educativo y de investigación resulta deseable disponer de un sistema modular, legible y fácilmente extensible que permita explorar de forma independiente cada etapa del proceso de codificación y transmisión de audio, incluyendo la cuantificación escalar, la codificación entrópica, las transformadas para decorrelación y la cuantificación perceptual basada en modelos psicoacústicos [6], [7].

En este artículo se presenta InterCom, un intercomunicador de audio full-duplex, de baja latencia, implementado íntegramente en Python [8]. InterCom se ha desarrollado como proyecto central de la asignatura *Tecnologías Multimedia* del Grado en Ingeniería Informática de la Universidad de Almería en la intensificación de Tecnologías de la Información, siguiendo la metodología de Aprendizaje Basado en Proyectos (ABP) [9]. Su diseño se basa en una arquitectura por capas donde cada módulo hereda del anterior y añade una funcionalidad específica, siguiendo el patrón de diseño *Template Method* [10]. Esta arquitectura permite aislar y estudiar individualmente cada técnica de codificación, de forma que cada capa constituye un hito (*milestone*) del proceso de aprendizaje.

Las principales contribuciones de este trabajo son:

1. Una arquitectura por capas que separa completamente las responsabilidades de transporte, buffering, codificación entrópica, cuantificación, control de tasa, decorrelación espacial, decorrelación temporal y cuantificación perceptual, proporcionando una progresión pedagógica que permite a los estudiantes modificar y experimentar con cada etapa de forma independiente.
2. La integración de la *Wavelet Packet Transform* (WPT) [11], [12] con solapamiento de chunks para eliminar artefactos de frontera, combinada con la *Mid/Side Transform* (MST) [7] para explotar la correlación inter-canal, junto con un análisis formal de sus propiedades de ortogonalidad y su impacto en la optimización RD independiente por sub-banda.
3. Un esquema de cuantificación perceptual basado en el umbral de audición (*Threshold of Hearing*, ToH) [6] que asigna pasos de cuantificación diferentes a cada sub-banda, con una discusión comparativa de la resolución frecuencial de las descomposiciones diádica (DWT), lineal (WPT) y MDCT frente a la escala Bark [7].
4. Una evaluación experimental que cuantifica la contribución de cada capa a la compresión mediante curvas RD, proporcionando a los estudiantes un marco para la validación empírica de las técnicas implementadas.

El resto del artículo se organiza como sigue. La

[1]Dpto. de Informática, Universidad de Almería, e-mails: savinspm@ual.es, juanjosemoreno@ual.es, vruiz@ual.es

Sección II revisa los fundamentos teóricos, incluyendo la teoría tasa-distorsión, la psicoacústica y las transformadas de análisis. La Sección III describe en detalle la arquitectura de InterCom y la progresión de sus capas. La Sección IV presenta las condiciones experimentales y los resultados obtenidos. Finalmente, la Sección V recoge las conclusiones y las líneas de trabajo futuro.

II. Fundamentos

A. Transmisión de audio sobre UDP

El protocolo UDP [13] proporciona un servicio de datagramas sin conexión idóneo para audio en tiempo real: no introduce retransmisiones que incrementarían la latencia y permite al receptor descartar paquetes que llegan fuera de plazo. Cada datagrama UDP transporta un *chunk* (fragmento de audio) completo, simplificando la sincronización. InterCom utiliza el modelo basado en interrupciones (*timer-based*) [14]: un *callback* se invoca periódicamente cada $t_{\text{chunk}} = F/f_s$ segundos, donde F es el tamaño de chunk en muestras y f_s la frecuencia de muestreo, garantizando una E/S de audio libre de *glitches*.

B. Ocultación del jitter

La latencia percibida por el usuario se puede modelar como [15], [16]:

$$t_u = t_p + t_i, \tag{1}$$

donde t_p es el tiempo de propagación de red (variable e incontrolable sin QoS) y t_i es la latencia introducida por InterCom (constante para una configuración dada). El jitter σ_p (variación de t_p) provoca que los paquetes lleguen con intervalos irregulares. Para absorberlo se emplea un buffer circular de acceso aleatorio (*dejitterizing buffer*) [15] indexado por el número de chunk. Cada paquete que llega se almacena en la posición chunk_number mód N_b, donde $N_b = 2B$ y B es el número de chunks de retardo configurable. La Calidad de Experiencia (QoE) es inversamente proporcional al jitter [17]: si B es demasiado pequeño, los paquetes llegan fuera de plazo; si es demasiado grande, la latencia total t_u supera el límite recomendado de 150 ms [2].

C. Teoría tasa-distorsión y cuantificación

La cuantificación es la principal fuente de distorsión en los sistemas de compresión con pérdidas [3], [18]. La cuantificación escalar uniforme con zona muerta (*dead-zone*) [3] reduce la precisión de las muestras dividiendo cada coeficiente por un paso de cuantificación Δ:

$$\hat{x} = \left\lfloor \frac{x}{\Delta} \right\rfloor, \qquad \tilde{x} = \hat{x} \cdot \Delta \tag{2}$$

donde $\hat{x}$ es el índice cuantificado y $\tilde{x}$ la reconstrucción. El ruido de cuantificación tiene una potencia [3]:

$$\sigma_q^2 = \frac{\Delta^2}{12} \tag{3}$$

que crece cuadráticamente con Δ. Un valor mayor de Δ produce menor tasa pero mayor distorsión, definiendo la *curva tasa-distorsión* (RD). Los cuantificadores con zona muerta producen más índices iguales a cero, lo que incrementa la ratio de compresión a costa de una mayor distorsión para coeficientes de baja magnitud.

D. Compresión DEFLATE y planos de bytes

DEFLATE [19] combina LZ77 con codificación Huffman para compresión sin pérdidas [20]. En InterCom, esta compresión se aplica por planos de bytes. Para ello, la señal de audio, representada con muestras de 24 bits (tras la cuantificación), se segmenta en tres planos correspondientes al byte más significativo (MSB1), al intermedio (MSB0) y al menos significativo (LSB). Cada plano se comprime independientemente, lo que resulta ventajoso porque los planos altos presentan menor entropía, especialmente durante las secuencias de audio silencioso [20].

E. Mid/Side Transform (MST)

La MST [7] decorrelaciona las señales estéreo $\mathbf{x} = [\mathbf{x_0}, \mathbf{x_1}]^{\mathrm{T}}$ (canales izquierdo y derecho) mediante la transformada lineal:

$$\mathbf{w} = \mathbf{K}\mathbf{x} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \mathbf{x_0} \\ \mathbf{x_1} \end{bmatrix} \tag{4}$$

donde $\mathbf{w_0} = \mathbf{x_0} + \mathbf{x_1}$ (señal *Mid*) y $\mathbf{w_1} = \mathbf{x_0} - \mathbf{x_1}$ (señal *Side*). La transformada inversa es:

$$\mathbf{x} = \mathbf{K^{-1}}\mathbf{w} = \frac{\mathbf{1}}{\mathbf{2}}\mathbf{K}^{\mathrm{T}}\mathbf{w} = \frac{\mathbf{1}}{\mathbf{2}}\mathbf{K}\mathbf{w} \tag{5}$$

La MST es una transformada **ortogonal** [21] (aunque no ortonormal), puesto que el producto interno de sus filtros es nulo: $\langle \mathbf{K_0}, \mathbf{K_1} \rangle = \langle [\mathbf{1}, \mathbf{1}], [\mathbf{1}, -\mathbf{1}] \rangle = \mathbf{1} \cdot \mathbf{1} + \mathbf{1} \cdot (-\mathbf{1}) = \mathbf{0}$. Esta propiedad es fundamental porque garantiza que las subbandas son estadísticamente independientes, permitiendo optimizar la curva RD de cada sub-banda de forma independiente [22]. La señal *Mid* concentra la mayor parte de la energía, mientras que *Side* contiene la diferencia estérea, que tiene menor varianza y se comprime más eficientemente.

F. Wavelet Packet Transform (WPT)

La WPT [11], [12] descompone la señal en 2^L subbandas de igual ancho de banda, donde L es el número de niveles de descomposición. Frente a la DWT diádica, que solo descompone las sub-bandas de baja frecuencia y genera $L+1$ sub-bandas (insuficientes para modelar el ToH con precisión). La WPT aplica recursivamente el banco de filtros de 2 canales tanto a las sub-bandas de baja como de alta frecuencia, ofreciendo una resolución frecuencial uniforme más adecuada para señales de audio [23]. InterCom utiliza por defecto la wavelet `db5` (Daubechies de orden 5) con $L = 6$ niveles, resultando en 64 sub-bandas, cada una con un ancho de banda de $f_s/(2 \cdot 2^L) = 344{,}5$ Hz.

Para evitar artefactos de frontera entre chunks consecutivos, se emplea una técnica de solapamiento

(*overlap-save*): cada chunk se extiende con muestras adyacentes, se aplica la WPT al chunk extendido, y se extraen únicamente los coeficientes correspondientes al chunk central [24].

G. Umbral de audición y escala Bark

El sistema auditivo humano (HAS, *Human Auditory System*) presenta una sensibilidad que depende de la frecuencia [6]. El umbral de audición (ToH) describe la presión sonora mínima (en dB SPL) perceptible, y para una persona de 20 años se aproxima mediante [7]:

$$T(f) = 3{,}64\left(\frac{f}{1000}\right)^{-0{,}8} - 6{,}5\,e^{-0{,}6\left(\frac{f}{1000}-3{,}3\right)^2} + 10^{-3}\left(\frac{f}{1000}\right)^{4} \tag{6}$$

donde f es la frecuencia en Hz.

La resolución frecuencial del HAS es finita y no lineal: la distancia mínima para distinguir dos tonos crece con la frecuencia. Este comportamiento se modela mediante la *escala Bark* [6], [7], que define bandas críticas cuyo ancho aumenta con la frecuencia. Combinando el ToH con la escala Bark, se puede asignar un paso de cuantificación Δ_k diferente a cada sub-banda k, de forma que la potencia del ruido de cuantificación $\sigma_k^2 = \Delta_k^2/12$ quede por debajo del umbral de percepción en cada banda crítica, logrando una compresión perceptualmente transparente.

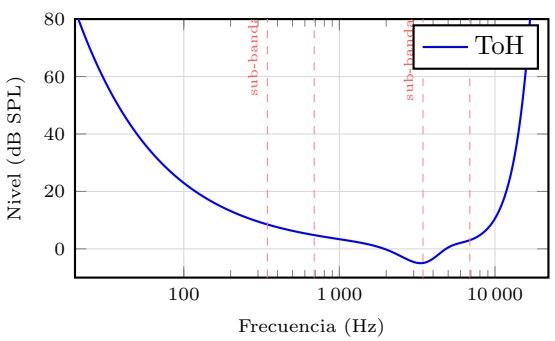


Fig. 1: Umbral de audición absoluto (ToH) en función de la frecuencia (Ecuación 6). Las líneas discontinuas rojas marcan los límites de algunas sub-bandas WPT ($\Delta f = 344{,}5$ Hz). La región de máxima sensibilidad (mínimo del ToH) se sitúa alrededor de 3–4 kHz.

H. Resolución frecuencial: DWT vs. WPT vs. MDCT

La elección de la transformada temporal determina la resolución frecuencial disponible para la cuantificación perceptual. La DWT diádica genera $N_{\text{DWT}} = L_{\text{DWT}} + 1$ sub-bandas con anchos que se duplican progresivamente, lo que proporciona pocas sub-bandas (típicamente ≤ 7) y no permite aproximar adecuadamente la escala Bark. La WPT genera $N_{\text{WPT}} = 2^{L_{\text{WPT}}}$ sub-bandas de ancho uniforme, lo que mejora significativamente la resolución pero no reproduce la distribución no lineal de la escala Bark. La MDCT (*Modified Discrete Cosine Transform*) [7] genera N sub-bandas para cada N muestras de entrada (gracias al solapamiento del 50 %), ofreciendo la mayor resolución frecuencial a costa de mayor complejidad computacional. La Tabla I resume las características de cada descomposición.

Tabla I: Comparación de resolución frecuencial de las transformadas temporales ($f_s = 44\,100$ Hz, $F = 1\,024$ muestras).

Transf.	Sub-bandas	Ancho mín.	Aprox. Bark
DWT ($L=6$)	7	344,5 Hz	Pobre
WPT ($L=6$)	64	344,5 Hz	Moderada
MDCT ($N=1024$)	1,024	21,5 Hz	Buena

III. Arquitectura de InterCom

InterCom se estructura como una pila de capas donde cada módulo Python hereda de la capa inferior y sobrecarga los métodos `pack()` y `unpack()` (y opcionalmente `analyze()`/`synthesize()`) para añadir su funcionalidad, siguiendo el patrón *Template Method* [10]. Cada capa constituye un hito (*milestone*) de la asignatura, de modo que los estudiantes abordan progresivamente las etapas del problema tasa-distorsión. La Tabla II resume las capas implementadas.

Tabla II: Capas de InterCom ordenadas de menor a mayor funcionalidad. Cada capa hereda de la anterior y constituye un hito de aprendizaje.

Capa	Módulo Python	Contribución
Minimal	`minimal.py`	Transporte UDP
Buffering	`buffer.py`	Ocultación jitter
DEFLATE Raw	`DEFLATE_raw.py`	Compresión lossless
DEFLATE BP3	`DEFLATE_byteplanes3.py`	Compresión por planos
BR_Control	`BR_control_*.py`	Cuantificación
Stereo MST	`stereo_MST_cod_32.py`	Decorrel. espacial
Temporal WPT	`temp_overlap_WPT.py`	Decorrel. temporal
ToH	`ToH_coding.py`	Cuantif. perceptual

A. Capa Minimal: transporte full-duplex

La capa base (`minimal.py`) implementa la estructura de procesamiento central, que sigue el algoritmo basado en interrupciones (*timer-based*). La biblioteca `sounddevice` [14] invoca un *callback* cada vez que el ADC dispone de un nuevo chunk, lo que proporciona una ejecución libre de *glitches* con tamaño de chunk constante. El flujo en cada interrupción es:

Listing 1: Pseudocódigo de la capa Minimal.

```
# Callback (ejecutado por interrupcion)
def record_send_and_play():
  chunk = record(F)           # (1) ADC -> chunk
  packed = pack(chunk)        # (2) Compresion
  send(packed)                # (3) UDP send
  chunk = unbuffer_next()     # (4) Buffer ->
      chunk
  play(chunk)                 # (5) chunk -> DAC

# Hilo principal
def receive_and_buffer():
  packed = receive()          # (1) UDP recv
  num, chunk = unpack(packed)# (2) Descompresion
  buffer(num, chunk)          # (3) Almacenar
```

Nótese que, aunque PortAudio invoca el *callback* desde un hilo de audio propio, el GIL (*Global Interpreter Lock*) de Python serializa la ejecución. La recepción y almacenamiento en el buffer se ejecuta en el hilo principal con la misma cadencia que las interrupciones, de modo que los chunks recibidos por el kernel se extraen a tiempo y no se pierden paquetes.

Esta arquitectura no requiere crear hilos ni procesos explícitos (un único proceso Python es visible en `htop`).

La latencia mínima teórica de esta capa es un chunk: $t_{\text{chunk}} = F/f_s = 1024/44100 \approx 23{,}2$ ms.

B. Capa Buffering: ocultación del jitter

La clase `Buffering` hereda de `Minimal` e introduce un buffer circular de acceso aleatorio (véase Sección II-B). Cada paquete incluye un número de secuencia de 16 bits como índice, lo que permite:

- Absorber el jitter sin necesidad de ordenar paquetes.
- Desacoplar recepción y reproducción: los chunks perdidos se sustituyen por chunks de silencio (*zero-chunks*), garantizando una reproducción sin discontinuidades (*gapless*).

C. Capas de compresión DEFLATE

La clase `DEFLATE_Raw` sobrecarga `pack()`/`unpack()` para aplicar `zlib.compress()`/`decompress()` al chunk completo. La clase derivada `DEFLATE_BytePlanes3` mejora la compresión segmentando cada muestra en tres planos de bytes (véase Sección II-D). La cabecera del paquete incluye las longitudes de los planos comprimidos.

La separación por planos explota el hecho de que los planos más significativos contienen carreras de ceros (*runs of zeros*), especialmente durante secuencias de audio silencioso.

D. Capa de cuantificación y control de tasa

La clase `BR_Control_No` introduce la cuantificación con zona muerta (Ecuación 2). La clase derivada `BR_Control_Conservative` implementa un control adaptativo de tasa mediante AIMD (*Additive Increase / Multiplicative Decrease*): si se detectan más de 2 paquetes perdidos en un periodo de observación (por defecto 1 s), Δ se duplica; en caso contrario, se reduce multiplicándolo por 1/1,1, con un mínimo configurable $\Delta_{\text{mín}}$. Esto permite adaptar la compresión a la capacidad instantánea del enlace simulando el comportamiento de los mecanismos de control de congestión [17].

E. Capa de decorrelación espacial (MST)

La clase `Stereo_MST_Coding_32` aplica la MST (Ecuación 4) en aritmética de 32 bits para evitar desbordamientos. Las operaciones `analyze()` y `synthesize()` implementan respectivamente las ecuaciones directa e inversa de la MST. Al ser una transformada ortogonal (véase Sección II-E), la decorrelación de las sub-bandas permite que la codificación entrópica posterior opere de forma óptima sobre cada sub-banda de manera independiente, lo que constituye una propiedad clave para la optimización RD en bancos de filtros [22].

F. Capa de decorrelación temporal (WPT con solapamiento)

La clase `Temporal_Overlapped_WPT` combina la WPT (Sección II-F) con solapamiento de chunks. El procedimiento en `analyze()` es:

1. Se aplica la MST al chunk de entrada.
2. Se almacena el chunk MST en un buffer deslizante de 3 posiciones.
3. Se construye el chunk extendido concatenando O muestras del chunk anterior, el chunk completo y O muestras del siguiente, con $O = l_{\text{máx}} \cdot 2^L$, donde $l_{\text{máx}}$ es la longitud máxima de los filtros de la wavelet.
4. Se aplica la WPT al chunk extendido usando `PyWavelets` [25], obteniendo $2^L = 64$ sub-bandas.
5. De cada sub-banda se extraen los $F/2^L$ coeficientes centrales, descartando los bordes afectados por efectos de frontera.

La descomposición WPT lineal genera sub-bandas de ancho uniforme $\Delta f = f_s/(2 \cdot 2^L) = 344{,}5$ Hz. Aunque esta partición no reproduce la distribución logarítmica de la escala Bark, proporciona suficiente resolución para una cuantificación perceptual efectiva con 64 sub-bandas (véase Sección II-H).

G. Capa de cuantificación perceptual (ToH)

La clase `ToH` hereda de `Temporal_Overlapped_WPT` y sobrecarga `quantize()` y `dequantize()` para utilizar un paso de cuantificación diferente por sub-banda, Δ_k. Los pasos se calculan durante la inicialización de forma que la potencia del ruido de cuantificación en cada sub-banda k se mantenga por debajo del umbral de percepción [6], [7]:

1. Para cada una de las 2^L sub-bandas WPT se calcula el nivel de presión sonora medio del ToH (Ecuación 6) integrando sobre el rango de frecuencias de la sub-banda.
2. Los niveles se normalizan para que el mínimo sea cero y se suman al paso base $\Delta_{\text{mín}}$.
3. El vector resultante Δ_k, $k = 1, \ldots, 2^L$, se replica para cubrir todos los coeficientes del chunk.

De esta forma, las sub-bandas alrededor de 3–4 kHz (máxima sensibilidad auditiva, véase Ecuación 6) reciben los Δ_k más pequeños, mientras que las sub-bandas por debajo de 100 Hz y por encima de 15 kHz se cuantifican más agresivamente. Nótese que el ToH varía entre individuos [7] (edad, sexo, exposición al ruido), lo que motiva la extensión hacia curvas ToH personalizadas (véase Sección V).

IV. Resultados experimentales y uso docente

A. Evaluación del sistema

Los experimentos se han realizado en modo *loopback* (`localhost`) con señales estéreo en formato OGA y Opus de diversa naturaleza a $f_s = 44\,100$ Hz, $F = 1\,024$ muestras y 2 canales. Las versiones

`*__verbose` de cada capa registran métricas con periodo de 1 s: tasa binaria (R), SNR, entropía (H) y RMSE. Adicionalmente, se ha empleado `tc` [26] para simular jitter y limitación de ancho de banda.

La Tabla III resume el efecto acumulativo de las capas sobre la tasa binaria media, partiendo de la tasa PCM sin comprimir $R_0 = 1\,411$ kbps.

Tabla III: Efecto acumulativo de las capas sobre la tasa binaria (valores orientativos para una señal musical típica).

Capa activa	R **(kbps)**	**Ratio**
Sin comprimir (PCM)	1 411	1:1
DEFLATE BytePlanes3	∼750	1,9:1
+ Cuantif. ($\Delta = 128$)	∼200	7:1
+ MST + WPT solapada	∼130	10,9:1
+ ToH	∼90	15,7:1

La cuantificación escalar aporta la mayor reducción individual (73 %), confirmando las predicciones de la teoría tasa-distorsión [3]. La cuantificación perceptual basada en el ToH añade un 31 % adicional explotando la menor sensibilidad del oído en las subbandas de baja y alta frecuencia. La latencia total en modo loopback es $t_{\text{total}} = t_{\text{chunk}} + t_{\text{buffer}} \approx 190$ ms (con $F = 1\,024$ y $B = 7$ chunks de retardo), dentro del límite de la ITU-T G.114 [2].

B. Organización de las prácticas

InterCom se utiliza como proyecto vertebrador de la asignatura *Tecnologías Multimedia* del Grado en Ingeniería Informática de la Universidad de Almería. Los estudiantes, organizados en grupos de hasta 4 personas, desarrollan el proyecto a lo largo de 15 sesiones de 2 horas (30 horas presenciales), más el trabajo autónomo correspondiente. La Tabla IV detalla la distribución temporal de los hitos.

Tabla IV: Distribución de sesiones prácticas y hitos de InterCom.

Sesiones	**Hito**	**Competencias**
1–2	Minimal + Buffering	Sockets UDP, jitter, latencia
3–5	DEFLATE + BytePlanes	Codificación entrópica
6–8	Cuantif. + BR Control	Teoría RD, control de tasa
9–11	MST estéreo	Decorrelación espacial
12–13	WPT solapada	Decorrelación temporal
14–15	ToH perceptual	Psicoacústica, umbral audición

Cada hito sigue una dinámica ABP [9] estructurada en tres fases:

1. **Estudio autónomo**: los estudiantes revisan la documentación teórica y el código de la capa anterior, disponible en el repositorio público del proyecto.
2. **Implementación**: cada grupo implementa la nueva capa heredando de la anterior y sobrecargando los métodos `pack()`/`unpack()`.
3. **Presentación y evaluación**: cada grupo presenta su solución al resto de la clase, mostrando las métricas obtenidas con la versión `*__verbose` y realizando una demostración en directo. La evaluación incluye tanto la calidad del código como la capacidad de explicar las decisiones de diseño.

La herramienta `tc` [26] permite a los estudiantes simular condiciones de red realistas (jitter, pérdida de paquetes, limitación de ancho de banda) en el laboratorio, lo que les permite observar directamente el efecto de cada capa sobre la QoE. Por ejemplo, al activar la capa de buffering bajo jitter simulado, los estudiantes comparan la reproducción con y sin buffer, apreciando el compromiso entre latencia y robustez.

C. Instrumentación para la validación

Las versiones `*__verbose` de cada módulo constituyen una herramienta didáctica esencial: muestran en tiempo real la tasa binaria, la SNR y la entropía, permitiendo a los estudiantes verificar experimentalmente los conceptos teóricos. Por ejemplo, al añadir la capa de cuantificación, observan cómo la reducción de Δ incrementa simultáneamente la tasa y la SNR, materializando la curva RD estudiada en clase. La evaluación subjetiva de la calidad de audio percibida complementa las métricas objetivas, siguiendo una escala de cinco niveles (*Perfecto*, *Bueno*, *Aceptable*, *Pobre*, *Inaceptable*).

D. Impacto en la calidad docente

Desde la adopción del ABP, la tasa de aprobados ha aumentado en aproximadamente un 25 % respecto a la evaluación mediante examen, superando el 90 % en los últimos cursos. Paralelamente, el número de estudiantes matriculados ha crecido de forma sostenida: desde los 24 del primer año de implantación (2007) hasta los 103 en 2024 y los 101 en 2025, lo que constituye un indicador indirecto del atractivo percibido de la propuesta pedagógica.

V. Conclusiones y trabajo futuro

En este artículo se ha presentado InterCom, un intercomunicador de audio en tiempo real implementado en Python y diseñado como herramienta docente para la enseñanza de Tecnologías Multimedia mediante ABP. Su arquitectura modular por capas permite reducir la tasa binaria de una señal estéreo PCM desde 1 411 kbps hasta aproximadamente 90 kbps (ratio 15,7:1), manteniendo una latencia inferior a 190 ms y una calidad perceptual subjetivamente aceptable.

Desde el punto de vista docente, la experiencia acumulada a lo largo de varios cursos académicos en la Universidad de Almería permite extraer las siguientes conclusiones:

- La progresión por capas/hitos proporciona una estructura natural para el ABP: cada hito tiene un alcance bien definido, es evaluable de forma independiente y conecta directamente con un concepto teórico específico (codificación entrópica, teoría RD, psicoacústica, etc.).
- La implementación en Python reduce significativamente la barrera de entrada respecto a códecs industriales en C/C++, permitiendo que los estudiantes se concentren en los conceptos de com-

presión y transmisión en lugar de en detalles de bajo nivel del lenguaje.

- Las versiones `*_verbose` y la herramienta `tc` proporcionan una instrumentación accesible que permite a los estudiantes verificar experimentalmente los conceptos teóricos y desarrollar intuición sobre el compromiso tasa-distorsión.
- Las presentaciones por grupos fomentan las competencias transversales de comunicación y trabajo en equipo, y permiten a cada grupo beneficiarse de las soluciones alternativas exploradas por otros compañeros.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación. Programa Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica (Comp4Health-II, PID2024-155521OB-I00). Savíns Puertas-Martín es beneficiario del Programa de Formación, Captación, Incorporación y Movilidad de los Recursos Humanos de I+D+I de la Estrategia EIDIA, Horizonte 2027, financiado por la Consejería de Universidad, Investigación e Innovación de la Junta de Andalucía y cofinanciado por el Programa FSE+ de Andalucía 2021-2027.

Referencias

[1] A. B. Johnston, "SIP: Understanding the session initiation protocol," 2009.
[2] ITU-T, "Recommendation G.114: One-way transmission time," Tech. Rep., International Telecommunication Union, 2003.
[3] K. Sayood, *Introduction to Data Compression*, Morgan Kaufmann, 5th edition, 2017.
[4] J.-M. Valin, K. Vos, and T. B. Terriberry, "Definition of the Opus audio codec," *RFC 6716, IETF*, 2012.
[5] J.-M. Valin, "The Speex codec manual," `https://www.speex.org/docs/manual/speex-manual/`, 2007.
[6] E. Zwicker and H. Fastl, *Psychoacoustics: Facts and Models*, Springer, 3rd edition, 2007.
[7] M. Bosi and R. E. Goldberg, *Introduction to Digital Audio Coding and Standards*, Springer, 2003.
[8] Python Software Foundation, "Python language reference, version 3," `https://www.python.org`, 2024.
[9] J. E. Mills and D. F. Treagust, "Engineering education — is problem-based or project-based learning the answer?," *Australasian Journal of Engineering Education*, vol. 3, no. 2, pp. 2–16, 2003.
[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.
[11] S. Mallat, *A Wavelet Tour of Signal Processing: The Sparse Way*, Academic Press, 3rd edition, 2009.
[12] M. Vetterli and J. Kovačević, *Wavelets and Subband Coding*, Prentice Hall, 1995.
[13] J. Postel, "User datagram protocol," RFC 768, IETF, 1980.
[14] M. Geier, "python-sounddevice," `https://python-sounddevice.readthedocs.io/`, 2024.
[15] C. Perkins, *RTP: Audio and Video for the Internet*, Addison-Wesley, 2003.
[16] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, Pearson, 5th edition, 2011.
[17] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, Pearson, 7th edition, 2017.
[18] D. S. Taubman and M. W. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards and Practice*, Springer, 2002.
[19] P. Deutsch, "DEFLATE compressed data format specification version 1.3," RFC 1951, IETF, 1996.
[20] M. Nelson and J.-L. Gailly, *The Data Compression Book*, M&T Books, 2nd edition, 1995.
[21] K. Sayood, *Introduction to Data Compression*, Morgan Kaufmann, 5th edition, 2017.
[22] M. Vetterli, J. Kovačević, and V. K. Goyal, *Foundations of Signal Processing*, Cambridge University Press, 2014.
[23] M. V. Wickerhauser, *Adapted Wavelet Analysis from Theory to Software*, A K Peters, 1994.
[24] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Pearson, 3rd edition, 2010.
[25] G. R. Lee, R. Gommers, F. Waselewski, K. Wohlfahrt, and A. O'Leary, "PyWavelets: Wavelet transforms in Python," `https://pywavelets.readthedocs.io/`, 2024.
[26] B. Hubert, *Linux Advanced Routing & Traffic Control*, 2012.

Entorno para el modelado y simulación de sistemas electrónicos digitales

José-Antonio Verde-Jiménez[1], Álvaro Guerrero-Espinosa[1], Diego Camarmas-Alonso[1], Félix García-Carballeira[1], Alejandro Calderón-Mateos [1]

Resumen— **El objetivo de este trabajo es desarrollar un nuevo modelo simplificado y fácil de usar para la especificación y simulación de circuitos electrónicos digitales. Para ello, se propone la creación de un entorno web y el diseño de una API que permita esta especificación. Esta API permite definir circuitos de manera natural mediante el establecimiento de las conexiones que unen los distintos componentes dando lugar a un grafo que representa la estructura del circuito.**

El sistema diferencia entre componentes combinacionales, secuenciales (con memoria) y grupos de componentes. La API define una serie de componentes primitivos que engloben las funcionalidades más comunes como por ejemplo un banco de registros, una memoria RAM, puertas lógicas, sumadores o multiplicadores, etc. entre otros.

El grafo obtenido tiene distintas restricciones sobre su naturaleza, como por ejemplo, ciclos existentes y cómo se conectan los pines de los componentes. A partir de este grafo se genera un grafo acíclico dirigido complementario donde las señales (conexiones entre pines) y los componentes (puertas lógicas) son los nodos; y donde las aristas indican la dirección del flujo de datos. A partir de este grafo acíclico dirigido se aplica un planificador que utiliza ordenamiento topológico para encontrar un orden de evaluación de las puertas lógicas. Y a partir de este ordenamiento se genera código ejecutable en TypeScript para su ejecución.

Para validar el modelo propuesto se presenta como caso uso el desarrollo de un sumador de 10.000 bits.

Palabras clave— **Simulador, circuitos digitales, HDL.**

I. Introducción

La motivación principal es poder definir un procesador elemental microprogramable que se pueda integrar al simulador educativo CREATOR[1] de una forma modular y sencilla, pero a la vez suficientemente completa y flexible [1].

Este proyecto parte del procesador elemental definido por WepSIM [2], una herramienta educativa para la enseñanza de microprogramación y el funcionamiento de un procesador por etapas [3]. Este procesador elemental se ha conseguido definir en una FPGA [4] recientemente.

La experiencia positiva de usar un procesador microprogramado en WepSIM se busca poder añadirla al simulador CREATOR, de manera que tenga un motor de ejecución basado en un circuito electrónico simulado (además de basado en comportamiento de instrucciones descritas como código).

[1]Universidad Carlos III de Madrid, ROR: `https://ror.org/03ths8210`, Departamento de Informática, Avenida de la Universidad, 30 (edificio Sabatini), 28911 Leganés (Madrid), España. e-mail: `jverde@pa.uc3m.es, {alvarogu,dcarcama,fgcarbal,acaldero}@inf.uc3m.es`

[1]`https://creatorsim.github.io/`

Pero se ha querido aprovechar para dar un salto cuantitativo más: en lugar de describir el circuito en código JavaScript como un vector de objetos en el caso de WepSIM, pasar a poder definir el circuito de una forma suficientemente simple pero basado en un modelo suficientemente completo como para generar el código que representa la simulación de ese circuito.

Se ha decidido utilizar el lenguaje de programación TypeScript, por su fuerte sistema de tipos, que es capaz de capturar gran cantidad de errores en tiempo de compilación y por sus altos niveles de comprensibilidad y mantenibilidad [5]. Además de que permite su uso tanto en web (transpila a JavaScript) como en escritorio. Además, se ha decidido utilizar `bun` como gestor de paquetes y entorno de desarrollo.

Además, se ha decidido separar la aplicación en distintas bibliotecas individuales, que limiten el número de dependencias y que permitan ejecutar casos de prueba de manera independiente, en lugar de tener una arquitectura monolítica. A fecha de publicación de este artículo estos repositorios se encuentran alojados en Codeberg: `creator-wepsim-core` [2], que contiene la descripción del los componentes y primitivas básicas; y `creator-wepsim-execution` [3], que contiene la parte de compilación y ejecución.

II. Estado de la cuestión

Un lenguaje de descripción de *hardware*, o HDL por sus siglas en inglés (*Hardware Description Language*), es una clase general de lenguajes cuyo dominio no es la descripción de algoritmos para su ejecución en sistemas de cómputo, sino la descripción de los propios sistemas de cómputo [6]. Los dos principales lenguajes estandarizados para la descripción del *hardware* son VHDL (IEEE estándar 1076 de 1993 con última revisión en 2019) [6] y Verilog (IEEE estándar 1800 de 1989 con última revisión en 2023) [7], que son textuales [8]. Hay también lenguajes de descripción de *hardware* no estandarizados que lo tratan desde un punto de vista educativo [9] o bien desde un punto de vista del flujo de datos [10]. Hay propuestas basadas en paso de mensajes y un diseño síncrono frente a la asincronicidad de VHDL o Verilog [11], e incluso basadas en lenguajes de programación funcional [12].

Sin embargo, existen otras propuestas que tratan de ofrecer una perspectiva diferente al proceso de descripción de circuitos digitales. Por ejemplo, se ha

[2]`https://codeberg.org/joseaverde/creator-wepsim-core`

[3]`https://codeberg.org/joseaverde/creator-wepsim-execution`

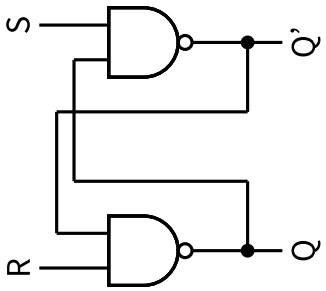


Fig. 1: Biestable asíncrono formado por dos compuertas NAND. Cuando S y R están activadas al mismo tiempo, el resultado en Q es el valor que estaba almacenado en memoria. Si solo está S activada, el valor en memoria pasa a ser 1; y si solo está activada R, pasa a a ser 0. Q' da el valor de Q negado.

probado a añadir a un entorno de desarrollo integrado para la descripción de *hardware*, un diagrama que muestra la representación del circuito digital y que se actualiza en vivo a la vez que el desarrollador programa [13].

Dicho enfoque está basado a su vez en otro estudio llamado *Polymorphic Blocks* [14], que utiliza un lenguaje de descripción de *hardware* integrado con el lenguaje de programación Python al que añaden polimorfismo. Es una aproximación que ofrece las ventajas de la reusabilidad y la encapsulación de la programación orientada a objetos. Sin embargo, está centrada principalmente a diseño placas, utilizando principalmente componentes de alto nivel [14].

III. Diseño

En un circuito electrónico se pueden diferenciar dos clases principales de componentes: secuenciales y combinacionales. Los componentes secuenciales se caracterizan por tener una memoria donde almacenan una serie de valores, que, si son síncronos, solo se puede actualizar o en flanco de subida o en flanco de bajada del reloj. Los componentes combinacionales, en cambio, no tienen una memoria intermedia y ejecutan asíncronamente.

Nótese que en la práctica, los componentes secuenciales se definen a partir de componentes combinacionales. Por ejemplo, un biestable, que es una memoria de un bit, se puede definir a partir de dos puertas NOR o dos puertas NAND, que se retroalimentan mutuamente (véase la Figura 1). Sin embargo, se ha decidido que la diferencia entre estos tipos de componentes sea explícita y se prohíba los ciclos en el propio circuito, en el sentido de un grafo cíclico. Así pues, los componentes secuenciales comunes, como registros o memorias, están disponibles como primitivas.

El sistema diferencia tres tipos de componentes: secuenciales, combinacionales y grupos de componentes, que siguen el patrón de diseño visitante. Todos los componentes secuenciales tienen una entrada implícita de un reloj común, y se pueden configurar el momento en el se quiere que se actualice: en flanco de subida o en flanco de bajada.

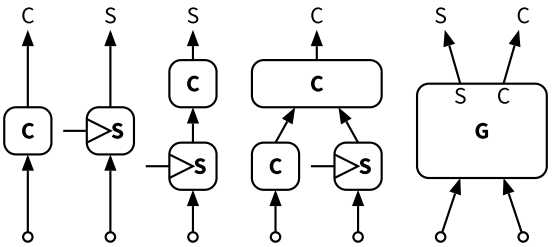


Fig. 2: Relación de secuencialidad de los pines de entrada con los pines de salida dependiendo del tipo de componentes que hay en su camino. La imagen se puede ver como un gran grupo con siete pines de entrada y seis pines de salida.

A. Grafo de componentes

Internamente los grupos de componentes se representan como un grafo dirigido. Una simplificación que se hace es que los pines sean o bien de entrada o bien de salida. El grafo se forma al conectar un pin de salida de un componente con el pin de entrada de otro, a esta conexión la denominamos señal y su sentido indica el sentido del flujo de datos.

Este grafo solo puede tener ciclos si hay una memoria entre medias, es decir, un componente secuencial. El objetivo es encontrar una forma de evaluar el grafo y dar un valor a las señales de manera secuencial. No todas las señales tienen un punto fijo y hay ciclos que tienen valores no estables, cuyo propósito principal sería el de implementar relojes o memorias, funciones que ya vienen dadas por las primitivas.

El grupo de componentes tiene una «interfaz» con pines de entrada y pines de salida, como si fuera un componente más. Se valida que los pines de la interfaz, estén conectados a señales dentro del grupo.

Un problema fundamental sobre los grupos, es si son secuenciales o si son combinacionales. Esto es para evitar bucles entre componentes combinacionales que estén dentro del mismo grupo. Se ha observado que la solución está en tratar la secuencialidad como una relación entre todos los pines de entrada para cada pin de salida como se ve en la Figura 2. Si hay al menos un componente combinacionales antes de un pin de salida, este se considera no secuencial.

A la hora de validar el grafo, se comprueba que los pines de entrada coincidan con los de salida; que todos los pines de entrada estén conectados y que las conexiones entre los pines tengan la misma anchura en bits. Adicionalmente, se utiliza un algoritmo en búsqueda en profundidad implementado sin recursión (porque hay entornos de ejecución web que tienen un límite de recursión) para la detección de ciclos.

Como se muestra en la Figura 2, internamente se consideran dos tipos de aristas, aristas fuertes y aristas débiles. Las aristas fuertes son las que conectan la salida de un pin combinacional; y las débiles, de un pin secuencial. A la hora de realizar la búsqueda en profundidad, no se exploran las aristas débiles, solamente las fuertes. Se considera ciclo si y solo si existe al menos un camino que parta y termine en el mismo nodo (teniendo en cuenta que no se siguen las aristas débiles).

B. Grafo de señales

A la hora de visualizar y simular el circuito, hay que tener en cuenta que son las señales las que tienen un valor (alto, bajo, alta impedancia o desconocido) y no los propios pines. Para ello, el grafo se transforma en un grafo acíclico dirigido, donde las señales son nodos además de los componentes. A este grafo se le denominará "grafo de señales". Los componentes se subclasifican en memorias y "combinadores".

Los valores lógicos son cuatro 1 (alto), 0 (bajo), Z (alta impedancia) o X (desconocido). Se utilizará $\mathcal{L} = \{0, 1, Z, X\}$ para denominar al conjunto de estos cuatro valores lógicos. Estos cuatro valores están inspirados de Verilog, que usa estos cuatro tipos de valores; y VHDL, que usa nueve (U, X, 0, 1, Z, W, L, H y $-$) de acuerdo con el IEEE estándar 1164 de 1993 [15]. En comparación con VHDL, esta propuesta considera equivalentes los siguientes:

- X: X (desconocido fuerte), W (desconocido débil), U (sin inicializar) y $-$ (*don't care*).
- Z: Z (alta impedancia).
- 0: 0 (0 fuerte) y L (0 débil).
- 1: 1 (1 fuerte) y H (1 débil).

Internamente estos valores se codifican en una clase llamada `HeavyInt`, que contiene dos valores: `value` y `mask`. Los bits en la posición n-ésima de ambos valores describen el tipo de valor lógico:

- X: `mask=0`, `value=0` (valor por defecto).
- Z: `mask=0`, `value=1`.
- 0: `mask=1`, `value=0`.
- 1: `mask=1`, `value=1`.

B.1 Componentes combinacionales

Un componente combinacional que tiene n entradas y m salidas se puede ver como una función $f : \mathcal{L}^n \to \mathcal{L}^m$ que se aplica a las n entradas para obtener las m salidas. El grafo acíclico dirigido tiene como raíces n señales de entrada, cada una de ellas representa en memoria una celda de un valor lógico. Estos nodos de entrada están conectados a un nodo que se va a denominar "combinador". Finalmente el combinador está conectado a los m nodos de salida. Como se observa en la Figura 3.

B.2 Componentes secuenciales

Los componentes secuenciales son especiales en el sentido en que se pueden dividir en dos. Dado un componente secuencial de n entradas, m salidas con k elementos en memoria, la salida del componente secuencial solo depende de lo que tenía en memoria, es una función: $f_o = \mathcal{L}^k \to \mathcal{L}^m$. El valor de dentro solo se actualiza en flanco de subida o en flanco de bajada, con una función del tipo: $f_i = \mathcal{L}^n \to \mathcal{L}^k$. La señal de reloj es implícita para todos los componentes secuenciales.

Por lo tanto, las salidas del componente secuencial se evalúan al principio, y sus entradas se actualizan al final después de haber evaluado el resto de señales como se ve en la Figura 4. Esto nos permite eliminar

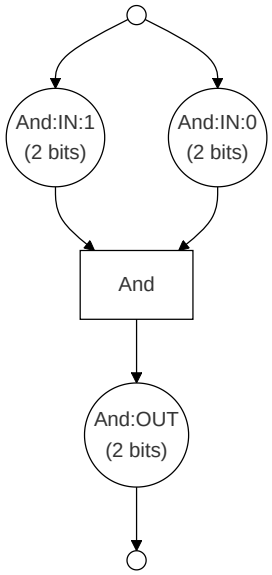


Fig. 3: Ejemplo de componente combinacional. Un puerta lógica AND de 2 bits. Tiene dos señales de entrada de dos bits cada una. Y tiene una señal de salida de dos bits. En el centro está el combinador *And* que ejecuta la función $f : \mathcal{L}^2 \to \mathcal{L}^2$ definida como $f([l_0, l_1], [r_0, r_1]) := [l_0 \wedge r_0, l_1 \wedge r_1]$.

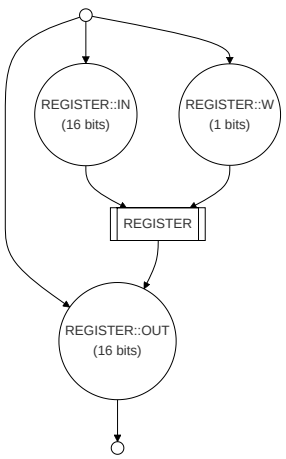


Fig. 4: Ejemplo de componente secuencial. Un registro de 16 bits que tiene dos señales entrada: una de 16 bits con el valor a almacenar, y otra de un bit para indicar cuándo escribir. El grafo acíclico dirigido empieza además en la salida del componente, porque las salidas de las memorias siempre retornan el valor del ciclo anterior. La arista que conecta *REGISTER* con la señal de salida se ejecuta al final de cada medio ciclo cuando coincide el flanco de reloj con el configurado para la actualización.

los ciclos que hay en los componentes secuenciales y encontrar un orden de evaluación válido para el componente.

B.3 Grupos de componentes

El grafo de un grupo de componentes se forma al unir los grafos generados para cada uno de los subcomponentes mediante las aristas nodo de salida a nodo de entrada. El grupo de componentes tiene una interfaz externa de pines que son visibles al usuario. Por ejemplo un sumador completo tiene tres entradas: operando primero, operando segundo y el

acarreo de la operación anterior; y dos salidas: el resultado y el acarreo.

Recursivamente se generan los grafos de cada uno de los subcomponentes que forman el grafo de componentes. A continuación, para cada una de las conexiones, se genera una arista que parte del pin de salida y se conecta a un nodo componente con la función identidad. De dicho nodo parte una arista que se conecta a un nodo señal, del que sale una última arista que se conecta al nodo señal del componente que referencia al pin de entrada. Se realizan conexiones adicionales, para simplificar la generación del grafo. La Figura 5 muestra el grafo generado para un sumador completo de un bit, equivalente al circuito del sumador completo de la Figura 6.

C. Compilación

Antes de compilar, se hace un ordenamiento topológico para encontrar un orden de evaluación. El ordenamiento topológico se hace con una búsqueda en profundidad que parte de los nodos raíz siguientes: componentes que no tienen entrada, los pines de entrada del componente, y la salida de las memorias.

A continuación, el grafo de señales pasa en orden topológico las señales y componentes a un objeto que implementa la interfaz de compilador. Este objeto, registrará las distintas señales y componentes y las relaciones entre ellos y retornará un objeto que implementa la interfaz ejecutor.

De momento, para la compilación se ha implementado un único *backend*, que utiliza el patrón visitante que estructura los componentes primitivos para retornar una función lambda que tenga la misma semántica que el componente dado. A continuación se concatenan todas las funciones lambdas generadas y se prepara para la ejecución.

D. Evaluación

Para la evaluación se ha decidido separar cada ciclo de reloj en dos medios ciclos. El primer medio ciclo tiene la señal de reloj baja y al final del mismo se ejecuta la función de actualización de las memoria configuradas por flanco de subida. Del mismo modo, el segundo medio ciclo simula la señal de reloj alta, y al final de dicho ciclo se actualizan las memorias por flanco de bajada.

Se ha decidido que el reloj sea una señal implícita de todos los componentes primitivos de tipo secuencial, porque simplifica el diseño el no tener varios relojes asíncronos trabajando sobre el mismo circuito. De todas formas, siempre es posible definir relojes auxiliares a partir de registros, que permitan ejecutar a frecuencias menores (fracciones de la frecuencia original).

Cada vez que se evalúa un medio ciclo, cada instrucción puede retornar un estado de error. La función de evaluación retorna una posible lista de errores con los componentes afectados y el tipo de error. Esto permite modelizar el concepto de cortocircuito si se define un bus de datos como una función con n entradas de b bits cada una y una salida de n bits, de modo que: si todas las entradas son altas impedancias, el resultado es una alta impedancia; si hay una única señal que no es alta impedancia, el resultado es el valor de de dicha señal; si no, el resultado es un cortocircuito.

Antes de ejecutar hay que configurar los valores de las entradas del circuito completo. Al final del ciclo de ejecución se pueden leer los valores de las salidas del mismo. Adicionalmente, si los componentes que conecta una señal tienen un identificador único, se puede obtener el valor que tuvo dicha señal a la hora de ejecutar el medio ciclo. Esto permitiría tener un visualizador en vivo que muestre los valores lógicos de cada conexión.

IV. Evaluación

A. Pruebas realizadas

Para la evaluación se ha decidido realizar un estudio de la escalabilidad del sistema. Para ello se va a utilizar como base el circuito de un sumador de n bits, que se ha creado a partir de puertas lógicas elementales. Un sumador completo (véase la Figura 6, es un sumador de un bit de anchura que tiene tres entradas de un bit: A (primer operando), B (segundo operando), C (acarreo de la operación previa); y tiene dos salidas de un bit *Result* (el resultado de la suma) y *Carry* (el acarreo de la suma). Si se colocan n sumadores completos con la distribución que muestra la Figura 7, se puede obtener un sumador de n bits.

El código que se ha utilizado para describir el circuito de la Figura 6 usando la API de TypeScript se puede ver en los siguientes listados. En el Listado 1, se importa el módulo y se describen las puertas lógicas, cada una de las cuales puede tener un identificador que se puede utilizar después para referirse a ellas. A continuación, en el Listado 2: en la constante `pins` se define la interfaz del grupo de componentes; y `internal`, con qué pines están conectados internamente. Luego, en el Listado 3, se describen las conexiones entre los distintos pines. Después, en el Listado 4, se crea, valida y compila el grupo de componentes. Finalmente, en el Listado 5, se inicia el valor por defecto de las memorias (no hay memorias en este caso) y se preparan las señales para la ejecución; se ejecuta un medio ciclo y se observan los valores resultantes.

```
import * as Core from "creator-wepsim-core";

const xor = new Core.ReductionGateComponent
    (3, Core.GateKind.XOR, "XOR");
const or = new Core.ReductionGateComponent
    (3, Core.GateKind.OR, "OR");
const and1 = new Core.GateComponent(1, Core.
    GateKind.AND, "AND::1");
const and2 = new Core.GateComponent(1, Core.
    GateKind.AND, "AND::2");
const and3 = new Core.GateComponent(1, Core.
    GateKind.AND, "AND::3");
const join1 = new Core.JoinComponent<3>(3,
    [1, 1, 1], "JOIN-RESULT");
const join2 = new Core.JoinComponent<3>(3,
    [1, 1, 1], "JOIN-CARRY");
const fork_a = new Core.ForkComponent<1,
    3>(1, 3, "FORK-A");
```

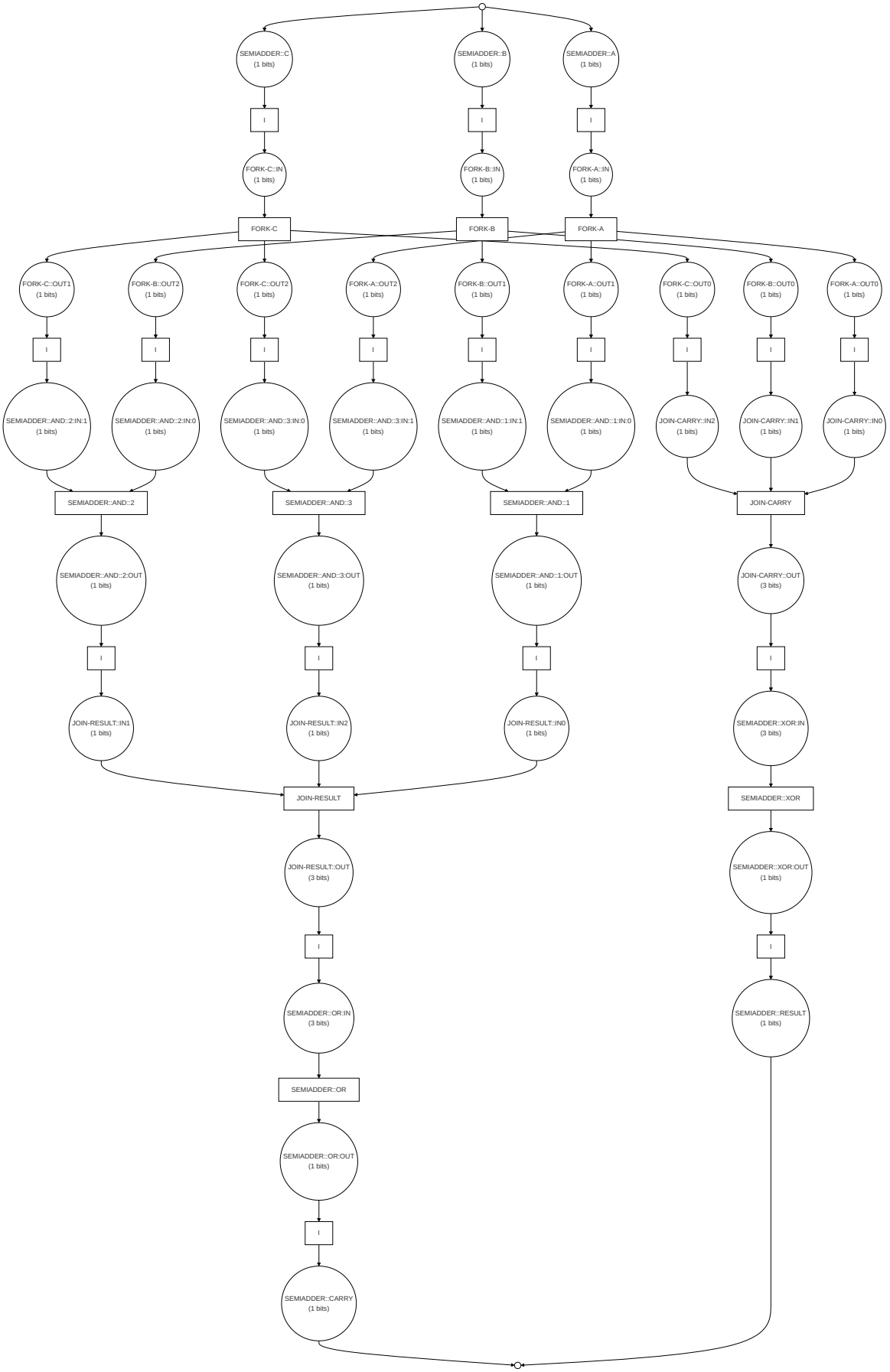


Fig. 5: Ejemplo de grupo de componentes. Sumador completo de un bit. Para cada cada conexión entre dos subcomponentes se añaden siempre una señal y una función identidad.

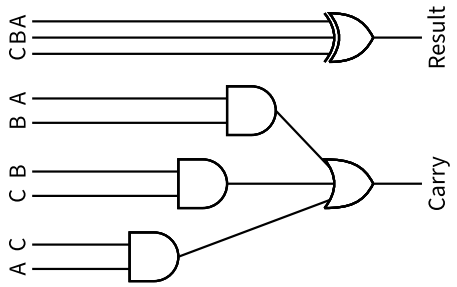


Fig. 6: Ejemplo de sumador completo.

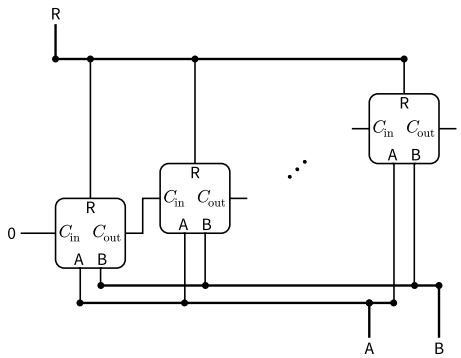


Fig. 7: Sumador de n bits.

```
const fork_b = new Core.ForkComponent<1,
    3>(1, 3, "FORK-B");
const fork_c = new Core.ForkComponent<1,
    3>(1, 3, "FORK-C");
```

Listado 1: Definición de los componentes.

```
const pins = {
  a: Core.makeInputPin(1, "A"),
  b: Core.makeInputPin(1, "B"),
  c: Core.makeInputPin(1, "C"),
  carry: Core.makeOutputPin(1, "CARRY"),
  result: Core.makeOutputPin(1, "RESULT"),
};

const internal = {
  a: fork_a.pins.in,
  b: fork_b.pins.in,
  c: fork_c.pins.in,
  carry: or.pins.out,
  result: xor.pins.out,
};
```

Listado 2: Definición de la entrada y salida.

```
let items: Core.Connection[] = [];
// Result
items.push([[join1.pins.out, join1], [xor.
    pins.in, xor]]);
items.push([[fork_a.pins.out[0], fork_a], [
    join1.pins.in[0], join1]]);
items.push([[fork_b.pins.out[0], fork_b], [
    join1.pins.in[1], join1]]);
items.push([[fork_c.pins.out[0], fork_c], [
    join1.pins.in[2], join1]]);
// Carry
items.push([[join2.pins.out, join2], [or.
    pins.in, or]]);
items.push([[and1.pins.out, and1], [join2.
    pins.in[0], join2]]);
items.push([[and2.pins.out, and2], [join2.
    pins.in[1], join2]]);
items.push([[and3.pins.out, and3], [join2.
    pins.in[2], join2]]);
items.push([[fork_a.pins.out[1], fork_a], [
    and1.pins.in0, and1]]);
items.push([[fork_b.pins.out[1], fork_b], [
    and1.pins.in1, and1]]);
items.push([[fork_b.pins.out[2], fork_b], [
    and2.pins.in0, and2]]);
items.push([[fork_c.pins.out[1], fork_c], [
    and2.pins.in1, and2]]);
items.push([[fork_c.pins.out[2], fork_c], [
    and3.pins.in0, and3]]);
items.push([[fork_a.pins.out[2], fork_a], [
    and3.pins.in1, and3]]);
```

Listado 3: Definición de las conexiones.

```
import * as Execution from "creator-wepsim-
    execution";
const group = Core.GroupComponent.make("FULL
    -ADDER", items, pins, internal, {});
if (Array.isArray(group)) {
  throw new Error("ERROR CONSTRUCTING\n" +
      group.join("\n"));
}
const graph = new Execution.ExecutionGraph(
    group);
const backend = new Execution.
    InterpreterBackend();
graph.compile(backend);
const executor = backend.generate();
```

Listado 4: Construcción del componente final.

```
// Prepare for execution
const a_signal=executor.fromPin(group.pins.a
     as Core.Pin)!;
const b_signal=executor.fromPin(group.pins.b
     as Core.Pin)!;
const c_signal=executor.fromPin(group.pins.c
     as Core.Pin)!;
const r_signal=executor.fromPin(group.pins.
    result as Core.Pin)!;
const o_signal=executor.fromPin(group.pins.
    carry as Core.Pin)!;
const initial_memory = new Map<Execution.
    Memory, Execution.HeavyInt[]>();
const inputs = new Map<Execution.Signal,
    Execution.HeavyInt>();
executor.reset(initial_memory);
// Execute
inputs.set(a_signal, {value: 1n, mask: 1n});
inputs.set(b_signal, {value: 1n, mask: 1n});
inputs.set(c_signal, {value: 0n, mask: 1n});
const cycle = executor.semiCycle(inputs);
console.log(`Status: ${cycle.kind}$`);
console.log(`Carry: ${executor.get(o_signal)
    !.value}`);
console.log(`Result: ${executor.get(r_signal
    )!.value}`);
```

Listado 5: Preparación para la ejecución y ejecución de medio ciclo de reloj.

Este sumador completo está formado por 5 puertas lógicas o componentes, nótese que hay variaciones del mismo circuito con seis puertas lógicas, en este caso la puerta lógica *OR* exclusiva (*XOR*) tiene tres señales de entrada. El número de componentes sería de 5 (una por cada salida de cada puerta lógica), más las 3 señales de entrada (en la Figura 6 aparecen repetidas para aumentar la legibilidad), dando un total de 8 señales de entrada. Teniendo esto en cuenta, un sumador de n bits necesitaría un total de $5n$ componentes y $8n$ señales.

Para estudiar la escalabilidad se ha probado a medir el tiempo de ejecución y el uso de memoria de cada una de las etapas de ejecución: generación del grafo de componentes, generación del grafo de señales, compilación del grafo de señales, preparación

para la ejecución y ejecución. Adicionalmente, se ha medido la sobrecarga de componentes necesaria para generar el grafo.

La generación del componente se refiere a la construcción utilizando la API de TypeScript del propio componente. El sumador completo es un grupo de componentes (puertas lógicas XOR, AND y OR y bifurcaciones, duplicaciones y concatenaciones de señales) y el sumador de n bits es un grupo de sumadores completos. A la hora de la generación de cada grafo, se hace una validación de todos los componentes, en total se hacen $n + 1$ validaciones.

La generación del grafo de señales se refiere a la construcción del grafo acíclico dirigido a partir de la descripción del componente y su posterior ordenamiento de instrucciones utilizando un planificador, en este caso, un ordenamiento topológico.

La compilación del grafo de señales se refiere al proceso de creación de un *backend* al que se le pasan ordenadas las señales y componentes y que genera a partir de ellos un ejecutor. En esta primera aproximación, el *backend* no tiene fase de optimización.

Finalmente, antes de ejecutar medio ciclo, es necesario extraer las señales y memorias a partir del grafo y darles valores iniciales. El paso de extracción solo se realiza una única vez; el de asignación, cada medio ciclo.

Las pruebas se han realizado en un Intel i7-10750H (12) @ 5.000GHz con 16 GiB de RAM y con GNU/Linux 6.12.68-1 (Manjaro). Se utiliza `bun` 1.2.22+6bafe2602 como entorno de ejecución web. Las pruebas se han realizado para sumadores de 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192 y 16384 bits respectivamente en dicho sistema. Para cada ancho de bits, las pruebas se han realizado un total de 32 veces, y para cada una de esas veces se han ejecutado 100 medios ciclos (50 ciclos) de ejecución, en total 32000 medios ciclos por cada ancho de bits.

B. Resultados

B.1 Generación del ejecutor

En la Figura 8 se comparan los tiempos necesarios para la generación del grafo de componentes, la del grafo de señales y la compilación del mismo. El eje ordenadas está en escala logarítmica y muestra el tiempo de ejecución en milisegundos. Del mismo modo, el eje de abscisas está en escala logarítmica y muestra el tamaño en bits del sumador. Se puede ver que a crece linealmente con el número de bits: al duplicar el número de bits, se duplica el tiempo de ejecución.

Se observa, que la fase más costosa es la de creación del grafo de componentes, especialmente cuando la anchura del sumador es menor. Esto se puede deber a todas las validaciones que se tienen que realizar para demostrar que se puede convertir en un grafo acíclico dirigido, y para poder realizar suposiciones que simplifican transformaciones posteriores. Sin embargo, a partir de los 2048 bits, el tiempo de compilación parece ser superior.

Se puede ver en la gráfica 9 el uso de memoria de

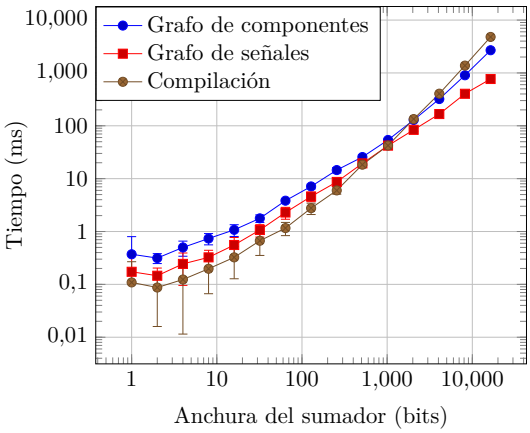


Fig. 8: Tiempo de ejecución en milisegundos para: la generación del grafo de componentes, la del de señales y para la compilación.

estas tres fases. Igual que en la Figura 8, ambos ejes de ordenadas y abscisas están en escala logarítmica, para observar mejor los resultados.

Se puede ver que el uso de memoria para la generación del grafo de componentes es errático y el uso tiene mucha variabilidad. El recolector de basura se llama antes de llamar a la subrutina, lo que permite obtener el uso de memoria actual. Y se vuelve a ejecutar después de haberla llamado. `bun` solo actualiza las estadísticas de uso de memoria cuando se llama al recolector de basura. Así que la gráfica, nos está dando el uso efectivo de memoria de cada una de las estructuras de datos. Aún así, el uso de memoria del grafo de componentes es mucho menor que cualquiera de las otras dos fases.

Finalmente, la Figura 10, muestra el uso total de memoria, desde el inicio del proceso completo hasta después de haber realizado tantos ciclos de ejecución. Se puede ver de nuevo que crece de manera lineal con la anchura del sumador. La Figura 11, muestra el factor de uso de memoria total entre el número de bits del computador. Esto, además de confirmar la linealidad de la escalabilidad, muestra que el uso de memoria es más variable cuanto más pequeño es el circuito y que el uso de memoria por cada bit del sumador tiende a casi 46 KiB por bit (unos 732.17 MiB de memoria para el caso de 16384 bits).

B.2 Ejecución

Para la parte de ejecución se compara el tiempo de preparación del ejecutor, con el tiempo necesario para cada medio ciclo de ejecución. La Figura 12 muestra el tiempo de ejecución en escala logarítmica respecto a la anchura del sumador en escala logarítmica. Se observa un comportamiento lineal con respecto a la escalabilidad: al duplicar el número de componentes, se duplica el tiempo de ejecución. El medio ciclo tiende a ser alrededor de 5 veces mayor que el tiempo de preparación.

Sin embargo, si se observa la gráfica de la Figura 13, donde se muestra la cantidad de combinadores ejecutados por segundo de ejecución, se puede ver que en realidad no escala al aumentar la anchura del

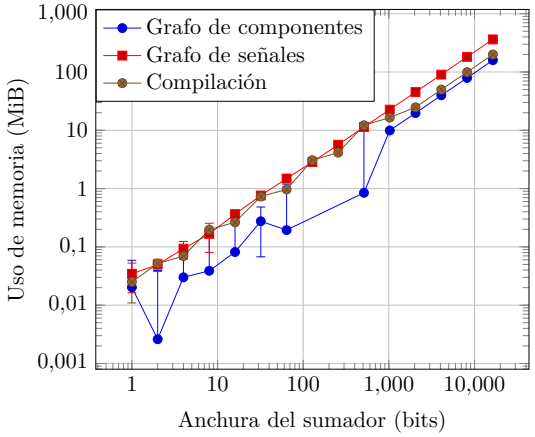


Fig. 9: Uso de memoria en MiB que se ha utilizado para: generar el grafo de componentes, el de señales y para compilarlo. Se ha llamado al recolector de basura de `bun` antes y después de ejecutar los bloques de código pertinentes.

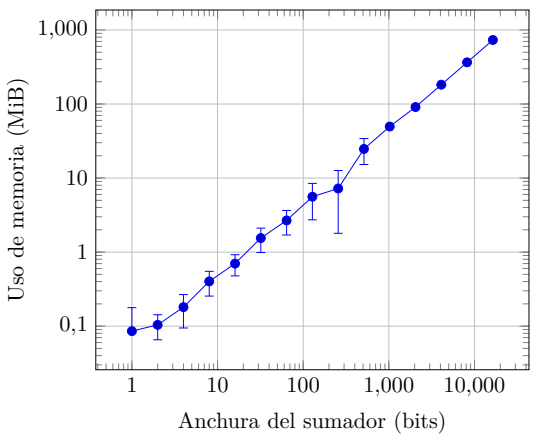


Fig. 10: Uso total de memoria desde la creación y compilación del componente, hasta después de ejecutar varios medios ciclos.

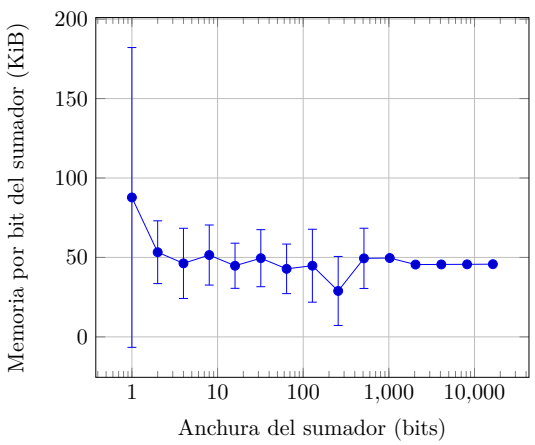


Fig. 11: Uso de memoria necesario en el proceso completo por cada bit adicional en la anchura del sumador.

sumador. Se procesan menos combinadores por segundo a medida que aumenta su número. Esto abre las puertas a posibles mejoras en el *backend* de compilación.

En cuanto al uso de memoria, como no se crean objetos nuevos a la hora de la ejecución, o bien, si se crea un objeto nuevo es para sustituir alguno antiguo, el uso de memoria adicional es prácticamente nulo. Por lo que se ha decidido omitir la gráfica.

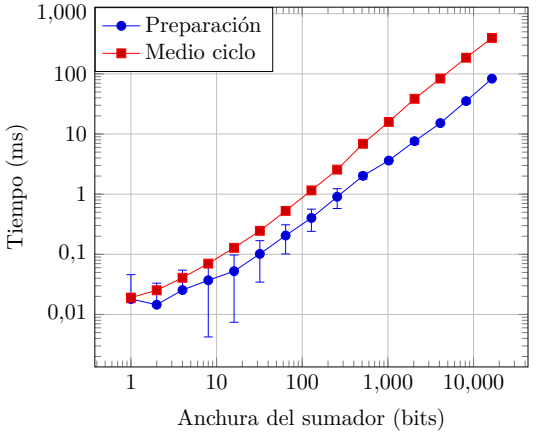


Fig. 12: Tiempo de ejecución en milisegundos comparando el tiempo necesario para preparar la propia ejecución del simulador una vez compilada, y el tiempo que tarda en ejecutar medio ciclo de ejecución.

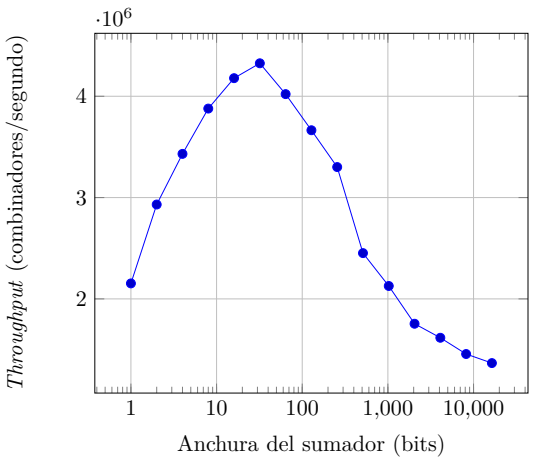


Fig. 13: Cantidad de combinadores (componentes) ejecutados por segundo para cada una de las anchuras del sumador.

B.3 Uso de señales y componentes

Finalmente se ha decidido comprobar cuál es la sobrecarga de señales y componentes generados respecto a la cantidad real necesaria. Nótese que debido al diseño del sistema, hay componentes "fantasma" (bifurcaciones, duplicación de señales, concatenación), que son implícitos en el sistema real, pero es necesario especificarlos en el grafo.

Otro problema fundamental a observar es la cantidad de señales repetidas y de componentes que ejecutan la función identidad (no hacen nada). Estos componentes superfluos se añaden a la hora de generar el grafo de señales para conectar los distintos subgrafos de manera sencilla. Para cada función identidad, existe una señal de salida asociada que tiene el mismo valor que la entrada, luego la señal está duplicada y no aporta información nueva.

En primer lugar en la Figura 14 se observa linealidad en la cantidad de señales y combinadores (componentes) al aumentar el tamaño del sumador.

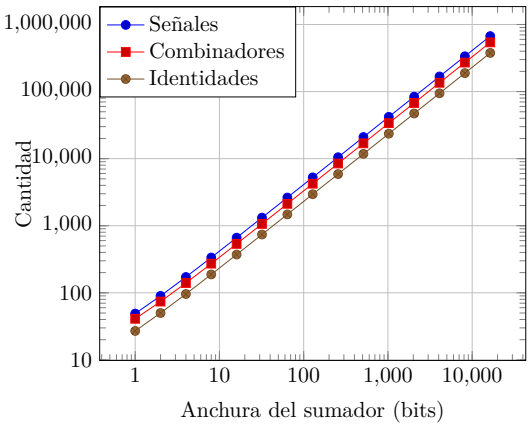


Fig. 14: Cantidad generada de señales, combinadores (puertas lógicas) y cuántos combinadores son identidades.

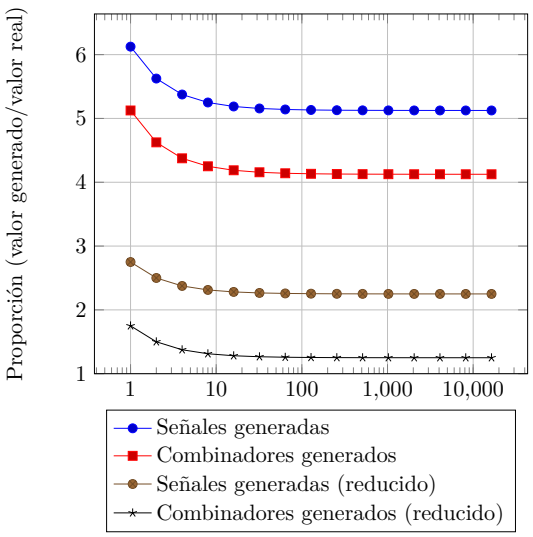


Fig. 15: En esta figura se muestran la proporción de señales y combinadores generados, respecto a la cantidad real necesaria. Adicionalmente, muestra cómo sería la proporción si se eliminaran los combinadores identidad y las señales que se refieren al mismo valor (reducido).

El número de identidades aumenta linealmente también, pues al añadir cada subgrupo de componentes, se añade tantas identidades como entradas y salidas tenga.

Ahora bien, si se hace el factor cantidad de señales generadas entre el valor real, nos da los resultados de la gráfica de la Figura 15. Además del factor de señales y combinadores generados con respecto a los reales, se ha añadido el factor de dichas señales y combinadores, pero substrayendo el número de identidades (componentes y señales superfluos). En el caso de combinadores, el resultado tiende a 1,25 (suponiendo que hay un paso de compilación que elimina los componentes identidad superfluos) comparado con el 4,125 que tiene ahora sin paso de compilación. De manera similar, las señales tenderían a 2,25 si ignoramos las duplicadas, comparadas con las 5,125 que hay ahora.

V. Conclusiones

El sistema es capaz de generar, validar, compilar y evaluar distintos tipos de circuitos. Además de ofrecer distintos componentes primitivos para la creación de prácticamente cualquier tipo de circuito digital síncrono.

Sin embargo, se ha visto que esta primera aproximación, que no optimiza el grafo, tiene el problema de que más de la mitad de las señales están duplicadas y casi tres cuartos de los componentes en el grafo son superfluos. Todavía se pueden hacer muchas optimizaciones, probar distintos planificadores e incluso introducir paralelismo.

También se quieren añadir compiladores distintos. Por ejemplo uno que genere código JavaScript válido en vez de funciones lambda como se hace ahora, para eliminar el coste de las llamadas a función y reducir el tamaño del código compilado. Otra idea es compilar a WebAssembly para mejorar al máximo la velocidad del programa. También se ha propuesto generar código C, para poder cargar imágenes y hacer pruebas rápidamente en local o incluso utilizar el *backend* de CLang de WebAssembly para generar ensamblador web.

Aprovechando el patrón visitante y las estructuras de datos internas, se pueden añadir extensiones al sistema: generación de código VHDL a partir de la descripción, generación de un diagrama interactivo, obtención de texto alternativo para dar accesibilidad al sistema, etcétera.

Los resultados obtenidos son muy positivos, y permite dar un paso importante en poder añadir al simulador CREATOR un procesador elemental como WepSIM descrito usando la propuesta presentada en este trabajo, de manera que pueda tener un motor de ejecución que también permita microprogramar las instrucciones.

Referencias

[1] Diego Camarmas-Alonso, Felix Garcia-Carballeira, Alejandro Calderon-Mateos, and Elias del Pozo-Puñal, "Creator: An educational integrated development environment for risc-v programming," *IEEE Access*, vol. 12, pp. 127702–127717, 2026.

[2] Felix Garcia-Carballeira, Alejandro Calderon-Mateos, Saul Alonso-Monsalve, and Javier Prieto-Cepeda, "Wepsim: an online interactive educational simulator integrating microdesign, microprogramming, and assembly language programming," *IEEE Transactions on Learning Technologies*, vol. 1, no. 1, pp. 0–8, 2 2019.

[3] Félix García-Carballeira, Alejandro Calderón-Mateos, Saúl Alonso-Monsalve, and Javier Prieto-Cepeda, "WepSIM: An online interactive educational simulator integrating microdesign, microprogramming, and assembly language programming," vol. 13, no. 1, pp. 211–218.

[4] Álvaro Guerrero-Espinosa, José-Antonio Verde-Jiménez, Félix García-Carballeira, and Alejandro Calderón-Mateos, "Implementación en fpga del procesador elemental de wepsim," in *Jornadas SARTECO 2026*, Madrid, Spain, June 2026, pp. 1–7.

[5] Justus Bogner and Manuel Merkel, "To type or not to type? a systematic comparison of the software quality of JavaScript and typescript applications on GitHub," in *Proceedings of the 19th International Conference on Mining Software Repositories*. MSR '22, pp. 658–669, Association for Computing Machinery.

[6] "IEEE standard for VHDL language reference manual," pp. 1–673.

[7] "IEEE standard for SystemVerilog–unified hardware design, specification, and verification language," pp. 1–1354.

[8] Douglas J. Smith, "VHDL & verilog compared & contrasted—plus modeled example written in VHDL, verilog and c," in *Proceedings of the 33rd annual Design Automation Conference*. DAC '96, pp. 771–776, Association for Computing Machinery.

[9] Karsten Becker, "A web based tool for teaching hardware design based on the plain simple hardware description language," in *2014 IEEE Global Engineering Education Conference (EDUCON)*, pp. 88–93, ISSN: 2165-9567.

[10] Oron Port and Yoav Etsion, "DFiant: A dataflow hardware description language," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, ISSN: 1946-1488.

[11] Dmitri Boulytchev and Oleg Medvedev, "Hardware description language based on message passing and implicit pipelining," in *2010 East-West Design & Test Symposium (EWDTS)*, pp. 438–441.

[12] Mahshid Shahmohammadian and Geoffrey Mainland, "Synthesizing efficient hardware from high-level functional hardware description languages," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 634–637.

[13] Richard Lin, Rohit Ramesh, Nikhil Jain, Josephine Koe, Ryan Nuqui, Prabal Dutta, and Bjoern Hartmann, "Weaving schematics and code: Interactive visual editing for hardware description languages," in *The 34th Annual ACM Symposium on User Interface Software and Technology*. UIST '21, pp. 1039–1049, Association for Computing Machinery.

[14] Richard Lin, Rohit Ramesh, Connie Chi, Nikhil Jain, Ryan Nuqui, Prabal Dutta, and Björn Hartmann, "Polymorphic blocks: Unifying high-level specification and low-level control for circuit board design," in *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. UIST '20, pp. 529–540, Association for Computing Machinery.

[15] "IEEE standard multivalue logic system for VHDL model interoperability (std_logic_1164)," pp. 1–24.

Implementación en FPGA del Procesador Elemental de WepSIM

Álvaro Guerrero-Espinosa[1], José-Antonio Verde-Jiménez[1], Félix García-Carballeira[1], Alejandro Calderón-Mateos [1]

Resumen— **Facilitar adquirir un mejor conocimiento en el diseño de procesadores con herramientas de trabajo más cercanas está tomando una mayor relevancia actualmente, tanto por el impulso de la IA como por aspectos socio-económicos que impulsan una mejor soberanía digital en Europa.**

Este artículo describe el trabajo realizado en el diseño e implementación en *hardware* real basado en FPGA del Procesador Elemental utilizado en el simulador docente WepSIM. Disponer del Procesador Elemental en FPGA ayuda a tener un marco de referencia que permite mantener una mejor visión global de los distintos elementos de un computador, su interacción, los efectos en la unión entre el *hardware* y el *software*, poder experimentar cambios y comprobar el impacto de dichos cambios, etc. Y todo ello ayuda tanto a mejorar el conocimiento en diversos aspectos del diseño de un procesador como facilitar una herramienta de trabajo más cercana.

Se ha partido del simulador WepSIM por ser de código abierto, y del Procesador Elemental de WepSIM por estar basado en una unidad de control microprogramada que permite cargar en su memoria de control distintos juegos de instrucciones. Esto facilita experimentar distintas variaciones en el diseño de instrucciones, así como entender el impacto real de dichas variaciones.

A la hora de llevar el Procesador Elemental a una FPGA se ha precisado revisar primero el diseño existente del procesador. Tras dicha revisión, se ha añadido a la memoria de control la implementación del conjunto de instrucciones RV32I (con extensión Zmmul) de RISC-V de forma que pueda probarse la FPGA por completo. El resultado ha sido muy satisfactorio, disponiendo de un Procesador Elemental que ejecuta a 25MHz en una FPGA Nexys A7-100T.

Palabras clave— **FPGA, RISC-V, microprocesador, microprogramación, ensamblador**

I. Introducción

El Megaprocessor [1] es un proyecto que se hizo famoso [2] por presentar el proceso de construcción de un ordenador con una CPU sencilla de 16 bits, construido a gran tamaño y que permitía ejecutar una implementación del famoso juego Tetris escrito en ensamblador. Del tamaño de una habitación entera y construido por un equipo de personas, permite introducir los elementos *hardware* y *software* básicos con la cercanía de poder visitar este sistema en un museo, lo que ayuda a acercar a todo tipo de público el diseño de *hardware* y *software*.

Ha sido la inspiración para un curso [3] basado en proyectos que buscan ayudar a que una persona pueda aprender a construir un ordenador moderno simplificado, desde los primeros principios a nivel cercano al *hardware* (desde puertas lógicas NAND) hasta el nivel cercano al *software* (al Tetris). Este curso está diseñado para ser lo más independiente y autocontenido posible, trabajando con los elementos más básicos de un computador, así como los principales pasos en la construcción de un sistema de forma simplificada.

Ambas ideas (el Megaprocesador y el curso de NAND a tetris) han supuesto una gran ayuda a redescubrir que aprender cómo se integra *hardware* y *software* con un computador simple de referencia ayuda a entender mejor el *hardware* y *software*. La construcción de un proyecto de computador simplificado permite tener un esqueleto donde comprender mejor dónde va cada elemento, ayudando a dar una perspectiva global y facilitando retener el aprendizaje al tener posibilidad de probar el por qué de las decisiones de diseño.

Sin embargo, ambos proyectos son muy difíciles de integrar en los cursos actuales de estructura y arquitectura de computadores de un cuatrimestre en un aula: el megaprocesador tiene una complicación en logística que hace muy difícil reproducir la experiencia, y el curso nand2tetris no facilita su integración en los actuales temarios por ser muy cerrado y autocontenido.

Este trabajo describe el diseño e implementación del Procesador Elemental utilizado en el simulador WepSIM [4][1] sobre una FPGA. Al implementar el Procesador Elemental en *hardware* real basado en FPGA, se facilita poder integrar su uso en los cursos actuales de estructura de computadores: es posible aplicar lo contado en los cursos existentes al Procesador Elemental por su simpleza, y es posible trabajar con el Procesador Elemental en *hardware* real basado en FPGA (lo que facilita una herramienta de trabajo más cercana).

Se ha partido del simulador WepSIM por ser de código abierto, estando su código fuente disponible en [2]. El Procesador Elemental en WepSIM está basado en una unidad de control microprogramada, lo que permite diseñar distintos juegos de instrucciones para entender mejor la unión entre el *hardware* y el *software* [4]. Gracias a WepSIM es posible especificar las señales de control para los distintos ciclos de reloj, así como obtener el contenido de la memoria de control a partir de dicha especificación.

A la hora de llevar el Procesador Elemental a una FPGA se ha precisado revisar primero el diseño exis-

[1]Universidad Carlos III de Madrid, ROR: https://ror.org/03ths8210, Departamento de Informática, Avenida de la Universidad, 30 (edificio Sabatini), 28911 Leganés (Madrid), España. e-mail: {alvarogu,fgcarbal,acaldero}@inf.uc3m.es, jverde@pa.uc3m.es

[1]https://wepsim.github.io/

[2]https://github.com/wepsim/wepsim

tente de manera que se acercara este procesador más todavía a la realidad y a la vez el diseño fuera más sencillo de usar en entornos educativos. Tras dicha mejora del Procesador Elemental, se ha añadido a la memoria de control la implementación del conjunto de instrucciones RV32I (con extensión Zmmul) de RISC-V [5] de forma que pueda probarse en la FPGA con todo integrado.

El resto de este trabajo se estructurará así: la Sección II describe el Estado del Arte, la Sección III describe cómo se ha implementado el procesador, unidad de control, y dispositivos de E/S. En la Sección IV se presentan los cambios que se han tenido que realizar en el diseño del circuito para permitir su implementación, y en la Sección V se presentan los resultados obtenidos tras implementar el procesador. Por último, la Sección VI presenta las principales Conclusiones y Trabajos Futuros.

II. Estado del Arte

Debido a diversos factores socio-políticos actuales, impulsar la soberanía digital por parte de la Unión Europea ha pasado a ser totalmente prioritario. Una de las áreas fundamentales es el diseño de *chips*, y disponer de herramientas y conocimiento en el diseño de *chips* de código abierto es clave, por lo que la Unión Europea está empezando a dar su apoyo [6].

Uno de los mayores problemas del diseño de *chips* es su elevado coste de fabricación: fabricar un milímetro cuadrado en una tecnología de cinco nanómetros cuesta unos 100.000 euros, y las licencias de las herramientas de diseño electrónico asistido por computador pueden alcanzar el millón de euros anuales [7]. Ello imposibilita a las pequeñas empresas y estudiantes experimentar, realizar prototipos, y aprender. La principal alternativa es el uso de FPGAs, que permiten implementar y modificar circuitos sin necesidad de fabricar nuevos *chips*.

En el ámbito educativo, actualmente existen muchos trabajos para la enseñanza de la estructura de un procesador y el diseño de *chips*. Algunos ejemplos son [8] y [9]. Estos utilizan diseños de procesadores y conjuntos de instrucciones propios, creados para ser lo más simples posibles. Para lograr esto, utilizan arquitecturas basadas en un acumulador, sin un banco de registros. Este tipo de diseños permite utilizar circuitos muy simples. Sin embargo, esta simplicidad también los vuelve menos generales, impidiendo el uso de conjuntos de instrucciones más convencionales como RISC-V.

En el ámbito más industrial actualmente existen múltiples implementaciones de procesadores RISC-V en FPGA, como [10] y [11]. Sin embargo, estas implementaciones son complejas para una primera aproximación al diseño de *chips*: utilizan *pipelines* complejas, memorias caché avanzadas, instrucciones vectoriales, etc.

Hasta donde los autores conocen, el único proyecto más cercano a un equilibrio entre simplicidad y completitud es [12]. Este diseño implementa el conjunto de instrucciones base de RISC-V (RV32I) y está basado en un bus interno sin lógica para *pipeline*. Sin embargo, el diseño de este procesador utiliza una unidad de control distribuída y cableada, lo que dificulta totalmente su modificación para experimentar y mejorar el aprendizaje de su funcionamiento. Disponer de una unidad de control microprogramada facilita poder probar distintos juegos de instrucciones y/o extensiones de una forma más intuitiva, ofreciendo más flexibilidad para experimentar y aprender.

Tras estudiar las alternativas disponibles en la actualidad, creemos que WepSIM, junto con su implementación en FPGA, es muy útil para la enseñanza de la estructura de un procesador, la programación en ensamblador, y la microprogramación debido a los siguientes motivos:

- Utiliza un diseño simple y fácil de entender, basado en un Procesador Elemental con bus interno y unidad de control microprogramable.
- El diseño del Procesador Elemental es lo suficientemente genérico y flexible como para implementar diferentes conjuntos de instrucciones, y no volverse una limitación en cuanto las funcionalidades que se pueden llegar a implementar en la unidad de control.
- La unidad de control microprogramable permite modificar el conjunto de instrucciones implementado fácilmente, y facilita un diseño de unidad de control más fácil de usar y cambiar.
- Cuenta con un simulador web con interfaz gráfica que permite visualizar su funcionamiento paso a paso, lo que facilita el aprendizaje.

III. Implementación

En esta sección se describirá cómo se ha implementado el procesador en una FPGA.

A. Placa escogida

Para llevar a cabo este trabajo, se ha utilizado la placa Nexys A7-100T, que se puede ver en la Figura 1.

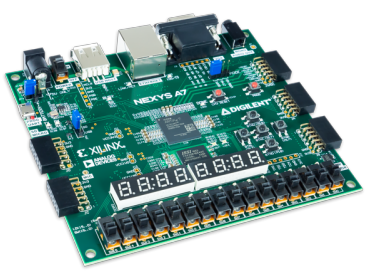


Fig. 1: FPGA Nexys A7-100T [13].

Esta placa cuenta con una FPGA con 63.400 *Look-Up Tables* (LUTs) de seis entradas y 126.800 *Flip-Flops* (FFs) para implementar la lógica. Estos se agrupan en 15.850 bloques lógicos independientes, cada uno con 4 LUTs y 8 FFs. También contiene una RAM de bloque configurable con hasta 4.860 Kbits, y un reloj de 100 MHz. [14]

La placa tiene 16 interruptores, 16 LED, 2 LED RGB, 5 botones, un *display* de 7 segmentos con 8

dígitos, y 32 pines de GPIO (*General Purpose Input/Output*). Además, cuenta con un conversor USB-UART (*Universal Asynchronous Receiver/Transmitter*) que la permite comunicarse con un ordenador utilizando un protocolo serie estándar a través de un cable USB.

B. Implementación del circuito

El diseño del Procesador Elemental de WepSIM se puede ver en la Figura 2. Por otro lado, su unidad de control se puede ver en la Figura 3. Estos diseños se han utilizado como referencia durante la implementación en FPGA.

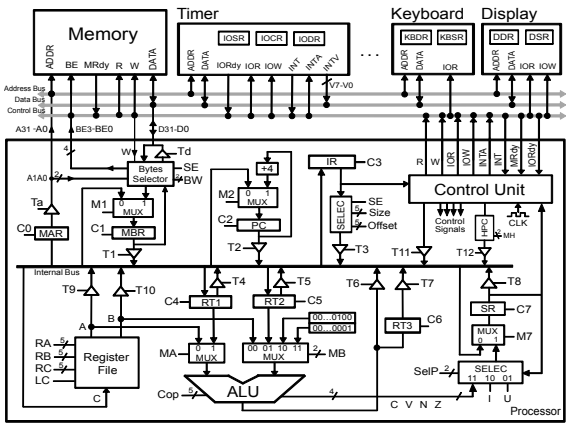


Fig. 2: Diseño del Procesador Elementar de WepSIM.

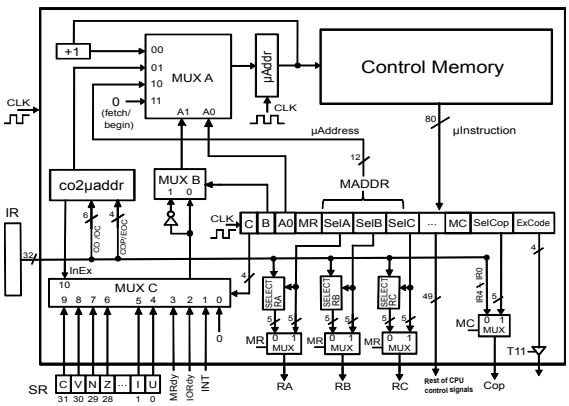


Fig. 3: Diseño de la Unidad de Control de WepSIM.

El circuito se ha implementado mayoritariamente en VHDL 2008 (IEEE 1076-2008), aunque se ha utilizado también un subconjunto reducido de VHDL 2019 (IEEE 1076-2019). Esto fue debido a que muchas de las herramientas existentes en la actualidad, y utilizadas durante el desarrollo de este trabajo, no soportan el estándar de 2019 completamente [15] [16].

Se han implementado primitivas genéricas para desarrollar circuitos, como multiplexores o registros, para facilitar el desarrollo y modificación del circuito implementado. Estas primitivas están parametrizadas, permitiendo ajustarlas según las necesidades de uso. Con esto se espera facilitar la modificación del circuito, ya que solo es necesario instanciar estas primitivas con los parámetros deseados y definir las conexiones entre los diferentes elementos.

Además del procesador y su unidad de control, también se han añadido integraciones con múltiples de los dispositivos presentes en la placa utilizada: interruptores, LED monocolor, *display* de 7 segmentos, y comunicación UART. Estos dispositivos pueden ser controlados por el procesador y el código ensamblador que este ejecute, permitiendo realizar pruebas del funcionamiento del circuito.

C. Depuración

Para facilitar la depuración del funcionamiento del circuito en la FPGA, se ha añadido una lógica adicional al circuito. Esta lógica permite pausar la ejecución del procesador y ejecutar el código ensamblador instrucción a instrucción. Para lograr esto, como no se puede pausar el reloj, se ha decidido bloquear el avance de la unidad de control. Esto se realiza durante una microinstrucción idempotente, ya que esta microinstrucción se ejecutará en bucle hasta que se reanude el avance del procesador. Esta funcionalidad también se ha utilizado para implementar la instrucción `ebreak`, que permite pausar la ejecución del procesador al llegar a ella.

Otra funcionalidad de depuración añadida es la visualización de diversos registros del procesador. Normalmente, los dispositivos de E/S de la FPGA están bajo el control directo del procesador. Sin embargo, cuando este se pausa, se permite el uso del *display* de 7 segmentos y los interruptores para mostrar el valor de los registros: dos interruptores se utilizan para seleccionar entre el contador de programa, el registro de instrucción, y el banco de registros. Cuando está seleccionado el banco de registros, cinco interruptores adicionales permiten seleccionar uno de los 32 registros contenidos en este.

D. Comuniación UART

UART es un protocolo de transmisión serie asíncrono, en el que los bits se envían uno a uno sin mantener relojes sincronizados entre los dispositivos. Los paquetes en UART están formados por un bit de inicio, 5-9 bits de datos, un bit de paridad opcional, y 1-2 bits de parada. La Figura 4 muestra un ejemplo de un paquete de UART. [17]

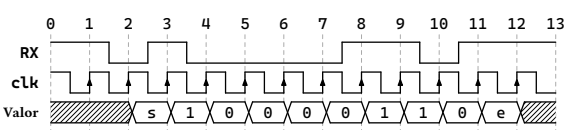


Fig. 4: Ejemplo de envío de un byte por UART, utilizando un bit de parada.

Para regular el flujo de datos entre los dispositivos y evitar la pérdida de datos, se puede utilizar control de flujo *software* o *hardware*. El control de flujo *software* utiliza dos bytes específicos para indicar cuándo se debe pausar y reanudar la transmisión, enviados por el mismo canal que los datos, mientras

que el *hardware* utiliza dos cables adicionales para esta tarea.

Se ha elegido utilizar control de flujo *hardware* por ser más fiable que el control de flujo *software*. Al utilizar un canal dedicado, el control de flujo *hardware* evita los posibles retrasos de los mensajes de control. De esta forma es más fácil evitar la pérdida o corrupción de los datos. Además, esto permite transmitir bytes arbitrarios, sin necesidad de utilizar secuencias de escape para diferenciarlos de los mensajes de control. [18]

El circuito implementado para las comunicaciones UART se ha dividido en tres componentes principales: un controlador para la transmisión de datos, otro para la recepción, y un circuito encargado de gestionar el uso de estos controladores por el procesador.

Los controladores de transmisión/recepción se ejecutan con una frecuencia de reloj 8 veces superior a la frecuencia de transmisión. Gracias a esto, estos se pueden sincronizar con el inicio de la transmisión de un paquete, evitando que los errores de sincronización se acumulen. Además, esto es especialmente importante en el caso de la recepción, al permitir medir el valor de la señal en la mitad del periodo de transmisión (reiniciando el contador 4 ciclos después de detectar el inicio de un paquete), lo que minimiza los errores [17] [19]. Como esta frecuencia es significativamente menor que la frecuencia del procesador, se ha utilizado un divisor de frecuencia.

Los controladores de transmisión y recepción utilizan una máquina de estados con una implementación directa del protocolo. Cabe destacar que para la recepción se utiliza una votación de las últimas 4 mediciones de la señal para evitar errores temporales. Este valor se determinó experimentalmente, buscando minimizar los errores sin añadir demasiada complejidad al circuito.

Para permitir al procesador utilizar estos controladores, se ha añadido una cola FIFO a cada uno de ellos, que hace de intermediario entre el procesador y los controladores. Estas colas se encargan de almacenar los datos generados por el procesador hasta que puedan ser enviados, y los datos recibidos hasta que el procesador los consuma.

Las colas se han implementado con un buffer circular, y se ejecutan a la frecuencia de reloj del procesador. Como las comunicaciones utilizan un reloj con una menor frecuencia, es necesario filtrar las señales de control de los controladores UART con un detector de flancos, para asegurar que los datos se añaden o eliminan de las colas una sola vez.

Estas colas se han creado con una capacidad máxima de 256 bytes cada una. Además, tienen señales de control que indican cuándo llegan a la mitad de su capacidad, que se utilizan para el control de flujo de datos. Esto da un margen de tiempo al dispositivo para pausar su transmisión antes de que se pierdan datos.

IV. Modificaciones del circuito

En esta sección se describirán los cambios que se han tenido que realizar al circuito del procesador y la unidad de control. Aunque se ha intentado mantener el circuito original (visible en las Figuras 2 y 3), ha sido necesario realizar algunas modificaciones debido a limitaciones del *hardware* y para poder implementar el conjunto de instrucciones de RISC-V.

A. Decodificación de instrucciones

Ha sido necesario modificar el circuito encargado de obtener la dirección de la memoria de control en la que se encuentra el microprograma asociado a la instrucción actual (componente "`co2`μ`addr`" en la Figura 3).

En el circuito original, para lograr esto, se extraía un rango específico de bits, correspondiente al código de operación. Este rango se utilizaba como íncice de una tabla cargada en una memoria ROM para obtener la dirección del microprograma asociado.

En RISC-V, sin embargo, los bits necesarios para identificar una instrucción pueden cambiar según la instrucción [5], como se puede ver en la Figura 5. Debido a esto, el método utilizado por el circuito original es insuficiente para implementar RISC-V.

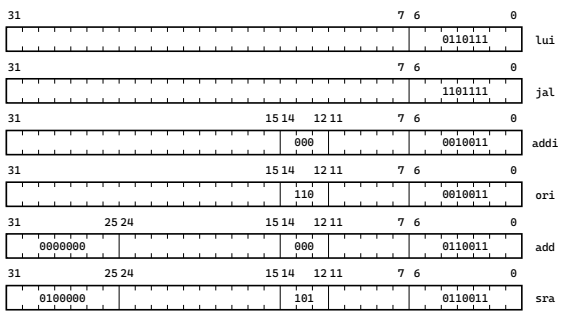


Fig. 5: Ejemplos de la codificación de las instrucciones en RISC-V.

Esto se ha solucionado remplazando este componente por un circuito que compara la instrucción con una serie de patrones de bits. Cada patrón se corresponde con los bits identificativos de una instrucción, indicando para cada bit si debe ignorarse (representado con `"-"`), o si debe tener un cierto valor (representado con `0/1`). Cada uno de estos patrones tiene además una dirección asociada, correspondiente a la dirección del microprograma que implementa la instrucción.

El nuevo circuito toma esta lista de patrones y los compara con la instrucción actual, hasta encontrar una coincidencia. Esto permite añadir un patrón que ignora todos los bits como última opción, que se utilizará cuando no se reconoce la instrucción por ninguno de los patrones anteriores. El funcionamiento de este circuito se puede ver en el Algoritmo 1.

B. Decodificación de inmediatos

Ha sido necesario modificar el circuito encargado de extraer los valores inmediatos de las instrucciones

Algorithm 1 Decodificación de una instrucción.

```
// patrones: lista ordenada de tuplas (patrón, dirección)
instrucción_desconocida ← true
dirección ← Indefinido
for all p in patrones do
  if registro_instrucción matches p.patrón then
    instrucción_desconocida ← false
    dirección ← p.dirección
    break
  end if
end for
return (instrucción_desconocida, dirección)
```

(componente "`selec`" debajo de "`IR`" en la Figura 2).

En el circuito original existe un componente que permite extraer un rango de bits de la instrucción actual. Este circuito toma como señales de control la posición del primer bit del rango y su tamaño, y devuelve el rango correspondiente extendido a 32 bits. Además, este permite seleccionar si esta extensión se realiza rellenando con 0s o con una extensión de signo (copiando el bit más significativo del rango).

La principal limitación de este circuito es que solo permite extraer un único rango de bits, que debe estar ordenado y empezar por el bit menos significativo.

En RISC-V, como se puede ver en la Figura 6, esto no se cumple: los valores inmediatos pueden estar partidos en varios segmentos, desordenados, e incluso no codificar alguno de los bits menos significativos [5].

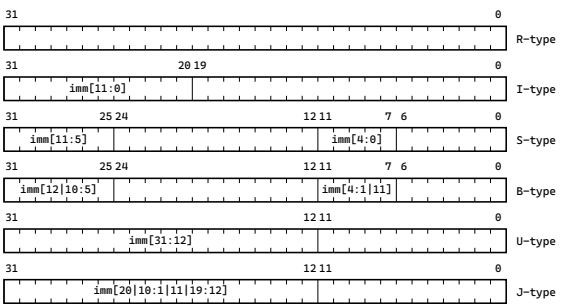


Fig. 6: Codificaciones de los valores inmediatos en RISC-V.

Para solucionar esto, se han considerado múltiples posibilidades. La primera opción fue permitir el uso de múltiples rangos de bits simultáneos, hasta un cierto límite (4 para RISC-V, ya que cada rango debe tener los bits ordenados). El problema de esta opción es que requiere un circuito muy complejo con muchas señales de control, ya que se necesitan dos señales de 5 bits para definir cada rango.

La segunda opción fue añadir un acumulador al circuito. Esto permitiría utilizar múltiples rangos en varios ciclos de reloj, añadiendo un nuevo rango al acumulador en cada ciclo. De esta forma, es posible utilizar las señales de control correspondientes a un único rango de bits, y no se limita la cantidad de rangos de bits que se pueden utilizar. Además, se simplifica significativamente el circuito necesario, y este se puede comportar como el circuito antiguo cuando se utiliza un único rango. El problema de esta opción es que dificulta el uso y comprensión del circuito, ya que requiere múltiples ciclos de reloj y mantener un estado adicional entre ciclos.

La última opción considerada, y la que se utilizó en el circuito final, fue extender el circuito original con los casos necesarios para RISC-V. Esta opción se aprovecha de que cuando se especifica un rango de tamaño 0, el resultado del circuito original siempre es 0. Esto deja espacio para utilizar la señal del desplazamiento para codificar otros datos, sin afectar al comportamiento del circuito cuando el tamaño del rango es positivo.

El nuevo circuito utiliza esto para codificar en el desplazamiento un identificador del tipo del inmediato a extraer, de entre un conjunto de opciones pre-programadas. Se han añadido todas las opciones necesarias para RISC-V, aunque esto se podría extender para permitir otros conjuntos de instrucciones. Se eligió esta opción por requerir el circuito más simple y fácil de utilizar, y por requerir las menores modificaciones.

C. Memoria RAM

Debido a limitaciones del *hardware*, se han tenido que realizar cambios en el funcionamiento de las lecturas.

El circuito original permite leer de memoria sin ninguna latencia, obteniendo los datos correspondientes en el mismo ciclo de reloj en el que se indica la dirección de memoria a leer. En la realidad, en cambio, se necesita una latencia de al menos un ciclo, utilizando la dirección de memoria indicada en el ciclo anterior. Al ser esto una limitación del *hardware*, es necesario modificar el microcódigo para tener en cuenta esta latencia, sin poder modificar el circuito para eliminarla u ocultarla.

Las operaciones de memoria utilizan una palabra (32 bits) como unidad mínima de operación. Si se quieren realizar lecturas de menos de una palabra, es necesario extraer los bytes correspondientes de la palabra, moverlos al inicio de la misma, y opcionalmente realizar una extensión de signo. En el circuito original, esto se realizaba utilizando un componente específico (componente "`Bytes Selector`" en la Figura 2).

Este componente funciona de forma puramente combinacional, sin latencia. Al añadir la latencia a las lecturas de memoria, este comportamiento significa que sus señales de control se tienen que especificar en el ciclo posterior a la lectura de memoria, que es cuando el valor leído de memoria está disponible para su uso. Sin embargo, durante las escrituras, se debe utilizar el valor actual de las señales de control, ya que en este caso los datos a escribir no tienen latencia.

D. Unidad Aritmetico-lógica (ALU)

Se han implementado todas las operaciones lógicas y bit a bit definidas en el circuito original. Con respecto a las operaciones aritmétcias, se han implementado todas a excepción de la división y su resto/módulo, ya que estas no están soportada por el sintetizador del fabricante de la FPGA [20]. Debido a esto, su implementación requeriría diseñar un circuito explícito, sin poder depender de que la herramienta genere este automáticamente. Además, estas operaciones comúnmente se implementan con un circuito secuencial, que lleva a cabo la operación a lo largo de múltiples ciclos de reloj. La ALU del circuito original, sin embargo, opera de forma puramente combinacional, por lo que implementar estas operaciones también requeriría rediseñar cómo se integra la ALU con el resto del circuito.

Cabe destacar que la implementación de las operaciones en coma flotante se ha quedado fuera del alcance de este trabajo.

E. Inmediatos en las microinstrucciones

El circuito original tiene un campo de 4 bits en las microinstrucciones que permite especificar un valor arbitrario a enviar al bus interno. Este valor se tiene que extender a los 32 bits del bus interno. Sin embargo, no está claro si esto se debe realizar añadiendo 0s (tratando el número como *unsigned*) o con extensión de signo (tratando el número como *signed*).

El circuito original ya cuenta con una señal de control para indicar si se realiza extensión de signo en varios componentes, como la RAM o el decodificador de inmediatos. Debido a esto, se ha decidido utilizar esta señal de control para indicar si se debe realizar extensión de signo al inmediato de la microinstrucción. De esta forma, queda claro cómo se interpreta este valor.

V. Resultados

Con los cambios mencionados en la sección anterior, se ha implementado el conjunto de instrucciones RISC-V en la unidad de control, en su variante de 32 bits. Como ya se ha mencionado, la ALU no tiene operaciones de división o coma flotante, por lo que se ha implementado el conjunto de instrucciónes base (RV32I) con la extensión Zmmul.

Además de este conjunto de instrucciones, también se ha implementado en la unidad de control soporte para interrupciones, tanto síncronas como asíncronas, y llamadas al sistema utilizando interrupciones síncronas. Esto se ha implementado utilizando una tabla de rutinas de tratamiento de interrupciones, localizada en la dirección de memoria 0, que contiene la dirección de memoria en la que se encuentra la función encargada de responder a cada interrupción.

Se ha definido la RAM con una capacidad máxima de 512 KB, ya que esta es la máxima potencia de 2 para la que tiene suficientes recursos la FPGA. Aunque la FPGA tiene un reloj de 100MHz, debido al diseño basado en un bus interno del circuito, se ha tenido que reducir su frecuencia a 25MHz para que las señales se puedan propagar a todos los componentes en un ciclo. Con esta frecuencia de reloj, se ha conseguido una frecuencia de transmisión por UART de 1.5MHz, lo que permite una tasa de transferencia máxima de aproximadamente 150KB/s.

Para facilitar las pruebas, se ha implementado un cargador en la unidad de control del procesador, que se ejecuta al encender el mismo antes de empezar la ejecución de instrucciones. Este cargador se encarga de leer un programa binario enviado por UART y copiarlo en RAM, permitiendo cargar un programa sin necesidad de compilarlo junto con el circuito del procesador. Aunque este cargador inicialmente se implementó en ensamblador, se decidió pasarlo a la unidad de control para evitar que este dependa de las instrucciones implementadas en la misma. Este cargador recibe una lista de tuplas (dirección, nº palabras, bytes) terminada con una dirección 0 de tamaño 0. Tras esto, recibe la dirección de inicio del programa enviado, y comienza su ejecución. Cabe destacar que las palabras se envían en *little-endian*.

El correcto funcionamiento del circuito se ha verificado utilizando diferentes programas de prueba. Cada uno de estos verifica una de las funcionalidades implementadas. Estos programas se han ejecutado en el circuito *flasheado* en la FPGA, y se ha verificado el resultado de su ejecución manualmente.

Para verificar que todas las funcionalidades implementadas se integran correctamente, también se ha ejecutado un programa complejo. Más concretamente, se ha ejecutado ForthV [21], un intérprete del lenguaje de programación Forth escrito en ensamblador RISC-V. Este interprete define una gran parte de las funciones de Forth, e incluye un visor de memoria hexadecimal, un decompilador, y un gestor de memoria dinámica.

A. Integración en WepSIM

La implementación del circuito se ha integrado con el simulador WepSIM. Desde este se puede programar el *firmware* del circuito y sintetizarlo junto con el código VHDL. Tras esto, es posible *flashear* la FPGA y cargar un programa ensamblador enviado por UART. La interfaz para realizar este proceso se puede ver en la Figura 7. Estas funcionalidades se han mantenido separadas para permitir sintentizar el circuito solo cuando se cambia el *firmware*, ya que este proceso es significativamente más lento que solo cambiar el programa ensamblador. Como WepSIM es un simulador web, esto se realiza mediante una API REST y un servidor local encargado de hacer de intermediario entre la FPGA y WepSIM.

Además de programar la memoria de control de la unidad de control, se ha permitido *meta-programar* algunas características del circuito mediante el *firmware* enviado por WepSIM:

- Decodificación de instrucciones: se pueden especificar los patrones de bits que identifican a las instrucciones, junto con la dirección de la memoria de control en la que se encuentra su programa.

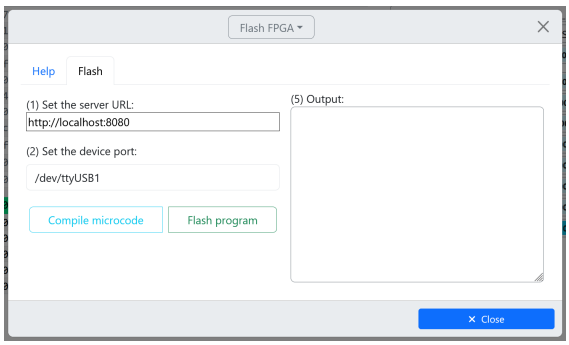


Fig. 7: Interfaz de WepSIM para compilar y *flashear* la FPGA.

- Decodificación de inmediatos: se puede definir la función que extrae el inmediato de una instrucción según un entero que representa su tipo de codificación.
- *Endianness* de la RAM: se puede seleccionar entre *little* y *big-endian*.

Este servicio web se ha empaquetado en un contenedor Docker, que contiene todas las dependencias necesarias para su uso. Sin embargo, debido a limitaciones de licencias, cabe destacar que para generar este contenedor es necesario descargar manualmente Vivado™[3], la herramienta para sintetizar VHDL y *flashearlo* en la FPGA indicada por el fabricante de la misma.

VI. Conclusiones

En este artículo se ha presentado el diseño y desarrollo de la implementación realizada en FPGA del Procesador Elemental de WepSIM.

Las principales conclusiones dados los resultados obtenidos son que se ha podido llevar un diseño mejorado del Procesador Elemental de WepSIM de forma muy satisfactoria a una FPGA. Aunque el diseño original del circuito del Procesador Elemental tenía algunos problemas que impedían su implementación directa en *hardware* basado en FPGA, estos se han podido corregir sin cambios significativos a su diseño inicial. Además se ha podido implementar el conjunto de instrucciones RV32I con la extensión Zmmul de RISC-V de forma microprogramada, validando que el diseño en FPGA con dicha memoria microprogramada permite ejecutar programas de RISC-V con las instrucciones implementadas sin problemas. Además, se ha integrado esta implementación en WepSIM, facilitando el uso del *hardware*.

Como trabajo futuro se podrían implementar otras extensiones, como por ejemplo la división, las operaciones de coma flotante, etc., aunque como ya se mencionó esto puede requerir algunas modificaciones al circuito para permitir operaciones de la ALU de varios ciclos. También se podrían implementar otras arquitecturas como MIPS para validar que el circuito es lo suficientemente genérico.

[3] `https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools.html`

Referencias

[1] James Newman, "The megaprocessor laughs at your puny integrated circuits," *IEEE Spectrum*, vol. 53, no. 10, pp. 24–25, September 2016.

[2] James Newman et al, "The MegaProcessor," 2017, Accedido en: 7 de abril de 2026.

[3] Noam Nisan Shimon Schocken, "From Nand to Tetris: building a Modern Computer From First Principles," 2008, Accedido en: 7 de abril de 2026.

[4] Felix Garcia-Carballeira, Alejandro Calderon-Mateos, Saul Alonso-Monsalve, and Javier Prieto-Cepeda, "Wepsim: an online interactive educational simulator integrating microdesign, microprogramming, and assembly language programming," *IEEE Transactions on Learning Technologies*, vol. 1, no. 1, pp. 0–8, 2 2019.

[5] "The RISC-V Instruction Set Manual Volume I," Especificación, RISC-V International, 2025.

[6] Anton Chichkov, "Chips Joint Undertaking – Working together to strengthen European industrie," 2026, Accedido en: 7 de abril de 2026.

[7] Javier Lacort, ""Mi familia me dice, ¿pero dónde están tus productos?": dentro del instituto español que está poniendo sus chips en Marte ," 2026, Accedido en: 7 de abril de 2026.

[8] Maicon Carlos Pereira, Paulo Viniccius Viera, André Luis Alice Raabe, and Cesar Albenes Zeferino, "A basic processor for teaching digital circuits and systems design with FPGA," in *2012 VIII Southern Conference on Programmable Logic*, 2012, pp. 1–6.

[9] Nikolay Kostadinov and Nina Bencheva, "Educational Tools for Processor Design Classes: A Case Study," in *2023 32nd Annual Conference of the European Association for Education in Electrical and Information Engineering (EAEEIE)*, 2023, pp. 1–4.

[10] Kamer Kırali and Can Bülent Fidan, "Implementation of FPGA based 32-bit RISC-V processor," *Engineering Science and Technology, an International Journal*, vol. 70, pp. 102139, 2025.

[11] D Chaver, S Harris, L Piñuel, O Kindgren, Z Kakakhel, C Owen, R Kravitz, J I Gómez, F Castro, K Olcoz, and R Owen, "The RISC-V FPGA (RVfpga) Teaching Package," *TechRxiv*, vol. 2026, no. 0220, 2026.

[12] Pablo Navarro-Torrero, Macarena C. Martínez-Rodríguez, Ángel Barriga-Barros, and Piedad Brox, "Full open-source implementation of an academic risc-v on fpga," in *2024 XVI Congreso de Tecnología, Aprendizaje y Enseñanza de la Electrónica (TAEE)*, 2024, pp. 1–5.

[13] Digilent, "Nexys A7," 2025, Accedido en: 27 de febrero de 2026.

[14] Digilent, "Nexys A7 Reference Manual," 2025, Accedido en: 27 de febrero de 2026.

[15] Advanced Micro Devices, "Supported VHDL-2019 Features," 2025, Accedido en: 1 de marzo de 2026.

[16] Nick Gasson, "NVC – Language Support," 2026, Accedido en: 1 de marzo de 2026.

[17] Frank Bruno and Guy Eschemann, *The FPGA Programming Handbook : An Essential Guide to FPGA Design for Transforming Ideas into Hardware Using SystemVerilog and VHDL*, Expert insight. Packt Publishing Ltd., Birmingham, UK, second edition. edition, 2024.

[18] SK Shenoy, "An xon/xoff-like ïn-band"flow control method for binary data transfers," 1999.

[19] "Determining Clock Accuracy Requirements for UART Communications," Documento técnico, Maxim Integrated, 2003.

[20] Advanced Micro Devices, "VHDL Constructs Support Status," 2025, Accedido en: 3 de marzo de 2026.

[21] José Antonio Verde Jiménez, "ForthV," 2026.

Evaluación de prestaciones (1/3)

Evaluación comparativa de la eficiencia energética entre CPU y GPU en tareas de machine learning

Emil Hristov, Belen Bermejo y Carlos Juiz[1],

Resumen— **Este trabajo explora la eficiencia energética en entornos de tareas de aprendizaje automático en servidores dotados con CPU y GPU. El objetivo principal es determinar qué unidad de procesamiento utilizar para un funcionamiento más eficiente para las tareas de aprendizaje automático. Mediante la evaluación empírica, basada en benchmarking, de tareas como la regresión lineal, la multiplicación de matrices y el aumento de gradiente, observamos que, si bien las GPU generalmente ofrecen un rendimiento más rápido y una mayor eficiencia energética para operaciones a gran escala, ciertas tareas ligeras consumen menos energía total cuando se ejecutan en CPU. Estos hallazgos respaldan la idea de la programación orientada a tareas, donde las tareas más pequeñas deberían asignarse preferentemente a la CPU para reducir el consumo total de energía del sistema.**

Palabras clave— **Consumo energético, Inteligencia Artificial, Aprendizaje Automático, GPU, CPU**

I. Introducción

El incremento en el uso de las TIC está dificultando la gestión de los mismos y también está afectando al consumo eléctrico de forma impactante. Esto se debe, entre otros aspectos, a la necesidad de cómputo debido a los algoritmos de *machine learning* que suelen utilizar en los procesadores gráficos (GPUs) [1] [2] de los servidores. Además, se ha detectado que hay cargas de trabajo que se ejecutan de forma más eficiente en términos energéticos en la CPU que en la GPU [3]. Sin embargo, no existe ningún estudio sistemático al respecto más allá de un estudio comparativo dentro del mismo ámbito [4]. El reparto de trabajo optimiza el uso de los recursos de los servidores haciendo uso del dispositivo adecuado para cada tarea de *machine learning*, ya sea durante el preprocesamiento de los datos, la ejecución de algoritmos o el entrenamiento de los modelos [5].

Por todo ello, en este trabajo se realiza un estudio empírico de la carga del sistema a la hora de ejecutar tareas, como entrenamiento de modelos o tareas de preprocesamiento [6] [7], utilizando las CPUs o las GPUs de los servidores en condiciones de carga homogéneas y heterogéneas. El objetivo es ver una clara distinción entre el consumo energético en ciertas cargas y así poder redistribuir el trabajo entre las CPUs y GPUs.

[1]Dpto. de Informática, Universitat de les Illes Balears, e-mail: emil7234@icloud.com, belen.bermejo@uib.es, cjuiz@uib.es

II. Conceptos relacionados

El objetivo de este trabajo es comparar las ejecuciones en CPU y GPU de las cargas de ML que se seleccionen para este trabajo.

A. Estructura y funcionamiento: CPU vs. GPU

Como el análisis realizado en este trabajo se fundamenta en el contraste de las arquitecturas de procesamiento (CPU, GPU), es necesario explicar en qué se basa cada una de ellas. Las arquitecturas CPU se basan en optimizar la ejecución secuencial de trabajos con el mínimo número de hilos en paralelo. Lo ideal para esta arquitectura son tareas no paralelizables, con control de flujo, procesamient ode datos y lógicas condicionales. Por otro lado, las arquitecturas basadas en GPUs están diseñadas para la ejecución masiva y paralela. También, son muy eficientes para cálculos matemáticos paralizables como multiplicación de matrices y convoluciones (típicos en el aprendizaje profundo) [8] [9].

B. Cuello de botella: overhead de transferencia de datos

En los servidores de alto rendimiento, basados en arquitecturas x86, CPU y GPU, con unidades físicas separadas con memoria propia (RAM del sistema y vRAM), la comunicación se realiza mediante el bus PCI Express (PCIe). La necesidad de una transparencia explícita de datos de RAM a vRAM introduce una sobrecarga (overhead) que consume tiempo, y en consecuncia, energía. Este coste adicional puede convertirse en un cuello de botella, especialmente para cargas de trabajo pequeñas o que requieren transferencias frecuentes [10].

C. Cargas de trabajo de Machine Learning (benchmarks)

Los benchmarks representativos que se han tenido en cuenta en este estudio en relación al ML y DL son los siguientes:

- Multiplicación de matrices: peración intensiva en cálculos de punto flotante. Núcleo fundamental de ML/DL. Ideal para explotar el paralelismo masivo de la GPU [1] [11].
- Regresión lineal: Basada en operaciones secuenciales y de baja latencia (multiplicaciones de vectores, gradiente). Puede ser más eficiente en la CPU en conjuntos de datos moderados, donde el *overhead* de la GPU no se justifica [12].

- Árboles de Decisión / XGBoost: Modelos basados en particiones binarias. XGBoost (Gradient Boosting) es conocido por su velocidad, escalabilidad y optimización para el procesamiento en paralelo [13].
- Transformers: Arquitectura basada en autoatención (ej. GPT-2, Whisper). Permite el procesamiento paralelo de secuencias, siendo clave en PLN. Whisper es un modelo end-to-end de voz a texto [14].

III. Metodología de experimentación

El objetivo principal de este estudio es realizar una evaluación empírica de la eficiencia energética y el rendimiento comparado entre la Unidad Central de Procesamiento (CPU) y la Unidad de Procesamiento Gráfico (GPU) en tareas representativas de *machine learning*. El fin es identificar el umbral de carga computacional a partir del cual el uso de la CPU resulta más eficiente que el de la GPU, considerando el tiempo de ejecución y el consumo total de energía.

La metodología se basa en la clásica de ingeniería del rendimiento de los sistemas informáticos [15], haciendo uso de una batería de experimentos diseñados para escalar la carga computacional desde tareas *super light* (regresión lineal y árboles de decisión con datos reducidos) hasta escenarios de *carga extrema* (multiplicación de matrices y *gradient boosting* con alta dimensionalidad). Las pruebas incluyen modelos clásicos de aprendizaje automático, operaciones base como la multiplicación de matrices, y modelos de redes neuronales profundas tipo *transformer* (GPT-2 y Whisper) para evaluar la inferencia intensiva en procesamiento de lenguaje natural y audio.

A. Entorno de prueba y medición

La evaluación se lleva a cabo en un entorno de hardware homogéneo, utilizando una CPU Intel Core i7-13700 y una GPU Nvidia GeForce RTX 4080. Las pruebas se diseñaron para variar la complejidad de los datos de entrada, manteniendo controladas el resto de las variables de ejecución.

Un bloque de pruebas complementario se ejecuta en una configuración orientada a centros de datos (CPU Xeon W-2245 y GPU RTX A5000) con el modelo Whisper, buscando analizar el impacto de la generación y arquitectura del hardware profesional en la eficiencia general del sistema.

Es crucial notar que la medición de la eficiencia se realiza capturando el **consumo total del sistema** en Watts (W) a través de un osciloscopio, en lugar de medir el consumo individual de la CPU o la GPU, para proporcionar una correlación más fiable con el consumo energético global (ver Figura 1).

B. Escenarios clave

El estudio segmenta las pruebas en bloques para evaluar distintas hipótesis:

- Cargas Ligeras (Super, Extremely Light): Se explora si el *overhead* de la transferencia de datos a la GPU en tareas de baja complejidad hace que la CPU, con su menor latencia y optimización para procesos secuenciales, resulte más eficiente.
- Cargas Media y Extrema (Mid, Extreme): Se lleva al límite la capacidad de la GPU en tareas demandantes con gran volumen y dimensionalidad de datos, donde se espera que la paralelización de la GPU ofrezca la ventaja más significativa.
- Modelos *Transformer (GPT-2, Whisper):* Aunque estos modelos de redes neuronales profundas están optimizados para GPU, se evalúa si para tareas de inferencia muy breves (como la generación de 500 *tokens* o la transcripción de audios cortos), el coste inicial de la GPU podría anular su beneficio.

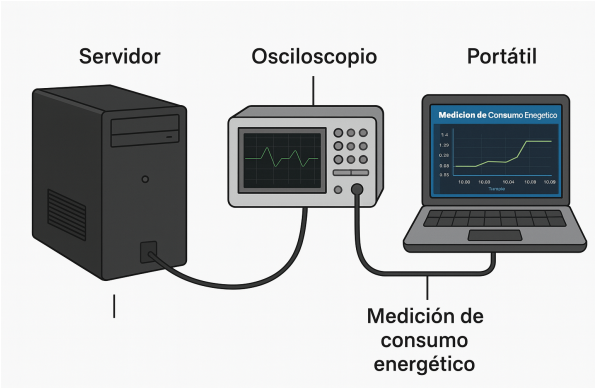


Fig. 1: Esquema de la toma de medidas. Se puede ver el servidor conectado a un osciloscopio para medir su consumo energético total. El portátil es el que se encarga de registrar los datos del osciloscopio y luego generar archivos .txt con el consumo instantáneo tomado a lo largo de la prueba. Este consumo incluye todo el sistema por lo tanto incluye tanto la GPU como la CPU.

El análisis de estos escenarios busca ofrecer una guía detallada sobre la mejor elección de unidad de procesamiento en función de la naturaleza y el volumen de la carga de trabajo de *machine learning*.

IV. Resultados del estudio

Esta sección presenta un resumen conciso de los resultados de un estudio comparativo sobre la eficiencia energética y el tiempo de ejecución entre la Unidad Central de Procesamiento (CPU) y la Unidad de Procesamiento Gráfico (GPU) bajo diferentes cargas de trabajo (Light, Mid, Extreme, LLMs y modelos Transformer). El análisis se centra en identificar el umbral de carga a partir del cual una arquitectura supera a la otra, considerando factores como la variabilidad de las mediciones, el *overhead* de transferencia de datos y la estabilidad del consumo.

A. Variabilidad y consistencia

Se observó una ligera variabilidad en el consumo energético absoluto de la CPU entre diferentes sesiones de prueba. No obstante, la diferencia relativa en la eficiencia entre la CPU y la GPU se mantuvo **consistente** a lo largo de las mediciones, lo que valida la comparativa general. Los picos puntuales de consumo en reposo se atribuyeron a procesos externos del sistema operativo.

B. Comportamiento de las cargas de trabajo

En las cargas ligeras y extraligeras (Super Light / Extremely Light) la CPU mostró mayor eficiencia en algoritmos poco paralelizables y con bajo volumen de datos, como la regresión Lineal y el árbol de decisión simple, superando el consumo total y el pico máximo de la GPU en el caso del Árbol de Decisión. Esto se debe principalmente a que la CPU evita el *overhead* de transferencia de datos a la VRAM de la GPU. Sin embargo, para Multiplicación de Matrices y XGBoost, la GPU se mantuvo más eficiente incluso con cargas mínimas.

En caso de las cargas medias y extremas (Mid / Extreme), la GPU demostró una superioridad indiscutible en tiempo de ejecución y eficiencia energética global. En la carga Extreme, la GPU fue aproximadamente 2.5 veces más eficiente que la CPU en consumo total (32kW vs 81kW), gracias a la drástica reducción del tiempo de ejecución. Aunque la GPU presentó picos de consumo más del doble que la CPU, esta desventaja se compensa con la rapidez. La CPU, por su parte, mostró un perfil energético notablemente más estable.

C. Modelos de lenguaje y transcripción (LLMs y Whisper)

En los modelos GPT2 (LLMs), la GPUfue superior, completando la tarea 14 veces más rápido y resultando hasta 5 veces más eficiente energéticamente que la CPU. La arquitectura transformer se beneficia al máximo del procesamiento paralelo de la GPU.

Por otro lado, en el modelo Whisper (Transcripción), en la prueba de audio corto, la GPU fue 7.5 veces más rápida. En una comparación de alta carga entre GPUs:

- La RTX 4080 (gama consumo) superó a la RTX A5000 (gama profesional) en tiempo de procesamiento (e.g., 650s vs 986s para audio largo) y calidad de transcripción.
- La 4080 fue globalmente más eficiente gracias a su arquitectura moderna, a pesar de tener un límite de potencia superior.

D. Resultados comparativos de eficiencia CPU vs GPU

La Tabla I recoge los tiempos de ejecución y el consumo energético de cada prueba, incluyendo además una columna que indica el valor máximo registrado por cada componente durante la medición.

Prueba	Operación	CPU (s)	GPU (s)	CPU (W)	GPU (W)	Idle mean (W)	CPU (W Max)	GPU (W Max)
super light	Matrix multiplication	0.0019	0.0016	3209.78	2322.96	55.75	192.8	221
	Linear Regression	0.2702	0.0323					
	XGBoost Training	0.5099	0.2692					
Extremely lighter v2	XGBoost Training	0.0731	0.1368	812.1	982.65	51.76	125	113.89
	DecisionTree	0.0026	0.0025					
light_load_test (Sesión 1)	Matrix multiplication	0.28	0.03	580	871.38	61.7	107	207
	Linear Regression	0.02	0.02					
	XGBoost Training	0.52	0.25					
mid_load_test (Sesión 1)	Matrix multiplication	18.34	0.03	9446.66	5128.83	61.7	209	427
	Linear Regression	4.29	4.03					
	XGBoost Training	25.21	4.01					
extreme_load_test (Sesión 1)	Matrix multiplication	60.17	0.04	63487	11105	61.7	224.7	448
	Linear Regression	40.52	38.34					
	XGBoost Training	293.12	15					
light_load_test (Sesión 2)	Matrix multiplication	0.38	0.04	1767.8	976	58.76	170	213
	Linear Regression	0.02	0.01					
	XGBoost Training	0.52	0.25					
mid_load_test (Sesión 2)	Matrix multiplication	14.55	0.03	17845	6100	58.76	215	467
	Linear Regression	4.17	4.17					
	XGBoost Training	25.1	4.01					
extreme_load_test (Sesión 2)	Matrix multiplication	58.57	0.05	81606	32553	58.76	223	428
	Linear Regression	40.59	37.51					
	XGBoost Training	162	14.88					
hugging_face_test (Sesión 1)	LLMs (GPT2)	70.05	4.98	19636	4264.55	61.7	128.75	275.2
hugging_face_test (Sesión 2)	LLMs (GPT2)	70.8	5.07	19217	3649	58.76	146	273

Tabla I: Resumen de los resultados obtenidos para la ejecución de las cargas en CPU y GPU: tiempos de respuesta (s) y consumo de potencia (W)

V. Discusión de los resultados

El estudio evaluó empíricamente el consumo energético realizando una comparativa entre CPU (arquitectura x86) y GPU en tareas de *machine learning* con distintas cargas y modelos.

A. Hallazgos principales

Los resultados confirman que la GPU demuestra una superioridad clara en escenarios de alta demanda computacional (pruebas *mid*, *extreme*, modelos *transformer* como GPT-2 y Whisper). La arquitectura paralela de la GPU reduce drásticamente el tiempo de ejecución, compensando el *overhead* de transferencia de datos y los picos de consumo instantáneo más elevados. Por ejemplo, en la prueba extrema, la GPU fue 2.5 veces más eficiente energéticamente que la CPU.

Por otro lado, la CPU exhibe mayor eficiencia energética en tareas ligeras, secuenciales o con bajo volumen de datos, como regresión lineal simple o árboles de decisión. Las pruebas *super light* y *extremely light* evidencian esta ventaja, ya que la CPU evita el costo energético del *overhead* y se beneficia de su menor latencia y arquitectura secuencial optimizada.

El análisis comparativo entre la RTX 4080 (consumo) y la RTX A5000 (profesional) demostró que la eficiencia depende crucialmente de la arquitectura interna y la generación del hardware, superando el modelo más moderno al profesional en rendimiento y eficiencia.

Desde una perspectiva práctica en centros de datos, se concluye que una estrategia óptima de asignación dinámica de cargas debe considerar el volumen de datos, el tipo de modelo y la arquitectura específica del hardware, y no priorizar el uso de GPU de forma indiscriminada.

B. Umbral de decisión

En base a los resultados obtenidos anteriormente, podemos determinar un umbral de decisión para la ejecución de las cargas en CPU o GPU. La CPU es más óptima para la ejecución de tareas más ligeras, secuenciales o con bajo volumen de datos, como las regresiones simples y árboles de decisión poco profundos. En cuanto a la GPU, ésta es más eficiente en cargas masivas, modelos altamente paralelizables o en situaciones donde la latencia total de ejecición es crítica. También, es necesario tener en cuenta que la sobrecarga (overhead) provocado es perjudicial para la GPU en tareas pequeñas, pero insignificantes en contextos de alta carga. Por lo tanto, podríamos decir que la eficiencia está intrínsecamente ligada a la generación y arquitectura del hardware.

C. Limitaciones del estudio

El estudio presenta limitaciones clave que acotan la generalización de los resultados.

En cuanto la dependencia del hardware, la experimentación se basó en un único sistema (CPU Intel Core i7-13700 y GPU Nvidia RTX 4080), restringiendo la validez de los resultados a otras arquitecturas (e.g., ARM) o configuraciones.

Las restricciones de vRAM también se deben tener cuenta, ya que la memoria gráfica disponible (16 GB) limitó el tamaño de los modelos y la complejidad de las cargas de trabajo.

Sobre las mediciones realizadas, el consumo se midió a nivel de sistema completo, sin un aislamiento preciso y confiable del aporte específico de la CPU o GPU. También, se detectó cierta variabilidad en el consumo (atribuible a condiciones térmicas o procesos de fondo no controlables) e imposibilidad de estandarizar completamente las condiciones operativas.

VI. Conclusiones y trabajo futuros

A. Conclusiones

El estudio alcanzó sus objetivos al determinar empíricamente las condiciones bajo las cuales la CPU es más eficiente que la GPU en tareas de *machine learning*. Mediante una metodología de benchmarking estructurada por niveles de carga, se lograron identificar patrones de comportamiento energético basados en el tipo de modelo, el volumen de datos y la arquitectura del hardware.

La CPU es energéticamente superior en tareas ligeras, con datos reducidos y modelos poco complejos (ej. regresión simple). Su ventaja radica en su menor *overhead* y su eficiencia en operaciones secuenciales. La GPU es la opción más eficiente en escenarios de alta demanda de paralelismo y gran volumen de datos (ej. modelos *transformer* o XGBoost). Logra reducir drásticamente el tiempo de ejecución y el consumo energético total. La eficiencia no solo depende del tipo de procesador, sino también de su generación y arquitectura interna. Contar con hardware actualizado es un factor crucial para el rendimiento y el consumo.

Este trabajo proporciona un marco metodológico reproducible y datos empíricos valiosos para la optimización de la eficiencia energética en infraestructuras como los centros de datos (CD), promoviendo un uso más inteligente de los recursos computacionales.

B. Líneas de trabajo futuro

Tras el trabajo realizado, se proponen las siguientes líneas de trabajo futuro. Se plantea incluir mediciones del consumo energético diferenciadas y precisas a nivel de componentes (CPU y GPU) mediante dispositivos o herramientas externas. También, se plantea extender las pruebas a un abanico más amplio de componentes de la arquitectura x86 para aumentar la capacidad de generalización de los resultados. Además, se podrían incluir pruebas con otros tipos de modelos de ML, como redes convolucionales, algoritmos de clustering o reinforcement learning. Por último, se plantea diseñar una metodología automatizada para la toma de decisones en tiempo real sobre la asignación dinámica de cargas CPU-GPU basada en el perfil energético observado.

Referencias

[1] Norman P Jouppi, Cliff Young, Nishant Patil, et al., "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.

[2] Ebubekir Buber and Banu Diri, "Performance analysis and cpu vs gpu comparison for deep learning," in *2018 International Conference on Computer Science and Engineering (CEIT)*. 2018, IEEE.

[3] Roblex Nana Tchakoute, Claude Tadonki, Petr Dokladal, and Youssef Mesri, "Benchmark-based study of cpu/gpu power-related features through jax and tensorflow," *arXiv preprint arXiv:2505.03398*, 2025.

[4] David Cortes, Carlos Juiz, and Belen Bermejo, "Estudio de la eficiencia en la escalabilidad de gpus para el entrenamiento de inteligencia artificial," *arXiv preprint arXiv:2509.03263*, 2025.

[5] IBM, "Cpu vs. gpu in machine learning," `https://www.ibm.com/think/topics/cpu-vs-gpu-machine-learning`, 2024, Accessed: 2025-06-03.

[6] Alec Radford, Jong Wook Gao, Tao Xu, Greg Brockman, Jeff McLeavey, and Ilya Sutskever, "Robust speech recognition via large-scale weak supervision," *arXiv preprint arXiv:2212.04356*, 2022.

[7] scikit-learn developers, "Frequently asked questions: Does scikit-learn support gpu acceleration?," `https://scikit-learn.org/stable/faq.html#does-scikit-learn-support-gpu-acceleration`, 2024, Accessed: 2025-06-03.

[8] Mário Véstias and Horácio Neto, "Trends of cpu, gpu and fpga for high-performance computing," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, 2014, pp. 1–6.

[9] Sparsh Mittal and Jeffrey S Vetter, "A survey of cpu-gpu heterogeneous computing techniques," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, pp. 1–35, 2015.

[10] Intel Corporation, "What are discrete and integrated gpus?," 2023, Accedido el 6 de agosto de 2025.

[11] Jack Dongarra, Michael A. Heroux, and Piotr Luszczek, "A new metric for ranking high performance computing systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2017.

[12] IBM Corporation, "What is linear regression?," `https://www.ibm.com/think/topics/linear-regression`, 2024, Consultado el 18 de junio de 2025.

[13] IBM Corporation, "Árboles de decisión: ¿qué son y cómo se usan?," `https://www.ibm.com/es-es/think/topics/decision-trees`, 2023, Consultado el 18 de junio de 2025.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5998–6008.

[15] Xavier Molero, Carlos Juiz, and Miguel Jesús Rodeño, "Evaluaci\'on y modelado del rendimiento de los sistemas inform\'aticos," *arXiv preprint arXiv:2508.16996*, 2025.

Estimación en Tiempo Real de la Planitud en Bandas de Acero Laminadas mediante un Modelo Generativo Condicionado

Darío G. Lema [1], F.J. delaCalle [1], Rubén Usamentiaga [1] y Joaquín Entrialgo [1]

Resumen— **Este artículo aborda la predicción de la planitud en bandas de acero laminadas en estado libre de tensiones. Durante el laminado, las tensiones inducidas por los rodillos ocultan ondulaciones que solo afloran en reposo, comprometiendo la calidad industrial. Para resolverlo, se propone el modelo Target-Conditioned Vector-Quantized Variational Autoencoder (TC-VQ-VAE). Esta arquitectura aprende la transformación entre la geometría bajo tensión y el estado final objetivo, integrando parámetros físicos como la fuerza de laminación y las dimensiones de la banda. Los resultados experimentales demuestran que el TC-VQ-VAE supera a modelos como LSTM y DDPM, alcanzando un coeficiente de determinación (R^2) de 0.83 y un error absoluto medio de solo 0.37 %. Con un tiempo de procesamiento de 0,0009 segundos por fibra, el modelo es idóneo para aplicaciones industriales en tiempo real, permitiendo un control de calidad inmediato y optimizando los costes operativos frente a métodos tradicionales.**

Palabras clave— **Flatness prediction, Steel rolling, Deep learning, Generative AI, Real-time prediction, Industry**

I. Introducción

El laminado es un proceso industrial en el que el espesor de una pieza metálica se reduce progresivamente al pasarla entre rodillos que aplican fuerzas de compresión controladas [1]. Este proceso, que puede realizarse en caliente o en frío, genera esfuerzos residuales que influyen directamente en la forma final del material [2]. Entre estos efectos, uno de los más importantes es la planitud de la banda, definida como la proximidad de la pieza a un plano ideal [3]. Mantener una planitud adecuada no solo asegura la calidad del producto, sino que también facilita las operaciones posteriores de manufactura. Por ello, la industria demanda sistemas de medición y control precisos, capaces de evaluar la planitud en tiempo real y garantizar la eficiencia del proceso.

Durante el laminado, las bandas de acero presentan ondulaciones que suelen reducirse por las tensiones de tracción longitudinal inducidas por los rodillos. Esto puede llevar a que las mediciones de planitud durante el proceso indiquen valores más favorables que los que se observarían una vez eliminadas las tensiones. En la práctica, esto significa que una banda puede cumplir con los estándares de calidad bajo tensión, pero presentar ondulaciones al regresar a un estado libre de esfuerzo, superando los límites permitidos. Este fenómeno plantea un desafío crítico: predecir en tiempo real la planitud de la banda en un estado libre de tensiones, a partir de:

[1]Dpto. de Informática, Universidad de Oviedo (Gijón, Asturias), e-mail: gonzalezdario@uniovi.es

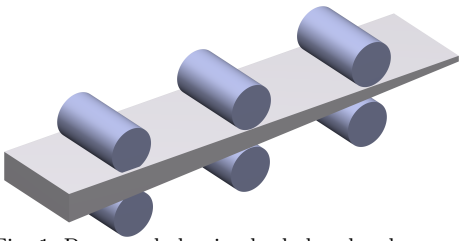


Fig. 1: Proceso de laminado de bandas de acero.

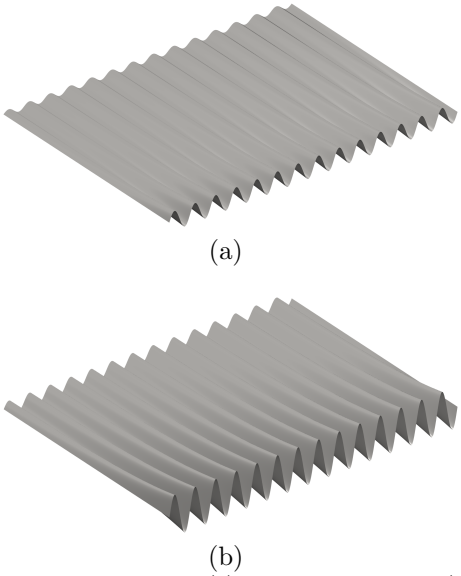
- La amplitud de las ondulaciones medidas durante el laminado bajo tensión.
- Los parámetros físicos aplicados en el proceso, como fuerza remota, ancho y longitud de la banda.

Para resolver este desafío, en este trabajo se propone un modelo generativo condicionado basado en redes neuronales profundas. El modelo estima la planitud de la banda en un estado libre de tensiones utilizando como entradas las ondulaciones medidas bajo tensión y los parámetros del laminado. Su capacidad de realizar predicciones en tiempo real permite mejorar el control del proceso y garantizar la calidad del producto final.

II. Contexto industrial

Las bandas de acero se someten al laminado para reducir su espesor y alcanzar dimensiones precisas requeridas en aplicaciones industriales. Durante este proceso, la banda atraviesa un tren de rodillos que aplican fuerzas de compresión controladas, reduciendo progresivamente su grosor. La Figura 1 ilustra este proceso. Aunque esta operación asegura uniformidad dimensional, también induce deformaciones internas que afectan directamente la planitud de la banda. La Figura 2 ilustra este efecto: en la Figura 2a se observa la banda bajo tensión, durante el laminado, mientras que en la Figura 2b se muestra la misma banda una vez liberada de las fuerzas aplicadas. Se aprecia que la planitud es mayor bajo tensión, pero al eliminarse las fuerzas de los rodillos aparecen ondulaciones que pueden superar los límites permitidos.

La planitud se evalúa considerando la banda como un conjunto de fibras longitudinales contiguas, donde cada fibra es una línea que recorre la banda en dirección de laminación. Una banda perfectamente plana tendría todas sus fibras con la misma longitud, mientras que diferencias en las longitudes generan ondulaciones en la superficie, manifestándose como defectos de planitud. La magnitud de estas on-



(a) (b)

Fig. 2: Proceso de laminado: (a) Banda bajo tensión. (b) Banda sin tensión.

dulaciones se mide mediante la altura máxima entre picos y valles de las fibras, proporcionando un indicador claro de planitud.

La geometría de la banda puede representarse mediante un mapa de alturas, construido a partir de la amplitud de cada fibra a lo largo del ancho de la banda. Este mapa es la base para calcular la planitud mediante las I-units (IU), un índice determinado para cada fibra. Para calcular las I-units se utiliza la Ecuación (1), donde L_i es la longitud de la fibra i y L_{ref} la longitud de referencia, que puede ser la fibra más corta o el promedio de todas las fibras. La longitud de cada fibra se calcula considerando sus ondulaciones, representando la fibra mediante M puntos a lo largo de la dirección Y, separados por Δy, y con alturas z_i en cada punto.

$$IU_i = \frac{L_i - L_{ref}}{L_{ref}} \times 10^5 \tag{1}$$

En la práctica, la medición de la amplitud de las fibras se realiza bajo tensión, durante el paso por el rodillo sensor. Sin embargo, la planitud en estado libre de tensión no puede obtenerse directamente. Por ello, este trabajo propone un método para predecir la amplitud de cada fibra sin tensión a partir de mediciones bajo tensión, permitiendo reconstruir el mapa de alturas y estimar la planitud real de la banda.

III. Trabajo relacionado

Durante décadas se han desarrollado métodos para medir y predecir la planitud de bandas de acero en estado libre de tensión. Los enfoques iniciales eran manuales, basados en inspección visual de secciones de la banda, pero su precisión dependía de la experiencia del operario y resultaba poco práctico para la producción continua.

Recientemente, se han propuesto métodos automáticos basados en redes neuronales y aprendizaje de máquina. Por ejemplo, [4] emplea una red profunda tipo Inception-ResNet-39 para predecir planitud durante el laminado en frío, alcanzando buenos resultados de MAE, RMSE y R^2. Otros estudios, como [5], utilizan técnicas de ensamblado (bagging, boosting, stacking) sobre diversos modelos base, mejorando significativamente la precisión. En [6] se combinan métodos de aprendizaje por refuerzo y ensamblado para controlar la planitud, superando en precisión y robustez a controladores convencionales. Por último, [7] propone un enfoque basado en Inception con detección de valores atípicos para predecir la planitud a partir de representaciones matriciales de las características medidas.

A pesar de estos avances, la mayoría de los estudios se enfocan en la planitud durante el laminado, cuando la banda todavía está bajo tensión. El desafío de predecir la planitud en estado libre de tensión permanece abierto.

Este trabajo aborda dicho desafío mediante un modelo generativo condicionado, el Target-Conditioned Vector-Quantized Variational Autoencoder (TC-VQ-VAE). A diferencia de los autoencoders convencionales, el TC-VQ-VAE se entrena para aproximar directamente un estado objetivo distinto al de entrada, aprendiendo la transformación entre la banda bajo tensión y su geometría final libre de tensiones. Esto permite predecir tempranamente la forma real de la banda, facilitando decisiones industriales y optimizando tiempo, energía y costes.

IV. Modelo propuesto basado en IA generativa

Los modelos convencionales de IA generativa aprenden a generar nuevos ejemplos a partir de la distribución de entrenamiento, pero su salida no está condicionada a un objetivo específico. En este trabajo, se propone un Target-Conditioned Vector-Quantized Variational Autoencoder (TC-VQ-VAE) para modelar la relación entre las fibras de acero bajo tensión y su estado libre de tensión. A diferencia de los enfoques estándar, la generación aquí es condicional y determinista, permitiendo reconstruir la geometría de la fibra sin tensión a partir de su versión deformada.

El TC-VQ-VAE codifica la fibra de entrada en un espacio latente discreto y la reconstruye en la salida, preservando características locales y globales. Para capturar información contextual adicional, se incorporan los parámetros físicos de la banda —fuerza remota y dimensiones de cada fibra— concatenados a la señal antes del codificador. También se integran mecanismos de atención y capas intermedias para modelar relaciones locales a lo largo de la fibra.

El entrenamiento combina varias funciones de pérdida:

- Pérdida de reconstrucción: para asegurar fidelidad con la fibra objetivo
- Espacio latente: para garantizar consistencia en el codificador discreto
- Pérdida adversarial: para mejorar realismo y reducir distorsiones

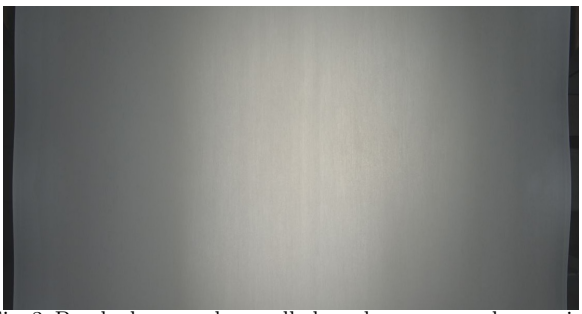


Fig. 3: Banda de acero desenrollada en la que se puede apreciar las ondulaciones producidas tras el laminado

- Pérdida perceptual: mediante LPIPS, para capturar diferencias estructurales y texturales

La pérdida total se pondera mediante coeficientes para equilibrar los distintos objetivos. Esta estrategia permite que el modelo aprenda un mapeo preciso y reversible entre fibras bajo tensión y libres de tensión, manteniendo la coherencia física y mejorando la calidad perceptual de la reconstrucción.

V. Conjunto de datos

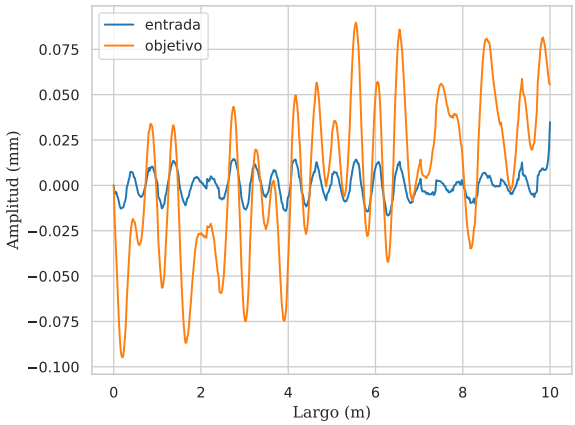
El entrenamiento del modelo propuesto requiere datos que representen tanto el estado de la banda de acero bajo tensión durante el laminado como su condición libre de tensiones. La Figura 3 muestra una banda desenrollada, donde las ondulaciones ocultas durante el laminado se hacen visibles, destacando la necesidad de información de ambos dominios.

Para ello, se creó un conjunto de datos real basado en datos de una acería. Se generó mediante simulaciones de elementos finitos (FEM) y captura las fibras de acero con y sin tensión. Durante el laminado, se registran las amplitudes de las fibras a lo largo de la banda, y se incorporan parámetros físicos clave como fuerza remota, largo y ancho de cada fibra. La salida objetivo del modelo se obtiene mediante un FEM lineal elástico validado en planta, que calcula la geometría libre de tensión de cada fibra. Aunque este modelo es demasiado costoso para uso en tiempo real, proporciona una referencia confiable para entrenar el modelo propuesto.

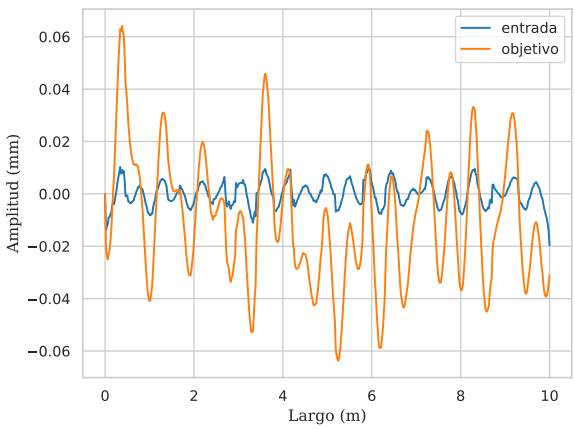
Dadas las dimensiones de las bandas (10 m de largo × 1,5 m de ancho) y su espesor relativamente pequeño, se realiza un análisis unidimensional a lo largo de fibras discretizadas. Cada fibra se representa mediante 500 puntos longitudinales, conectados mediante elementos viga Euler-Bernoulli 1D (B23), con sección transversal definida por el espesor de la banda y el ancho de la fibra. En cuanto a condiciones de contorno, los desplazamientos en dos direcciones se restringen en un extremo, mientras que en el extremo opuesto se limita solo el movimiento vertical y se aplica una carga axial longitudinal uniformemente distribuida. Esta estrategia evita fenómenos de pandeo y no linealidades que no ocurren en la banda real al liberar la tensión.

La Figura 4 ilustra ejemplos representativos:

- La fibra bajo tensión (color azul) presenta menor amplitud, mientras que la fibra objetivo libre de tensión (color naranja) recupera su forma y muestra mayor amplitud
- Algunas fibras objetivo muestran valores negativos debido a ondulaciones hacia la cara inferior de la banda
- En general, la geometría bajo tensión se alinea coherentemente con la geometría libre de tensión obtenida mediante FEM, confirmando la validez del conjutno de datos para entrenar el modelo



(a)



(b)

Fig. 4: Ejemplos representativos del conjunto de datos: fibra bajo tensión (azul) y fibra libre de tensión (naranja).

VI. Resultados experimentales

A. Métricas de evaluación

Para evaluar el enfoque propuesto y compararlo objetivamente con otros modelos, se emplean métricas estándar ampliamente utilizadas en la literatura. Cada una aporta una perspectiva diferente del rendimiento del modelo, considerando la magnitud del error, la capacidad para explicar la variabilidad de los datos y la sensibilidad a la amplitud de la señal. En conjunto, permiten una evaluación integral y equilibrada.

- MSE. El Error Cuadrático Medio (Mean Squared Error) calcula el promedio de los errores al cuadrado entre las predicciones del modelo y los valores reales. Penaliza con mayor intensidad los errores grandes, por lo que es sensible a valores atípicos. Valores más bajos de MSE indican una

mayor precisión global, aunque su interpretación puede resultar menos intuitiva al no estar en la misma escala que los datos originales.

- RMSE. La Raíz del Error Cuadrático Medio (Root Mean Squared Error) es la raíz cuadrada del MSE, lo que permite expresar el error en la misma unidad que los datos originales. Mantiene la sensibilidad a errores grandes, pero facilita la interpretación directa. Valores más bajos de RMSE reflejan mayor fidelidad en las predicciones.
- MAE. El Error Absoluto Medio (Mean Absolute Error) calcula el promedio de las diferencias absolutas entre los valores reales y los valores predichos. Es más sencillo de interpretar que el MSE y el RMSE, ya que representa el error medio absoluto. Además, es menos sensible a valores atípicos. Un MAE más bajo indica mejores predicciones.
- NMAE. El Error Absoluto Medio Normalizado (Normalized Mean Absolute Error) es similar al MAE, pero se normaliza respecto a un valor de referencia global, como el rango de la señal. Esto lo convierte en una métrica adimensional, útil para comparar errores entre señales con diferentes escalas.
- R^2. El Coeficiente de Determinación cuantifica la proporción de la variabilidad de los datos que es explicada por el modelo. Un valor cercano a 1 indica un buen ajuste, mientras que valores cercanos a 0 o negativos reflejan un desempeño deficiente. Esta métrica evalúa la calidad global del ajuste.

En conjunto, estas métricas permiten analizar tanto la precisión absoluta y relativa de las predicciones como la capacidad del modelo para capturar la variabilidad de los datos, ofreciendo una evaluación completa del rendimiento.

B. Predicción de amplitudes de fibras en estado sin tensión

Esta sección presenta los resultados obtenidos al comparar el modelo propuesto TC-VQ-VAE con diversas arquitecturas de referencia empleadas comúnmente en procesamiento de secuencias y generación de señales, utilizando únicamente datos reales de fibras medidas en planta. El objetivo es evaluar la capacidad de cada modelo para aprender la relación entre fibras bajo tensión y sus correspondientes fibras sin tensión, considerando tanto la fidelidad de reconstrucción como la consistencia física.

Los resultados se muestran en la Tabla I. Modelos como LSTM presentan errores relativamente altos y un R^2 negativo, indicando dificultad para capturar la relación entre fibras bajo tensión y sin tensión. DCGAN muestra una ligera mejora, pero mantiene errores significativos y baja correlación con los datos reales. SIG2SIG reduce notablemente los errores, con un NMAE de 0.98 % y $R^2 = 0.43$, capturando buena parte de la tendencia general. Entre los modelos generativos, DDPM destaca con MSE = 0.0003, NMAE = 0.7 % y $R^2 = 0.71$, mostrando alta fidelidad y coherencia física. La versión ruidosa de DDPM reduce su rendimiento ($R^2 = 0.22$, NMAE = 1.14 %).

Tabla I: Resultados de evaluación de los modelos sobre el conjunto de datos, mostrando MSE, RMSE, MAE, NMAE y R^2.

	MSE	RMSE	MAE	NMAE	R^2
LSTM	0.0011	0.033	0.027	1.31 %	-0.23
DCGAN	0.0017	0.041	0.035	1.70 %	-0.18
SIG2SIG	0.0006	0.024	0.020	0.98 %	0.43
DDPM	0.0003	0.017	0.014	0.70 %	0.71
Noisy DDPM	0.0008	0.028	0.024	1.14 %	0.22
TC-VQ-VAE	**0.0002**	**0.014**	**0.008**	**0.37 %**	**0.83**

El modelo propuesto TC-VQ-VAE supera consistentemente a todas las demás arquitecturas, logrando MSE = 0.0002, NMAE = 0.37 % y $R^2 = 0.83$, capturando de manera efectiva la variabilidad observada en planta. En la Figura 5 se muestran ejemplos de fibras sin tensión, comparando los perfiles predichos con los valores reales para evaluar visualmente la precisión de la reconstrucción.

Los resultados muestran que TC-VQ-VAE reconstruye consistentemente las fibras sin tensión a partir de mediciones bajo tensión, superando a las arquitecturas comparadas y preservando la coherencia física de los perfiles.

C. Reconstrucción de la banda sin tensión y evaluación de planitud

El objetivo principal de este trabajo es estimar la planitud de la banda en estado sin tensión a partir de mediciones realizadas bajo tensión. Para ello, se calculan las I-units, que requieren disponer de un mapa de alturas. El modelo propuesto TC-VQ-VAE permite predecir este mapa para las fibras sin tensión, facilitando el cálculo de las I-units. Utilizando estas predicciones junto con las mediciones durante el laminado, es posible reconstruir la banda en tres dimensiones. La Figura 6 muestra la reconstrucción de la banda bajo tensión y sin tensión. Se observa que la altura de la banda sin tensión es mayor que bajo tensión, debido a que el estiramiento reduce la ondulación aparente mientras la banda está bajo carga. Al liberar la tensión, la banda se contrae y aparecen irregularidades, aumentando la amplitud de las fibras. Para apreciar mejor estas diferencias, Figura 7 muestra un segmento de 1 m.

A partir de los mapas de altura es posible calcular las I-units para toda la banda, obteniendo una medida cuantitativa de planitud global (Figura 8). Para analizar la planitud local, se construyen mapas de planitud en 2D (Figura 9) y en 3D (Figura 10), mostrando que la planitud es menor en los bordes debido a ondulaciones locales.

Para aplicaciones industriales, la estimación debe realizarse en tiempo real. La velocidad de los trenes de laminación utilizados varían entre 1 m/s y 10 m/s, lo que deja entre 1 s y 10 s para procesar cada sección de 10 m, que contabiliza 1000 fibras. Los modelos FEM requieren aproximadamente 47 s por fibra, siendo inviables para aplicaciones online. En contraste, TC-VQ-VAE procesa una fibra en 0.003 s con un tamaño de lote de 1 y 0.0009 s con 512 (Fi-



Fig. 5: Ejemplos de fibras sin tensión predichas por TC-VQ-VAE comparadas con las fibras esperadas

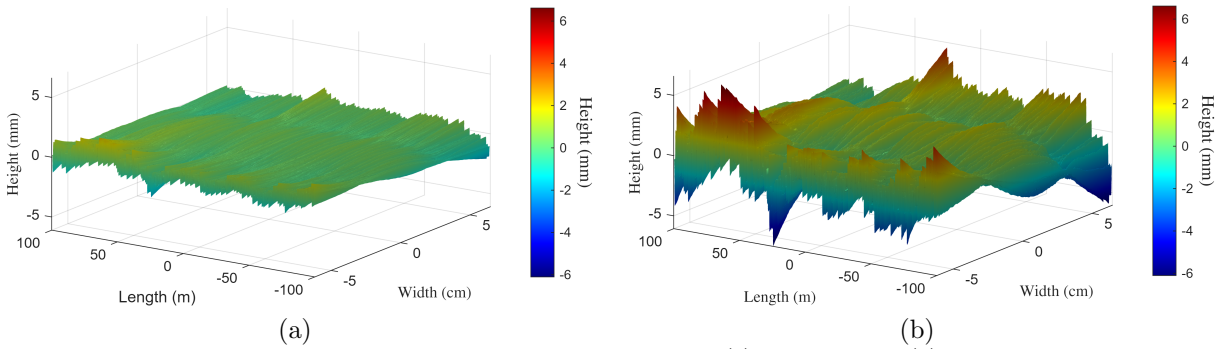
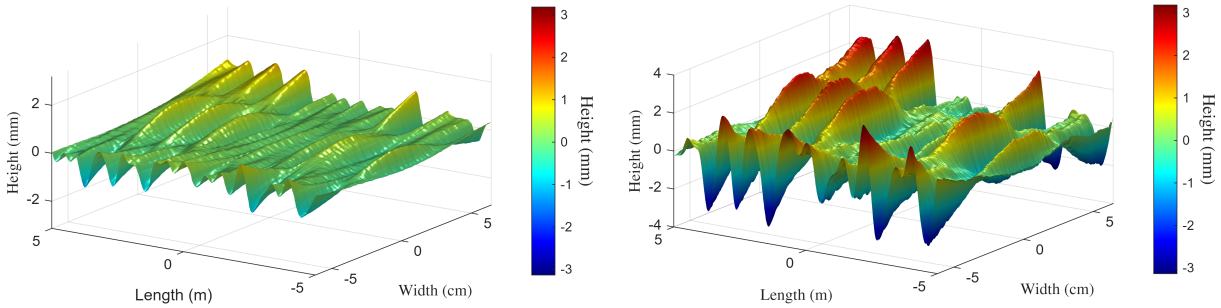


Fig. 6: Reconstrucción de una banda de acero de 200 m: (a) bajo tensión y (b) sin tensión



(a) (b)

Fig. 7: Reconstrucción de un segmento de 1 m: (a) bajo tensión y (b) sin tensión

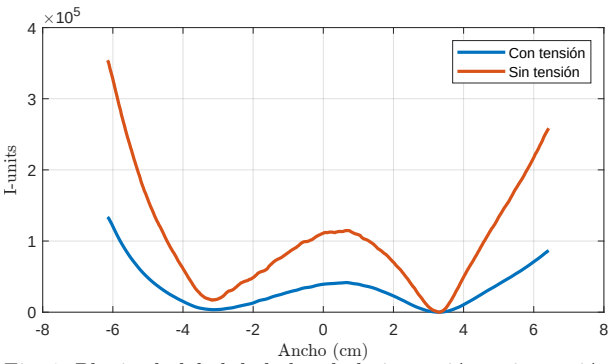
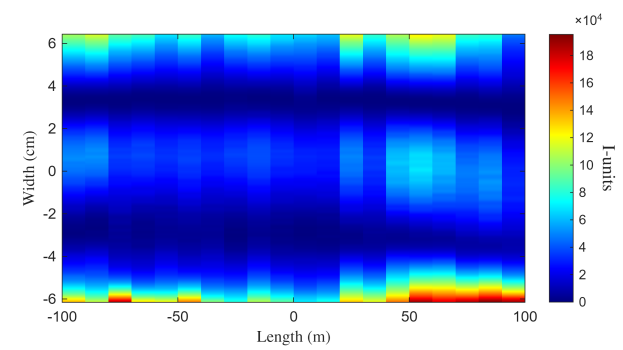
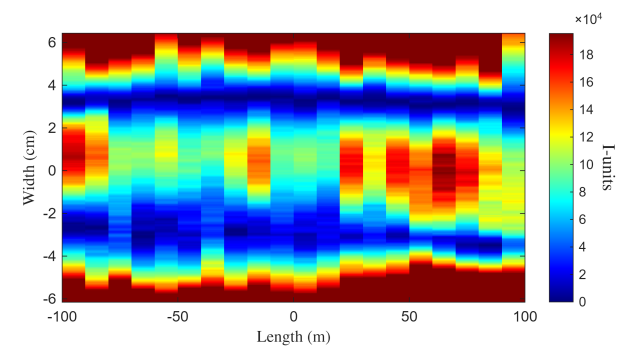


Fig. 8: Planitud global de la banda bajo tensión y sin tensión

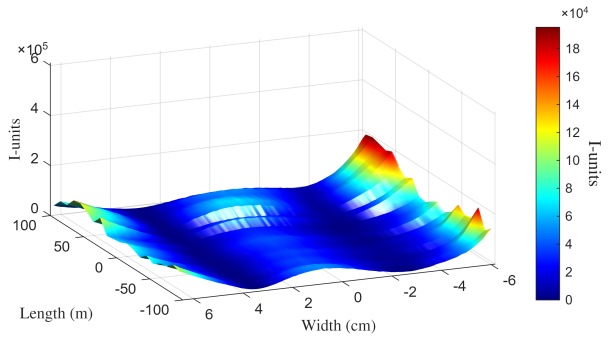


(a)

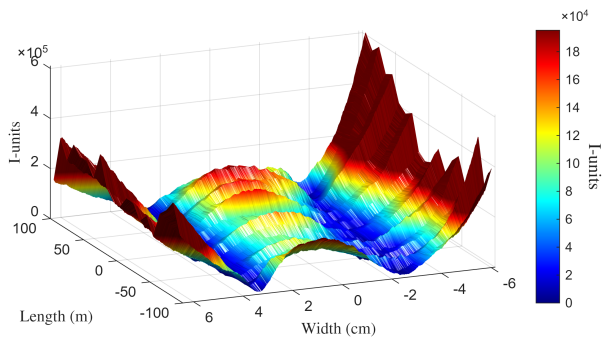


(b)

Fig. 9: Mapas de planitud 2D: (a) bajo tensión y (b) sin tensión



(a)



(b)

Fig. 10: Mapas de planitud 3D: (a) bajo tensión y (b) sin tensión

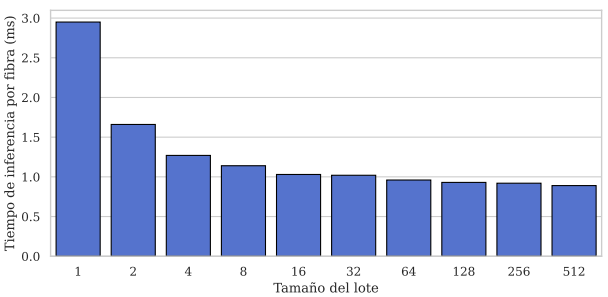


Fig. 11: Tiempo de procesamiento por fibra en función del tamaño de lote

gura 11), cumpliendo ampliamente con los tiempos necesarios para estimación en línea.

En conjunto, los resultados muestran que TC-VQ-VAE permite reconstruir la banda sin tensión y calcular planitud en tiempo real, preservando la coherencia física y capturando tanto la ondulación global como las irregularidades locales.

VII. Conclusiones

En este trabajo se presenta un enfoque novedoso para estimar la planitud de bandas de acero en estado sin tensión mediante un modelo generativo de inteligencia artificial, el Target-Conditioned Vector-Quantized Variational Autoencoder (TC-VQ-VAE). Las principales conclusiones son las siguientes:

- **Predicción precisa de la planitud:** el modelo

TC-VQ-VAE propuesto es capaz de predecir con alta precisión la geometría sin tensión de las fibras de acero a partir de mediciones bajo tensión longitudinal. Esto permite reconstruir de manera confiable el mapa de alturas y calcular las I-units, proporcionando una evaluación cuantitativa de la planitud tanto global como local.

- **Desempeño superior frente a arquitecturas alternativas:** evaluaciones extensas sobre datos reales muestran que TC-VQ-VAE supera consistentemente a arquitecturas convencionales como LSTM, DCGAN, SIG2SIG y DDPM, logrando menores errores (MSE, RMSE, MAE, NMAE) y mayores valores de R^2, preservando la coherencia física en la reconstrucción de fibras.
- **Viabilidad práctica para aplicaciones industriales en tiempo real:** a diferencia de las simulaciones FEM, que son computacionalmente costosas, TC-VQ-VAE procesa cada fibra en aproximadamente 0.0009 s, permitiendo la estimación de planitud en tiempo real incluso a altas velocidades de laminación. Esto demuestra su potencial para implementación en líneas de producción, facilitando la detección temprana de desviaciones y reduciendo desperdicio, tiempo y costos.
- **Capacidad de manejar diversas fuentes de datos:** la estrategia de entrenamiento híbrido permite al modelo generalizar entre mediciones automáticas y manuales de planta, capturando tanto la variabilidad inherente como los patrones periódicos de defectos en la banda.

En resumen, este trabajo introduce un método práctico, preciso y de alta velocidad para predecir la planitud sin tensión de bandas de acero, conectando las mediciones bajo tensión con la geometría final observada tras el procesamiento. Trabajos futuros podrían enfocarse en integrar este enfoque en sistemas de control en línea y extenderlo para predecir la planitud bajo diferentes condiciones de laminación o tipos de material.

Agradecimientos

El presente trabajo ha sido financiado mediante el plan nacional PID2021-124383OB-I00 y el plan regional SEK-25-GRU-GIC-24-003.

Referencias

[1] Siddhartha Ray, *Introduction to Rolling Process*, pp. 1–29, Cambridge University Press, 2016.

[2] Hao Wu, Jie Sun, Xing Lu, Wen Peng, Qinglong Wang, and Dianhua Zhang, "Predicting stress and flatness in hot-rolled strips during run-out table cooling," *Journal of Manufacturing Processes*, vol. 84, pp. 815–831, 2022.

[3] Julio Molleda, Rubén Usamentiaga, and Daniel F. García, "On-Line Flatness Measurement in the Steelmaking Industry," *Sensors*, vol. 13, no. 8, pp. 10245–10272, Aug. 2013, Number: 8 Publisher: Multidisciplinary Digital Publishing Institute.

[4] Qinglong Wang, Jie Sun, Yunjian Hu, Wenqiang Jiang, Xinchun Zhang, and Zhangqi Wang, "Deep learning-based flatness prediction via multivariate industrial data for steel strip during tandem cold rolling," *Expert Systems with Applications*, vol. 237, pp. 121777, 2024.

[5] Wu-quan Yang, Zhi-ting Zhao, Liang-yu Zhu, Xun-yang Gao, and Li Wang, "Strip flatness prediction of cold rolling based on ensemble methods," *Journal of Iron and Steel Research International*, vol. 31, no. 1, pp. 237–251, Jan 2024.

[6] Wen Peng, Jiawei Lei, Chengyan Ding, Chongxiang Yue, Gengsheng Ma, Jie Sun, and Dianhua Zhang, "A novel deep ensemble reinforcement learning based control method for strip flatness in cold rolling steel industry," *Engineering Applications of Artificial Intelligence*, vol. 134, pp. 108695, 2024.

[7] Yu Wang, Changsheng Li, Lianggui Peng, Ruida An, and Xin Jin, "Application of convolutional neural networks for prediction of strip flatness in tandem cold rolling process," *Journal of Manufacturing Processes*, vol. 68, pp. 512–522, 2021.

[8] Wu-quan Yang, Zhi-ting Zhao, Liang-yu Zhu, Xun-yang Gao, and Li Wang, "Strip flatness prediction of cold rolling based on ensemble methods," *Journal of Iron and Steel Research International*, vol. 31, no. 1, pp. 237–251, Jan. 2024.

[9] Wen Peng, Jiawei Lei, Chengyan Ding, Chongxiang Yue, Gengsheng Ma, Jie Sun, and Dianhua Zhang, "A novel deep ensemble reinforcement learning based control method for strip flatness in cold rolling steel industry," *Engineering Applications of Artificial Intelligence*, vol. 134, pp. 108695, Aug. 2024.

[10] S. Abdelkhalek, P. Montmitonnet, N. Legrand, and P. Buessler, "Coupled approach for flatness prediction in cold rolling of thin strip," *International Journal of Mechanical Sciences*, vol. 53, no. 9, pp. 661–675, Sept. 2011.

[11] Qinglong Wang, Jie Sun, Xu Li, Zhenhua Wang, Pengfei Wang, and Dianhua Zhang, "Analysis of lateral metal flow-induced flatness deviations of rolled steel strip: Mathematical modeling and simulation experiments," *Applied Mathematical Modelling*, vol. 77, pp. 289–308, Jan. 2020.

[12] Guomin Han, Hongbo Li, Yujin Liu, Jie Zhang, Xianhao Li, Zhiyuan Hu, Xuechang You, and Zelong Chen, "Flatness jump phenomenon in cold-rolled steel strip with high temperature: Root cause and solution," *Applied Thermal Engineering*, vol. 258, pp. 124563, Jan. 2025.

[13] Lei Cao, Xu Li, and Dianhua Zhang, "Research into the effect of flatness control actuators on vibration stability in cold rolling using the finite element method," *The International Journal of Advanced Manufacturing Technology*, vol. 125, no. 1, pp. 245–266, Mar. 2023.

[14] Hao Wu, Jie Sun, Xing Lu, Wen Peng, Qinglong Wang, and Dianhua Zhang, "Predicting stress and flatness in hot-rolled strips during run-out table cooling," *Journal of Manufacturing Processes*, vol. 84, pp. 815–831, Dec. 2022.

[15] Feng Zhang and Arif Malik, "Stochastic Method to Predict Effects of Roll Grinding Deviations on Sheet Flatness in Cold Rolling," *Journal of Manufacturing Science and Engineering*, vol. 144, no. 061014, Dec. 2021.

[16] Yukio Shigaki, Pierre Montmitonnet, and Jonatas M. Silva, "3D finite element model for roll stack deformation coupled with a Multi-Slab model for strip deformation for flat rolling simulation," *AIP Conference Proceedings*, vol. 1896, no. 1, pp. 190018, Oct. 2017.

[17] Jifei Deng, Seppo Sierla, Jie Sun, and Valeriy Vyatkin, "Physics-informed generative regression for industrial process modeling in steel strip rolling," *Expert Systems with Applications*, vol. 282, pp. 127713, 2025.

[18] Binghua Hu, Yunlong Wang, Yinghua Han, Jinkuan Wang, and Qiang Zhao, "A flatness predict model based on deep belief network for steel rolling process," in *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*, 2019, pp. 235–239.

[19] Yang-huan Xu, Dong-cheng Wang, Hong-min Liu, and Bo-wei Duan, "Intelligent representation method of image flatness for cold rolled strip," *Journal of Iron and Steel Research International*, vol. 31, no. 5, pp. 1177–1195, May 2024.

[20] Marcos Alonso, Daniel Maestro, Alberto Izaguirre, Imanol Andonegui, and Manuel Graña, "Depth data denoising in optical laser based sensors for metal sheet flatness measurement: A deep learning approach," *Sensors*, vol. 21, no. 21, 2021.

[21] Marcos Alonso, Alberto Izaguirre, Imanol Andonegui, and Manuel Graña, "Optical dual laser based sensor denoising for onlinemetal sheet flatness measurement using hermite interpolation," *Sensors*, vol. 20, no. 18, 2020.

[22] Suhua Xiao, Jianyi Wu, Nanying Lai, Ruihao Lin, Mingjuan Qiao, Zhiyong Wang, Wenbin Luo, Youzhi Fu, Peng Liang, Peipei Zhou, and Pujing Liu, "Research on flatness measurement of large-size parts based on 3-d machine vision," *IEEE Transactions on Instrumentation and Measurement*, vol. 72, pp. 1–12, 2023.

[23] Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu, "Neural discrete representation learning," in *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. 2017, vol. 30, Curran Associates, Inc.

[24] Xiao Liu and Hong Xiao, "Theoretical and experimental study on the producible rolling thickness in ultra-thin strip rolling," *Journal of Materials Processing Technology*, vol. 278, pp. 116537, 2020.

[25] Sami Abdelkhalek, "A proposal improvement in flatness measurement in strip rolling," *International Journal of Material Forming*, vol. 12, no. 1, pp. 89–96, Jan 2019.

[26] Rebecca Nakhoul, Pierre Montmitonnet, and Michel Potier-Ferry, "Multi-scale method for modeling thin sheet buckling under residual stresses in the context of strip rolling," *International Journal of Solids and Structures*, vol. 66, pp. 62–76, 2015.

[27] Yue Liu, Zhengwei Yang, Zhenyao Yu, Zitu Liu, Dahui Liu, Hailong Lin, Mingqing Li, Shuchang Ma, Maxim Avdeev, and Siqi Shi, "Generative artificial intelligence and its applications in materials science: Current situation and future perspectives," *Journal of Materiomics*, vol. 9, no. 4, pp. 798–816, 2023.

[28] T. Chai and R. R. Draxler, "Root mean square error (rmse) or mean absolute error (mae)? – arguments against avoiding rmse in the literature," *Geoscientific Model Development*, vol. 7, no. 3, pp. 1247–1250, 2014.

[29] Yue Wang, Yongming Han, Xuan Hu, and Zhiqiang Geng, "Long-term series forecasting for industrial processes based on multiscale hybrid decomposition feature extraction network," *IEEE Transactions on Industrial Informatics*, vol. 21, no. 7, pp. 5401–5410, 2025.

[30] Abhishek V Tatachar, "Comparative assessment of regression models based on model evaluation metrics," *International Research Journal of Engineering and Technology (IRJET)*, vol. 8, no. 09, pp. 2395–0056, 2021.

[31] Dabao Zhang, "A coefficient of determination for generalized linear models," *The American Statistician*, vol. 71, no. 4, pp. 310–316, 2017.

[32] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[33] Alec Radford, Luke Metz, and Soumith Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2016.

[34] SangYeon Kim, Hyunwoo Lee, Jonghee Han, and Joon-Ho Kim, "Sig2sig: Signal translation networks to take the remains of the past," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 3620–3624.

[35] Jonathan Ho, Ajay Jain, and Pieter Abbeel, "Denoising diffusion probabilistic models," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, Eds. 2020, vol. 33, pp. 6840–6851, Curran Associates, Inc.

[36] Rubén Usamentiaga, Daniel F. Garcia, Julio Molleda, Francisco G. Bulnes, and Gabriel Bonet, "Vibrations in steel strips: Effects on flatness measurement and filtering," in *2013 IEEE Industry Applications Society Annual Meeting*, 2013, pp. 1–10.

[37] Jifei Deng, Seppo Sierla, Jie Sun, and Valeriy Vyatkin, "Reinforcement learning for industrial process control: A case study in flatness control in steel industry," *Computers in Industry*, vol. 143, pp. 103748, 2022.

Evaluando las limitaciones de la multiplicación-acumulación en formato posit y las herramientas generadoras de código

Mario Alonso[1], Miguel A. Sacristán[1], Guillermo Botella[1] y Alberto A. del Barrio[1]

Resumen— **El formato posit™ fue introducido por John Gustafson en 2017 como un reemplazo de los formatos IEEE 754 de representación de números en punto flotante. Desde entonces múltiples diseños de unidades funcionales han sido descritos en la literatura. Este artículo analiza detalladamente la operación multiplicación-acumulación (MAC) para datos en formato posit, tomando como referencia uno de los pocos diseños de código abierto presentes en la literatura, generado por medio de la herramienta FloPoCo. Adicionalmente, se han realizado varias síntesis para tecnología de 28nm con y sin restricciones de tiempo, las cuales corroboran las limitaciones de las posit MAC, a saber: un diseño no balanceado, un cuello de botella en la etapa de decodificación y multiplicación, y un timing muy justo en la etapa de acumulación.**

Palabras clave— **Posits, MAC, quire, sumador, síntesis.**

I. Introducción

REPRESENTAR números reales es un desafío, especialmente en computadores o cualquier dispositivo que solo pueda manejar información discreta y finita. Hace décadas, cuestiones como la longitud de la mantisa y el comportamiento de redondeo de las operaciones con números de punto flotante podían diferir entre fabricantes de computadoras e incluso entre modelos del mismo fabricante. Esto causaba problemas de portabilidad y reproducibilidad de los resultados. Sin embargo, la aparición del Estándar IEEE para Aritmética de Punto Flotante (IEEE 754) en 1985 solucionó estas deficiencias al proporcionar definiciones de formatos aritméticos, esquemas de redondeo, operaciones, representación de números especiales y gestión de excepciones [1].

Si bien el estándar IEEE 754 ha evolucionado continuamente, por ejemplo, con la revisión de 2008 [2] o la más reciente de 2019 [3], se han identificado varias deficiencias desde su adopción, como la redundancia de Not a Numbers (NaNs), ceros con signo o inconsistencias en los resultados entre máquinas debido a los diferentes modos de redondeo [4]. En 2017, John Gustafson [5] propuso un nuevo tipo de datos llamado *posit* para abordar algunas de estas deficiencias, el cuál introdujo su último estándar en 2022 [6].

Una de las unidades funcionales más relevantes tanto en el estándar IEEE 754 como con los posits es la multiplicación-acumulación (MAC) [7], [8], [9], [10], muy utilizada en Inteligencia Artificial (IA) y en cualquier aplicación que requiera el uso de operaciones de álgebra lineal. Es en el diseño de la MAC en donde más difieren los posits y los floats, ya que el estándar posit [6] introduce las operaciones fusionadas que postergan el redondeo hasta que la secuencia de multiplicación-acumulaciones haya terminado. Es esta característica una de las que proporciona una ganancia de exactitud notable en el caso de los posits [11], [12], [13]. No obstante, el diseño de las posit MAC es altamente complejo. En este artículo analizaremos el estado del arte de estas unidades y, tomando como referencia la posit MAC presentada en [9], la cual está basada en la herramienta FloPoCo [14], comentaremos las limitaciones existentes, así como las líneas de trabajo futuras para mejorar sus prestaciones.

El resto del artículo se organiza de la siguiente manera: en la Sección II introduciremos una serie de conceptos esenciales para comprender el trabajo; en la Sección III describiremos el estado del arte en materia de unidades posit MAC; en la Sección IV analizaremos una de las pocas unidades posit MAC en abierto para detectar puntos de mejora; en la Sección V daremos datos detallados de síntesis que se alinean con el análisis anterior; y finalmente en la Sección VI resumiremos los puntos más importantes del trabajo y delinearemos las líneas de acción críticas para mejorar el diseño de la posit MAC de cara al futuro.

II. Preliminares

El tipo III universal, *unum type 3*, más conocido como *posit*, se introdujo en 2017 "como reemplazo directo de los números de punto flotante del estándar IEEE 754" [5], [15]. La aritmética de posit está diseñada para realizar cálculos de hardware con una lógica similar a la del formato de punto flotante existente.

Dado un número posit de n bits, éste está compuesto por un bit de signo (s), varios bits para el régimen (r), 2 bits para un exponente sin sesgo (e), y los bits restantes para la fracción normalizada entre 0 y 1 (f), tal y como se muestra en la Figura 1. El régimen es un código unario de k bits idénticos (r_0) terminados en la negación ($\overline{r_0}$), el cual simboliza factor de escala adicional r según la Ecuación (1),

$$r = \begin{cases} -k & \text{if } r_0 = 0 \\ k-1 & \text{if } r_0 = 1 \end{cases} . \quad (1)$$

La aritmética posit destaca por su capacidad de adaptar la estructura de sus bits según la magnitud del número, utilizando un campo de régimen de longitud variable que optimiza el equilibrio entre precisión y rango dinámico. Cuando los valores se acercan

[1]Facultad de Informática, Dpto. de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, e-mail: {marioa25,msacri02,gbotella,abarriog}@ucm.es.



Fig. 1: Codificación genérica del formato posit de n bits.

Tabla I: Ejemplos de decodificación para el formato posit8. Nótese que el binario sigue el código de colores utilizado en la Figura 1 para facilitar su visualización. El valor numérico final, así como el valor de los cuatro campos de un número en formato posit, también se muestran: signo (s), régimen (r), exponente (e), y fracción (f) [15].

Binario	s	r	e	f	Valor
00000001	0	-6	0	0	$\sim 5{,}96046 \times 10^{-8}$
00001011	0	-3	1	0.5	$7{,}324\,218\,75 \times 10^{-4}$
01000000	0	0	0	0	1,0
01101011	0	1	2	0.75	112
01111111	0	6	0	0	$1{,}677\,721\,6 \times 10^{7}$
10000001	1	-6	0	0	$-1{,}677\,721\,6 \times 10^{7}$
10010101	1	-2	1	0.25	-112
11000000	1	0	0	0	$-1{,}0$
11110101	1	2	2	0.5	$-7{,}324\,218\,75 \times 10^{-4}$
11111111	1	6	0	0	$\sim -5{,}96046 \times 10^{-8}$

al entorno de ± 1, el régimen se contrae y cede espacio a la fracción, lo que permite una resolución mucho más fina en ese intervalo conocido como *golden zone* [16]. Por el contrario, para magnitudes extremas, el régimen se expande para ampliar el rango representable, llegando incluso a absorber los bits destinados al exponente o la fracción si la escala lo requiere. Esta flexibilidad ocasiona que el exponente y la fracción no tengan un tamaño fijo.

El valor real p de un número posit interpretado como complemento a 2 se define mediante la Ecuación (2). Como puede observarse, el campo fracción tiene un 1 implícito a la izquierda del punto decimal si el número es positivo o un -2 si es negativo [17].

$$p = ((1 - 3s) + f) \times 2^{(1-2s)\times(4r+e+s)}. \qquad (2)$$

En la aritmética posit, el valor NaR se representa como el entero más negativo en complemento a dos, lo que permite que todos los números se ordenen igual que los enteros. Esta disposición permite realizar comparaciones utilizando los mismos circuitos destinados a los enteros tradicionales, eliminando la necesidad de componentes adicionales para la gestión de números reales. Varios ejemplos de codificación de números posit se muestran en la Tabla I.

Por otro lado, este sistema destaca por el uso de operaciones fusionadas y el *quire*, un registro de alta precisión que acumula resultados intermedios en formato de punto fijo. Al evitar los redondeos en cada operación intermedia, el *quire* garantiza una exactitud superior y una mayor fidelidad en los resultados finales en comparación con los estándares de coma flotante convencionales [13], [18], [19].

III. Estado del Arte

En esta sección describiremos los diseños más relevantes de unidades de multiplicación-acumulación (MAC) para datos bajo el formato posit.

En [20], se presentó Deep Positron, un acelerador de baja precisión ($n \leq 8$) que incluye una MAC exacta. Aunque a nivel global los autores determinaron que la eficiencia energética de los posits es competitiva frente al formato equivalente en punto flotante, también señalaron una mayor cantidad de recursos hardware. En una línea similar, el trabajo en [21] introdujo un generador de unidades MAC segmentadas en 5 etapas con el objetivo de alcanzar frecuencias de reloj altas. Si bien esta implementación también superó en área a la unidad equivalente en punto flotante, los autores concluyeron que la mayor exactitud y rango dinámico de los posits compensan este coste adicional. Por otro lado, el artículo en [23] propuso una unidad capaz de ajustar dinámicamente el tamaño del exponente en tiempo de ejecución, aunque penalizando en área y frecuencia con respecto a trabajos anteriores. Es importante destacar que, a pesar de sus diferencias, ninguna de estas tres propuestas emplea el acumulador quire para operaciones fusionadas, limitándose a estructuras de tres operandos.

En el artículo [22] se presentó MArTo, una biblioteca de C++ compatible con los entornos modernos de Síntesis de Alto Nivel (SAN). En dicha biblioteca de código abierto, además de suma, resta o multiplicación, se incluye una implementación de la MAC parametrizada y con soporte para el quire. Los autores de [22] concluyen que el coste y rendimiento de un quire para posits de 32 bits es comparable al de un acumulador de Kulisch [24]para números de punto flotante de 32 bits. No obstante, el quire es algo más pequeño que el acumulador de Kulisch equivalente (512 vs 589 bits). En el trabajo publicado en [9] se presenta una MAC parametrizada con soporte para quire, consiguiendo reducciones de consumo con respecto al estado del arte en MACs para posits [9]. Dichas unidades posteriormente se incluyeron en un flujo de diseño SAN a través de la herramienta Bambu [18]. Los autores en [10] introdujeron una unidad funcional dedicada a la realización de productos escalares, incluyendo soporte para quire.

Las unidades anteriormente citadas se han presentado en la literatura de manera aislada. Adicionalmente, el estado del arte también ha mostrado en los últimos años unidades MAC integradas dentro de procesadores con soporte para instrucciones con datos en formato posit, como por ejemplo en CLARINET [12], en PERCIVAL tanto para datos con precisión de 32 como 64 bits [11], [13], en el procesador comercial Tunga [25] e incluso en aceleradores como PHEE [26], que soporta posits de 16 bits, MINOTAUR, un array sistólico 2D para posits de 8 bits, o TALU-V [27], un coprocesador vectorial con capacidades de transprecisión.

No obstante, la desventaja principal de todos estos trabajos es que las MACs para posits, y unidades funcionales en general, son significativamente más grandes y lentos que sus equivalentes (misma precisión) de punto flotante. A cambio, los trabajos de la literatura coinciden en una ganancia en términos de exactitud.

Tabla II: Diseños del estado del arte que soportan la multiplicación-acumulación con datos en formato posit

	Parametrizada	Pipeline	Quire	Implementación	Tecnología	Código abierto
Carmichael et al. [20] (Deep Positron)	✓*	✓	✓	Verilog	Virtex-7	✗
Zhang et al. [21]	✓	✓	✗	Verilog	ASIC	✗
Uguen et al. [22] (MArTo)	✓	✓	✓	Vitis HLS	Kintex-7	✓
Neves et al. [23]	✓	✓	✓	–	Virtex-7 & ASIC	✗
Murillo et al. [9]	✓	✓	✓	VHDL†	ASIC	✓
Li et al. [10]	✓	✓	✓	SystemVerilog	ASIC	✓

* Parcialmente soportado. † Generado con FloPoCo [14].

IV. Análisis de la posit MAC

En esta sección presentaremos un análisis detallado de uno de los escasos diseños de posit MAC que se encuentran en abierto, como lo es el del artículo [9][1].

Tal y como puede observarse en la Figura 2, la posit MAC recibe dos datos en formato posit (A yB) y un tercero en punto fijo (C), que es la salida del registro acumulador quire. El motivo principal es que esta MAC posteriormente está integrada en la PAU de un procesador [11], [13]. Ha de notarse también que la posit MAC para posit32 está organizada en 3 etapas de pipeline, mientras que la de posit64 necesita 4 etapas, aunque a nivel global se distinguen claramente 3 grandes bloques:

- Decodificación de las entradas A y B, multiplicación de fracciones y escalado del resultado.
- Alineación del resultado anterior por medio de un desplazador de barril.
- Acumulación en el registro quire.

Como puede observarse, la principal diferencia es el número de etapas necesarias para realizar el desplazamiento del producto normalizado de las fracciones de A y B. El motivo principal es que este diseño fue generado a través de la herramienta FloPoCo [14], lo cual facilita su parametrización y segmentación, pero todas las decisiones de diseño están orientadas a optimizar su mapeo sobre una FPGA. En concreto, en la generación del diseño presentado en [9] se utilizó la configuración por defecto, la cual optimiza para la familia Kintex 7, en la cual es mucho más sencillo mapear un multiplicador 61×61, como el empleado en la Figura 2b, que un desplazador de barril de 1024 bits. Como veremos en la Sección V, al realizar síntesis ASIC esto causa que las diversas etapas queden desbalanceadas.

Como consecuencia de este desequilibrio entre etapas, la primera y la última etapas son las que realizan más trabajo y tienen mayor complejidad.

- La primera etapa tiene su cuello de botella en el multiplicador, 29×29 para posit32 y 61×61 para posit64. Adicionalmente, hay que tener en cuenta que el proceso de decodificación de los posits es ligeramente más complejo que el de los datos en punto flotante, ya que hay campos cuya longitud varía dinámicamente (el régimen y la fracción) [5], [**?**].
- La última etapa depende de un sumador muy grande: 512 bits para posit32 y 1024 bits para posit64. Además de introducir un retardo alto, implica una penalización en área y consumo elevadas, ya que los sumadores rápidos escalan su complejidad de manera no lineal. Por ejemplo, el área de un sumador Kogge-Stone [28], el sumador prefijo má rápido, tiene una complejidad $O(nlog(n))$.

[1] `https://github.com/artecs-group/Flo-Posit`

V. Resultados experimentales

En esta sección presentaremos varios resultados tras realizar la síntesis de la unidad posit MAC [9] con precisión de 32 y 64 bits. Para ello se ha utilizado la herramienta Design Compiler de Synopsys conjuntamente con una librería de TSMC de 28nm.

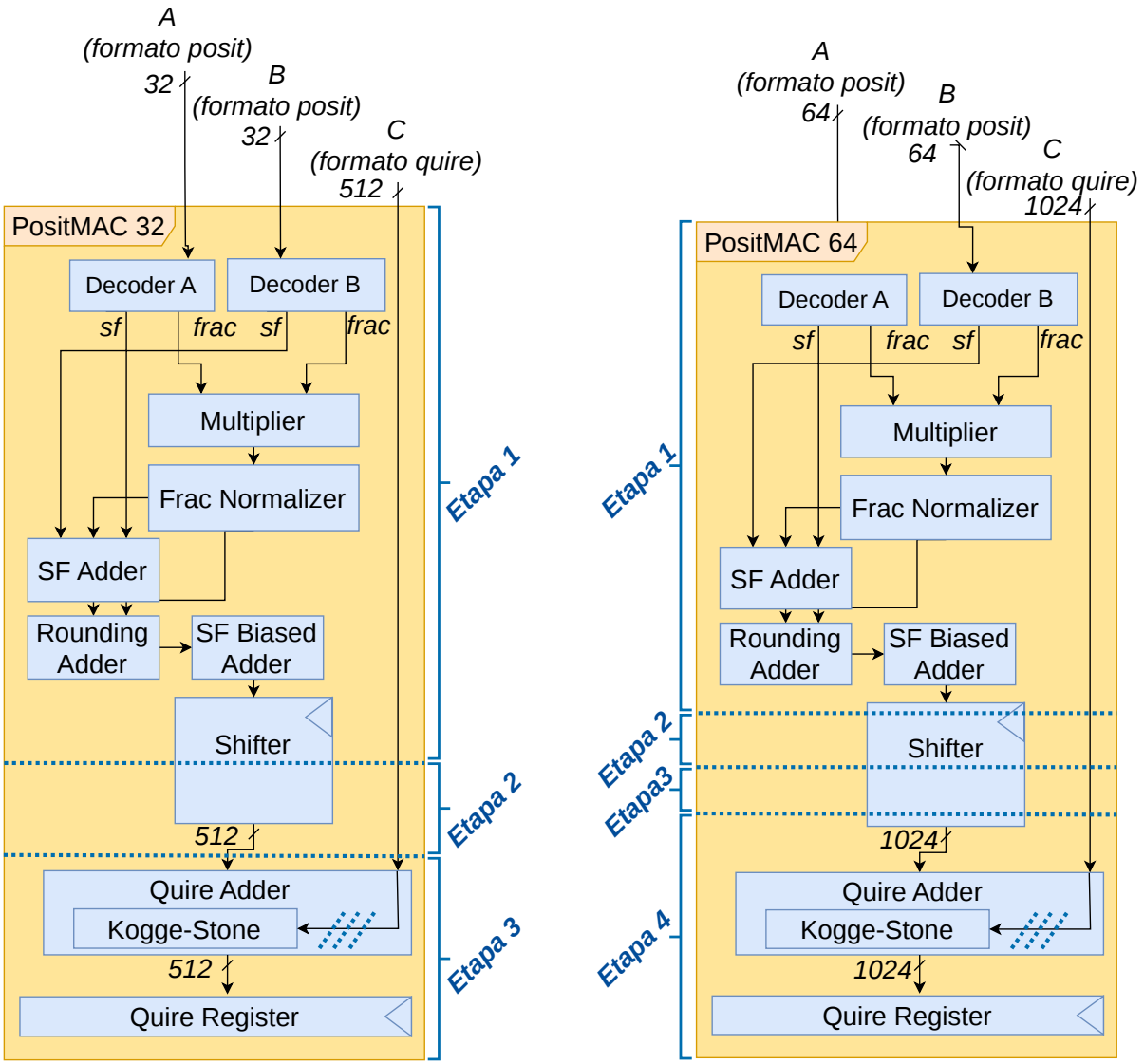
A. Resultados de síntesis

Las Figuras 3 y 4 muestran los resultados de síntesis sin restricciones temporales y con una restricción de tiempo de ciclo de 1 ns ($f = 1GHz$) respectivamente, asumiendo una incertidumbre del 10 % y el parámetro `switching_activity` = 0,1.

En términos de retardo, las Figuras 3a y 4a muestran valores diferentes, aunque tendencias similares. Claramente, el principal cuello de botella se encuentra en la primera etapa tanto con 32 como con 64 bits. Incluso aplicando una restricción de tiempo de 1 ns, el camino crítico de esta etapa supera con creces dicha restricción. Las etapas en las que se divide el desplazador (etapa 2 para 32 bits, etapas 2 y 3 para 64 bits) no tienen problemas para cumplir con la restricción temporal; pero la última etapa se encuentra al límite del tiempo de ciclo de 1 ns, con lo cual es esperable que al aumentar la frecuencia la última etapa tuviera problemas para cumplir con la nueva restricción temporal.

A nivel de área y consumo, tanto de potencia, como de energía, pueden observarse tendencias similares en las gráficas de las Figuras 3b, 3c y 3d con respecto a las Figuras 4b, 4c y 4d, en las que se analizan por componentes de la posit MAC los distintos parámetros de la síntesis. Por un lado hay un incremento generalizado al pasar de 32 a 64 bits, y por otro lado puede observarse cómo hay tres bloques hardware dominantes: el multiplicador de las fracciones, el desplazador, y el sumador (en conjunto con el quire).

Adicionalmente al comparar las Figuras 3 y 4 puede verse cómo, de manera general, la inclusión de la restricción temporal por un lado mejora los retardos



(a) Tipo de datos posit32, quire de 512 bits. (b) Tipo de datos posit64, quire de 1024 bits.

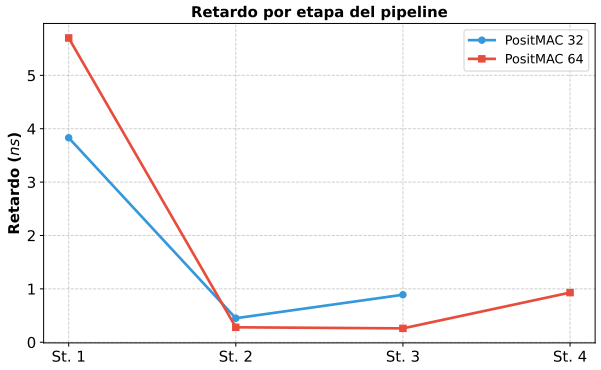
Fig. 2: Arquitectura de la posit MAC presentada en [9].

de cada etapa; mientras que por otro lado los componentes ocupan y consumen más, siendo más notable el incremento de área que de consumo.

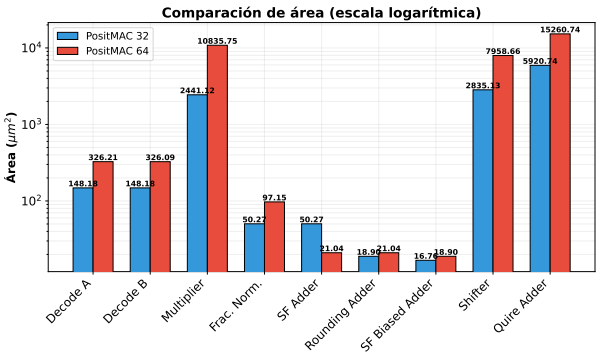
B. Limitaciones encontradas

A tenor de los anteriores resultados podemos observar varias limitaciones en el diseño del estado del arte evaluado:

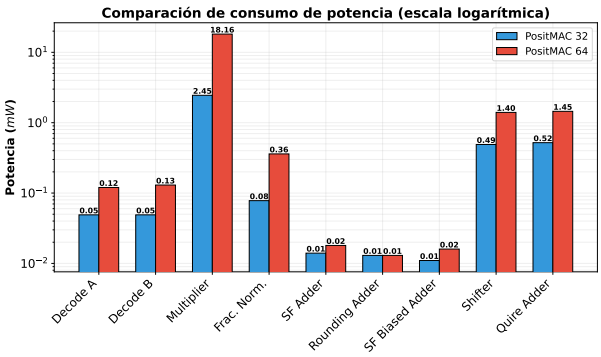
- Es cierto que FloPoCo facilita la generación de código parametrizado, con y sin pipeline, el mapeo del código VHDL no es bueno en síntesis ASIC. Como ha podido verse, la inserción automática de registros de pipe genera etapas desbalanceadas. De cara a futuro sería interesante desacoplar el diseño de la unidad con respecto a la herramienta FloPoCo para evitar estos problemas.
- La primera etapa necesita dividirse. Incluyendo la restricción temporal de $f = 1GHz$ se ha disminuido el retardo de la primera etapa considerablemente, pero aun así se incumple el tiempo de ciclo de 1 ns. El número de etapas puede variar dependiendo de la frecuencia objetivo, pero sí que es conveniente desacoplar el multiplicador de los decodificadores y normalizadores.
- El multiplicador de la posit MAC está generado a través de bloques DSP y componentes pensados para su mapeo en FPGA. Para optimizar los resultados de síntesis ASIC sería conveniente sustituirlos por diseños más eficientes, como los basados en recodificación Booth [29].
- El desplazador es costoso a nivel de recursos hardware, pero no tiene un retardo excesivamente elevado. Es decir, podría combinarse con algunos de los sumadores previos, que no son muy grandes.
- El sumador de la etapa de quire es el segundo elemento más demandante de la posit MAC a nivel de recursos hardware y de retardo. Pese a que con la restricción de $f = 1GHz$ se consigue un retardo por debajo de 1 ns en la última etapa, es bastante probable que con frecuencias más elevadas, i.e. tiempos de ciclos más bajos,



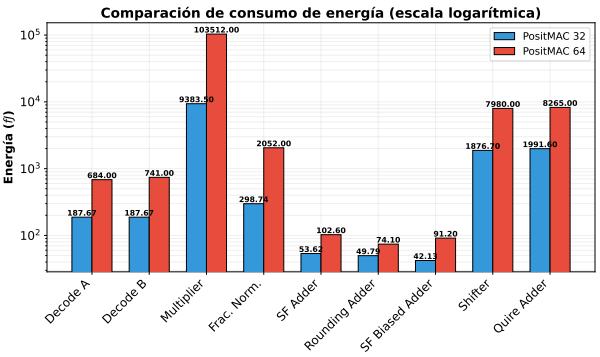
(a) Retardo por etapa del pipeline (ns).



(b) Área de cada componente (μm^2).

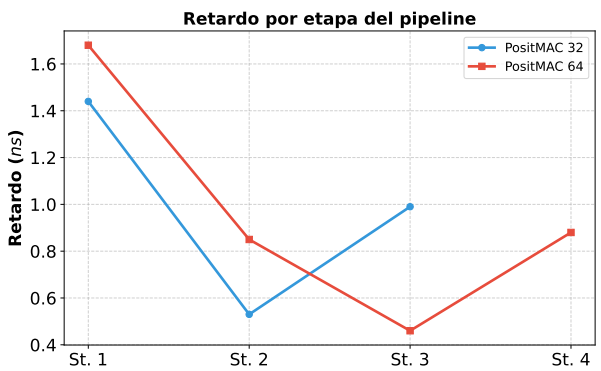


(c) Potencia consumida por cada componente (mW).

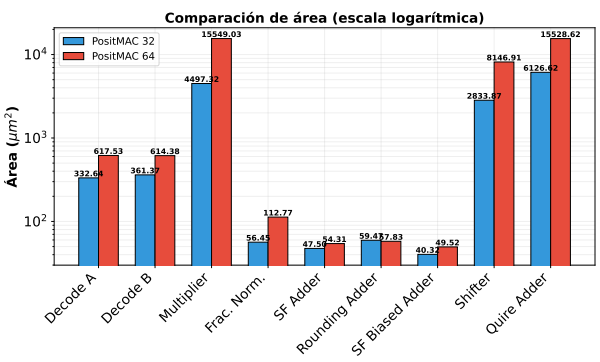


(d) Energía consumida por cada componente (pJ).

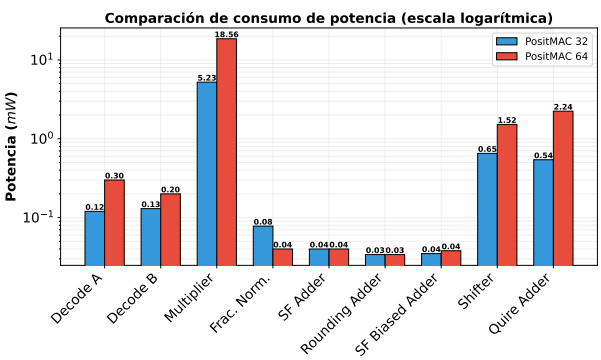
Fig. 3: Resultados de síntesis sin restricciones de tiempo.



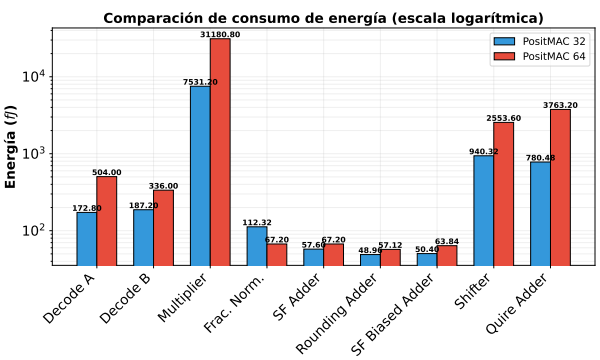
(a) Retardo por etapa del pipeline (ns).



(b) Área de cada componente (μm^2).



(c) Potencia consumida por cada componente (mW).



(d) Energía consumida por cada componente (pJ).

Fig. 4: Resultados de síntesis con una restricción de tiempo de ciclo de $1ns$ ($f = 1GHz$).

esto no pueda cumplirse, o que para cumplirse la penalización en área y consumo sea muy alta. Por tanto, sería recomendable dividir esta etapa también.

VI. Conclusiones

En este artículo hemos presentado un análisis pormenorizado de uno de los diseños de referencia en el estado del arte de las unidades posit MAC. Una de las bondades del diseño en [9] es la automatización que proporciona la herramienta FloPoCo, pero esto causa que las etapas del pipeline queden desbalanceadas. Por otro lado, el multiplicador y el sumador son componentes hardware muy demandantes a nivel hardware dentro de la posit MAC, por lo que sería aconsejable desacoplarlas del camino crítico e incluso investigar diseños más eficientes para poder conseguir frecuencias $\geq 1GHz$.

Como trabajos futuros se plantean la modificación del número de etapas y/o la restructuración de las mismas para equilibrar el retardo entre ellas, así como el uso de implementaciones más eficientes en los componentes con mayor camino crítico, como el multiplicador de fracciones y el sumador de la etapa final.

Agradecimientos

El presente trabajo ha sido apoyado por los proyectos PID2021-123041OB-I00 y PID2024-158203OB-I00 financiados por MCIN/AEI/ 10.13039/501100011033 y por ERDF/EU.

Referencias

[1] IEEE Computer Society, "IEEE Standard for Binary Floating-Point Arithmetic," *ANSI/IEEE Std 754-1985*, pp. 1–20, 1985.

[2] IEEE Computer Society, "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008 (Revision of IEEE 754-1985)*, vol. 2008, pp. 1–70, 2008.

[3] IEEE Computer Society, "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

[4] David Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys (CSUR)*, vol. 23, no. 1, pp. 5–48, 1991.

[5] John L. Gustafson and Isaac Yonemoto, "Beating Floating Point at its Own Game: Posit Arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, pp. 71–86, jun 2017.

[6] Posit Working Group, "Standard for Posit Arithmetic (2022)," `https://posithub.org/docs/posit_standard-2.pdf`, Feb. 2022.

[7] Alberto A. Del Barrio, Nader Bagherzadeh, and Román Hermida, "Ultra-low-power adder stage design for exascale floating point units," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 3s, Mar. 2014.

[8] Stefan Mach, Fabian Schuiki, Florian Zaruba, and Luca Benini, "Fpnew: An open-source multiformat floating-point unit architecture for energy-proportional transprecision computing," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 29, no. 4, pp. 774–787, 2021.

[9] Raúl Murillo, David Mallasén, Alberto A. Del Barrio, and Guillermo Botella, "Energy-Efficient MAC Units for Fused Posit Arithmetic," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, Oct. 2021, pp. 138–145.

[10] Qiong Li, Chao Fang, and Zhongfeng Wang, "PDPU: An Open-Source Posit Dot-Product Unit for Deep Learning Applications," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2023, pp. 1–5.

[11] David Mallasén, Raul Murillo, Alberto A. Del Barrio, Guillermo Botella, Luis Piñuel, and Manuel Prieto-Matias, "PERCIVAL: Open-Source Posit RISC-V Core With Quire Capability," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 3, pp. 1241–1252, July 2022.

[12] Niraj N. Sharma, Riya Jain, Mohana Madhumita Pokkuluri, Sachin B. Patkar, Rainer Leupers, Rishiyur S. Nikhil, and Farhad Merchant, "CLARINET: A quire-enabled RISC-V-based framework for posit arithmetic empiricism," *Journal of Systems Architecture*, vol. 135, pp. 102801, Feb. 2023.

[13] David Mallasén, Alberto A. Del Barrio, and Manuel Prieto-Matias, "Big-PERCIVAL: Exploring the Native Use of 64-Bit Posit Arithmetic in Scientific Computing," *IEEE Transactions on Computers*, vol. 73, no. 6, pp. 1472–1485, June 2024.

[14] Florent De Dinechin and Bogdan Pasca, "Designing Custom Arithmetic Data Paths with FloPoCo," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, July 2011.

[15] David Mallasén, Raul Murillo, Guillermo Botella, and Alberto Antonio Del Barrio, "Navigating posit arithmetic: A comprehensive survey of principles, hardware, and applications," vol. 58, no. 5, 2025.

[16] Florent de Dinechin, Luc Forget, Jean-Michel Muller, and Yohann Uguen, "Posits: the good, the bad and the ugly," in *2019 Conference for Next Generation Arithmetic (CoNGA)*, New York, NY, USA, mar 2019, pp. 1–10, ACM.

[17] Andre Guntoro, Cecilia De La Parra, Farhad Merchant, Florent De Dinechin, John L. Gustafson, Martin Langhammer, Rainer Leupers, and Sangeeth Nambiar, "Next Generation Arithmetic for Edge Computing," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. mar 2020, pp. 1357–1365, IEEE.

[18] Raul Murillo, Alberto A. Del Barrio, Guillermo Botella, and Christian Pilato, "Generating Posit-Based Accelerators With High-Level Synthesis," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 10, pp. 4040–4052, Oct. 2023.

[19] Luis Crespo, Pedro Tomas, Nuno Roma, and Nuno Neves, "Unified Posit/IEEE-754 Vector MAC Unit for Transprecision Computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 5, pp. 2478–2482, May 2022.

[20] Zachariah Carmichael, Hamed F. Langroudi, Char Khazanov, Jeffrey Lillie, John L. Gustafson, and Dhireesha Kudithipudi, "Deep Positron: A Deep Neural Network Using the Posit Number System," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. mar 2019, pp. 1421–1426, IEEE.

[21] Hao Zhang, Jiongrui He, and Seok-Bum Ko, "Efficient Posit Multiply-Accumulate Unit Generator for Deep Learning Applications," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. may 2019, vol. 2019-May, pp. 1–5, IEEE.

[22] Yohann Uguen, Luc Forget, and Florent de Dinechin, "Evaluating the Hardware Cost of the Posit Number System," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. sep 2019, pp. 106–113, IEEE.

[23] Nuno Neves, Pedro Tomas, and Nuno Roma, "Dynamic Fused Multiply-Accumulate Posit Unit with Variable Exponent Size for Low-Precision DSP Applications," *2020 IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 1–6, 2020.

[24] Ulrich Kulisch, "Computer Arithmetic and Validity: Theory, Implementation, and Applications," in *Computer Arithmetic and Validity*. De Gruyter, Aug. 2008.

[25] Calligo Technologies, "Calligo technologies' products," `https://calligotech.com/`, 2024.

[26] David Mallasén, Pasquale Davide Schiavone, Alberto A. Del Barrio, Manuel Prieto-Matias, and David Atienza, "Increasing the energy-efficiency of wearables using low-precision posit arithmetic with phee," 2025.

[27] Ayushi Dube, Gian Singh, and Sarma Vrudhula, "A compact, low power transprecision alu for smart edge devices," in *2025 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2025, pp. 1–8.

[28] Peter M. Kogge and Harold S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786–793, 1973.

[29] Andrew D. Booth, "A signed binary multiplication technique," *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 01 1951.

Estudio comparativo de arquitecturas Intel y AMD con aplicaciones de grafo

Jaime de Lacy Orellana[1], Lucía Pons[1] y Julio Sahuquillo[1]

Resumen— **Las aplicaciones de grafo presentan un comportamiento complejo en arquitecturas actuales, ya que su escalabilidad y eficiencia están fuertemente condicionadas por la irregularidad de los accesos a memoria, la sincronización entre hilos y la capacidad de la arquitectura para explotar el paralelismo. Este trabajo presenta la caracterización del *benchmark suite* GAP, sobre dos plataformas representativas de servidores actuales, Intel y AMD. Se han seleccionado nueve combinaciones de algoritmo y grafo, que se han ejecutado en un único zócalo, variando el número de *cores* y el modo de ejecución ST y SMT. Para el análisis se han considerado métricas de tiempo de ejecución, *speedup* y el método de análisis de microarquitectura Intel Top-Down. Los resultados han mostrado que la escalabilidad y el beneficio del SMT dependen significativamente del algoritmo, grafo y arquitectura. Los algoritmos que más se han aprovechado del paralelismo y del SMT son Triangle Counting y Betweenness Centrality, mientras que PageRank muestra una mejora más limitada. Se ha observado que las aplicaciones con mayor peso de *Frontend* y *Bad Speculation* se benefician más del SMT, en comparación con las que están limitadas por memoria. La comparación entre plataformas concluye que no existe una arquitectura claramente superior para todas las aplicaciones, sino que el comportamiento depende de la interacción entre la carga y las características microarquitectónicas del sistema.**

Palabras clave— **Aplicaciones de grafos, Intel, AMD, SMT, Microarquitectura, Escalabilidad, GAP Benchmarks Suite**

I. Introducción

LAS aplicaciones de grafos han adquirido una importancia creciente en los sistemas de computación actuales, ya que numerosos campos de interés en los últimos años modelan de forma natural su información mediante nodos y aristas [1]. Este tipo de representación funciona muy bien para describir relaciones complejas y de gran escala, como es el caso de las redes sociales, sistemas de recomendación, análisis de la Web, bases de conocimiento o distintos problemas de ciberseguridad [2] [3] [4] [5] [6]. Como consecuencia, el procesamiento eficiente de grafos se ha convertido en una tarea relevante tanto en entornos de análisis de datos como en sistemas de propósito general, donde algoritmos como PageRank (PR), Betweenness Centrality (BC) o Triangle Counting (TC) permiten extraer información estructural, identificar nodos relevantes y caracterizar patrones de conectividad en conjuntos de datos cada vez más grandes.

La caracterización de este tipo de cargas en los procesadores actuales no resulta trivial. A diferencia de otras cargas con patrones de ejecución más regulares, las aplicaciones de grafos presentan un paralelismo irregular, que depende tanto del algoritmo considerado como de la estructura del grafo en sí [7]. Como consecuencia, la escalabilidad de los sistemas no es uniforme al incrementar el número de *cores*, por lo que diferentes combinaciones de algoritmo y grafo pueden mostrar comportamientos muy diferentes [8]. Además, este tipo de aplicaciones mantiene una interacción intensa con la jerarquía de memoria, debido a sus accesos dispersos y a la escasa reutilización de datos en caché [9]. Por lo tanto, las prestaciones finales no dependen únicamente de la capacidad de cómputo disponible, sino también de cómo la arquitectura es capaz de tolerar la presión sobre las cachés, la memoria principal y otros recursos compartidos del sistema.

El objetivo de este trabajo es caracterizar el comportamiento de las aplicaciones de la *benchmark suite* GAP [10] en dos plataformas, Intel Xeon Gold 6438Y+ y AMD EPYC 9224, representativas de los servidores de cómputo actuales, y analizar hasta qué punto las diferencias observadas en las prestaciones pueden relacionarse con las características microarquitectónicas y de organización del sistema de ambas arquitecturas. Este escenario es de especial interés comparativo, ya que permite estudiar el comportamiento sobre una plataforma Intel de generación reciente y una plataforma AMD, lo que, hasta donde alcanza nuestro conocimiento, no se ha estudiado anteriormente. Para ello, en este trabajo se estudian distintas combinaciones de algoritmo y grafo, variando el número de *cores* y el modo de ejecución, en configuraciones *Single Thread* (ST) y *Simultaneous Multi-Threading* (SMT). Para realizar el análisis, se consideran métricas de prestaciones como el tiempo de ejecución y el *speedup* así como el método de análisis de microarquitectura de Intel Top-Down [11], con el objetivo de evaluar la escalabilidad de las aplicaciones, el efecto del SMT y la influencia que ejercen la jerarquía de memoria y los recursos compartidos en función de la plataforma.

II. Trabajos relacionados

En los últimos años, el estudio de aplicaciones de grafos ha puesto de manifiesto que las prestaciones obtenidas no dependen solo del algoritmo en sí, sino también de los recursos compartidos del procesador y de la jerarquía de memoria. En esta línea, [8] analiza aplicaciones de la *suite* de *benchmarks* GAP y muestra que la escalabilidad al aumentar el número de *cores* no es uniforme, sino que depende del cuello de botella dominante en cada caso. Concretamente, el trabajo distingue distintos comportamientos de es-

[1]Dpto. de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mails: jdelac6@upv.es, lupones@disca.upv.es, jsahuqui@disca.upv.es

calabilidad en función de la combinación algoritmo-grafo en cuestión, y los relaciona con la presión ejercida sobre la caché de último nivel (LLC) y la memoria principal, lo que permite interpretar por qué algunas aplicaciones son capaces de aprovechar mejor el paralelismo disponible que otras.

Por otro lado, el subsistema de memoria se muestra como el principal factor limitante de las prestaciones en aplicaciones de grafo. Debido al gran tamaño de los conjuntos de datos y a sus accesos altamente irregulares, las cargas de grafos muestran una localidad reducida y una fuerte dependencia de DRAM. En este sentido, [9] demuestra que las jerarquías de caché convencionales resultan poco eficaces para este tipo de aplicaciones, ya que el 78,6 % de los accesos que fallan en la caché L1D también fallan en la caché L2 y la LLC, terminando en la DRAM. El artículo utiliza esta observación para motivar mecanismos orientados a reducir la latencia y la polución en la caché.

También se ha mostrado cómo el comportamiento de las aplicaciones varía con la plataforma de ejecución. [12] compara la utilización de CPU y memoria sobre dos plataformas ARM. El trabajo muestra que, en aplicaciones irregulares como las del *benchmark suite* GAP, tanto la estructura del grafo como la disponibilidad de hilos y la presencia de una LLC de mayor capacidad influyen de forma relevante en las prestaciones observadas.

III. Entorno experimental

Sistemas. El trabajo se ha realizado en dos plataformas experimentales, un Intel Xeon Gold 6438Y+ y un AMD EPYC 9224. Las principales características de las mismas se detallan en la Tabla I. Como se puede ver, ambas arquitecturas presentan diferencias significativas, tanto en capacidad de cómputo como en organización de la jerarquía de memoria. En primer lugar, la plataforma Intel dispone de un mayor número de núcleos por zócalo, con 32 núcleos SMT frente a los 24 de la plataforma AMD, en ambos casos, 2 hilos por núcleo. Además, Intel incorpora una mayor capacidad de caché privada por núcleo, con 48 KB de L1D y 2 MB de L2, mientras que AMD dispone de 32 KB de L1D y 1 MB de L2 por núcleo. Por otro lado, la organización de la caché de último nivel es más distribuida en AMD. Mientras que Intel presenta 60 MB de LLC compartida a nivel de zócalo, AMD dispone de 64 MB de LLC total, organizados en cuatro bloques de 16 MB, compartidos por grupos de 6 núcleos. Esto no solo es importante porque ofrece más espacio disponible, sino también porque cambia la forma en que dicho espacio se comparte entre los núcleos. De igual manera, ambas plataformas difieren en la capacidad de memoria principal instalada: Intel cuenta con 256 GiB de memoria ECC distribuidos en 16 módulos DIMM de 16 GiB, mientras que AMD dispone de 64 GiB ECC implementados mediante 4 módulos DIMM de 16 GiB.

Tabla I: Características de las plataformas empleadas en el entorno experimental.

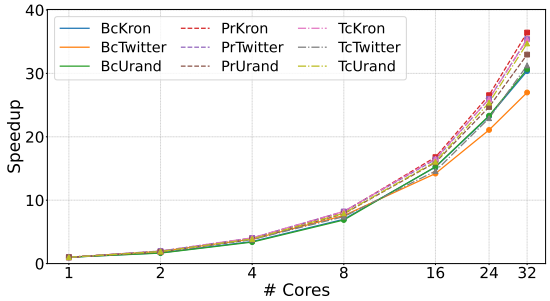
Componente	**Intel**	**AMD**
Procesador	Intel Xeon Gold 6438Y+	AMD EPYC 9224
Zócalos	2	1
Núcleos por zócalo	32	24
Hilos por *core*	2	2
L1D por *core*	48 KB	32 KB
L1I por *core*	32 KB	32 KB
L2 por *core*	2 MB	1 MB
L3 total	60 MB	64 MB
Memoria total	256 GiB	64 GiB
Configuración de memoria	16 × 16 GiB DIMMs	4 × 16 GiB DIMMs

Cargas de trabajo. Para llevar a cabo los experimentos, en este trabajo se han utilizado nueve aplicaciones del conjunto de *benchmark suite* GAP, definidas como la combinación de tres algoritmos y tres grafos representativos. En particular, se han evaluado las aplicaciones tcKron, tcTwitter, tcUrand, correspondientes a TC; bcKron, bcTwitter, bcUrand, asociadas a BC; y prKron, prTwitter y prUrand, pertenecientes a PR. Se ha decidido evaluar estas ya que, dentro del conjunto de combinaciones de algoritmos y grafos disponibles en el benchmark, son las que han mostrado un *speedup* mayor que 25 con 32 *cores* SMT en la plataforma Intel. Esto permite centrar el análisis en configuraciones donde el paralelismo resulta más representativo y donde las diferencias de comportamiento entre plataformas pueden observarse con mayor claridad.

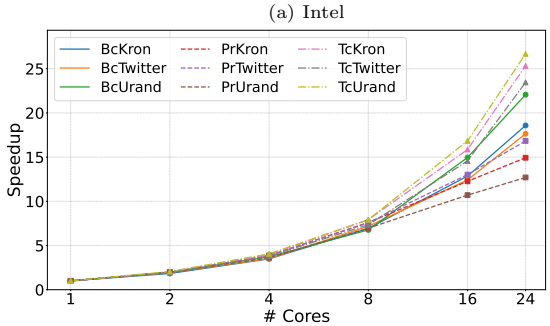
Metodología. Los experimentos se han realizado sobre un único zócalo en ambas plataformas, variando el número de núcleos y el modo de ejecución, tanto en configuraciones ST como en SMT. En modo ST, se ha asignado un único hilo por cada núcleo físico habilitado, lo que se incrementa conforme aumenta el número de núcleos físicos activos. Por otro lado, en modo SMT, se han asignado dos hilos por cada núcleo físico, de forma que los dos hilos del mismo núcleo participan en la ejecución. Para poder garantizar la reproducibilidad de los resultados y reducir la variabilidad entre ejecuciones, se ha fijado la frecuencia de operación de los procesadores durante los experimentos a 2500 MHz. Para la realización de los experimentos, se ha empleado un *framework* propio basado en Python que permite ejecutar las cargas de trabajo, monitorizar los contadores de prestaciones con Perf y almacenar los resultados.

IV. Estudio de caracterización

En esta sección se caracteriza el comportamiento de las aplicaciones del *benchmark suite* GAP a partir de su escalabilidad, del efecto del SMT y del análisis de las limitaciones microarquitectónicas. Para ello, se estudia su evolución con el número de hilos y núcleos, el impacto del segundo hilo *hardware* y los principales cuellos de botella identificados mediante la metodología de análisis de microarquitectura de Intel *Top-Down*.

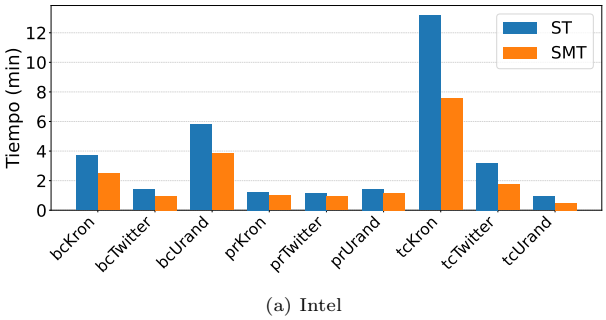


(a) Intel

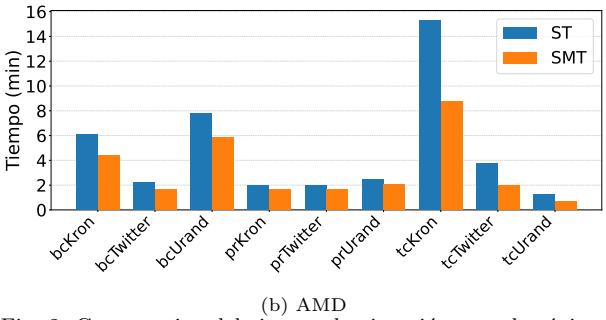


(b) AMD

Fig. 1: Evolución del *speedup* de las aplicaciones GAP en función del número de *cores*.



(a) Intel



(b) AMD

Fig. 2: Comparativa del tiempo de ejecución con el máximo número de *cores* disponibles en las configuraciones SMT y ST.

A. *Escalabilidad con número de hilos y núcleos*

Para poder evaluar el paralelismo y la capacidad de aprovechar los recursos al incrementar los hilos de cómputo, se ha realizado un estudio de la escalabilidad al incrementar el nivel de paralelismo en las dos plataformas previamente descritas. En las Figuras 1a y 1b esta representado el crecimiento del *speedup* en función de los núcleos que se asignan a la ejecución de las aplicaciones. Se observa que al incrementar el paralelismo, el *speedup* mejora, aunque la escalabilidad no es constante entre las aplicaciones. Las diferencias entre combinaciones algoritmo-grafo son reducidas cuando se asignan pocos *cores*, pero aumentan a medida que se van incrementando, sobre todo a partir de 16. En términos generales, las aplicaciones de TC son las que mejor escalan en ambas plataformas; BC presenta un comportamiento intermedio y PR muestra los menores valores de *speedup*, lo que pone en manifiesto que la capacidad de aprovechar el paralelismo disponible depende de manera significativa de la naturaleza de la carga.

B. *Efecto del SMT*

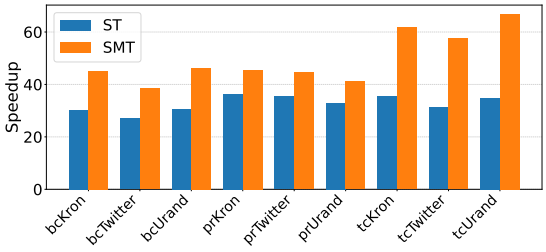
Con el fin de analizar el efecto del SMT sobre las prestaciones del *benchmark suite* GAP, en este apartado se comparan los resultados obtenidos en los modos ST y SMT para las plataformas Intel y AMD.

Las Figuras 2a y 2b muestran el tiempo de ejecución con el máximo número de *cores* disponibles en cada procesador: 24 para AMD y 32 para Intel. Se observa que el uso de SMT reduce el tiempo de ejecución de las nueve aplicaciones analizadas, tanto en AMD como en Intel. Sin embargo, esta mejora no es homogénea, ya que las mayores reducciones se observan en las aplicaciones de TC. Este algoritmo parece beneficiarse de la capacidad SMT para ocultar latencias y aumentar la ocupación efectiva de recursos, siendo el mayor beneficiado tcUrand con una mejora del 46,33 % para Intel y 46,58 % para AMD. Las aplicaciones de BC también presentan mejoras relevantes, aunque más moderadas, mientras que en PR el beneficio es claramente menor, siendo el peor caso para Intel, con una mejora del 16,34 % para prUrand y para AMD una mejora del 15,95 % para prKron.

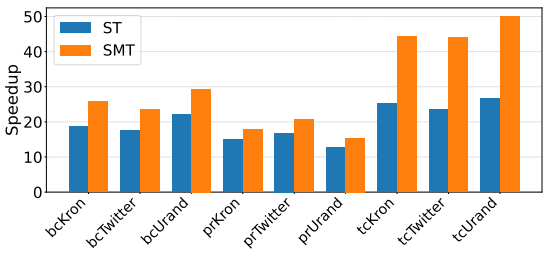
Las Figuras 3a y 3b muestran las gráficas de *speedup*, calculadas con respecto a 1 *core* ST, y confirman la misma tendencia observada en el tiempo de ejecución. En ambas plataformas, SMT supera a ST en las nueve aplicaciones, lo que indica que la reducción del tiempo de ejecución se traduce en una mejora efectiva de la escalabilidad. De igual forma que con el tiempo de ejecución, el incremento de *speedup* no es el mismo entre todos los algoritmos, observándose en TC la mayor mejora tanto en AMD como en Intel. En cambio, las mejoras en BC y PR son más moderadas. En Intel, las aplicaciones de BC muestran una mejora del 42 % en el peor de los casos, mientras que en PR la ganancia apenas alcanza el 25 % en sus tres aplicaciones. Por su parte, en AMD, las aplicaciones de BC mejoran aproximadamente un 20 %, mientras que las de PR alcanzan mejoras cercanas al 30 %. En Intel, las diferencias entre ST y SMT tienden a ser más marcadas. No obstante, este comportamiento no debe interpretarse directamente como una mayor eficiencia de SMT, ya que también influye el hecho de que en esta plataforma se utiliza un mayor número de núcleos.

C. *Análisis de la microarquitectura*

Para profundizar en las causas del comportamiento observado, en esta subsección se analizan los principales cuellos de botella microarquitectónicos a partir de la metodología propuesta por Intel *Top-Down*. En las Figuras 4a y 4b se muestra la distribución de los ciclos ejecutados en base a los cuellos de bote-

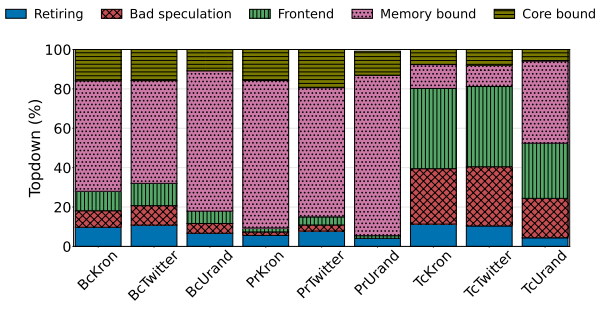


(a) Intel

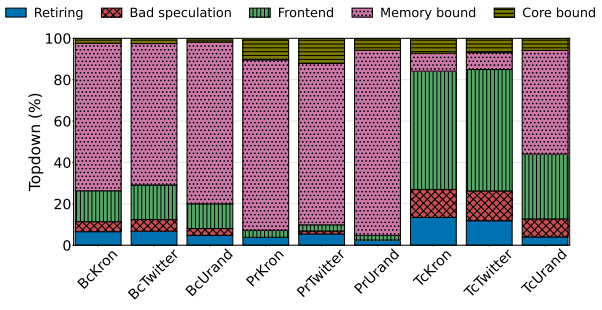


(b) AMD

Fig. 3: Comparativa del *speedup* obtenido con el máximo número de *cores* posibles para las configuraciones SMT y ST.



(a) Intel



(b) AMD

Fig. 4: Análisis con el método Top-Down con el máximo número de *cores* posibles en modo ST.

lla de la microarquitectura. En las aplicaciones más limitadas por *Frontend* o *Bad Speculation*, el SMT suele ayudar más, mientras que cuando predomina *Memory Bound*, su efecto tiende a ser menor, porque ambos hilos dependen de los mismos accesos a memoria y compiten por recursos compartidos. A partir de esta relación, es posible interpretar las diferencias observadas entre los algoritmos analizados.

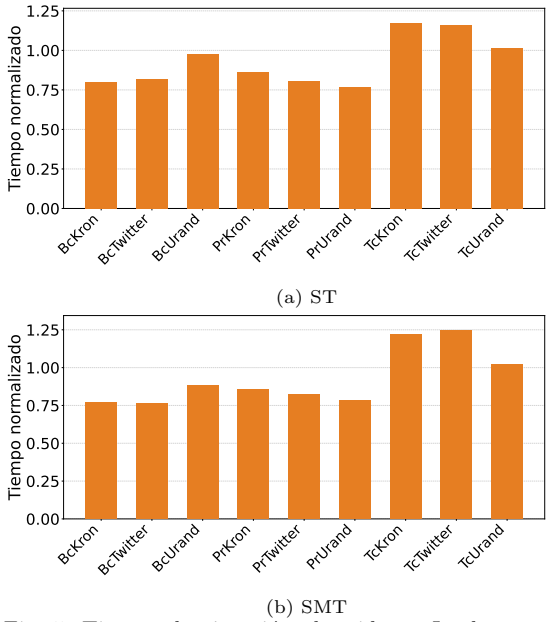
Si se fija la atención con más detalle en los algoritmos, el caso de TC resulta el más distinto de los demás. Este algoritmo presenta un flujo de control más irregular que el de los demás algoritmos analizados. Esto se debe a que la intersección de listas de adyacencia, que se realiza para encontrar los vecinos comunes entre dos nodos conectados, es una operación fundamental para detectar triángulos y depende de la estructura del grafo [13]. Como consecuencia, la secuencia de decisiones del algoritmo varía con frecuencia, lo que hace que el comportamiento de los saltos sea más irregular y, por lo tanto, más difícil de predecir [14]. Esta característica se refleja en que presenta porcentajes más altos de *Bad Speculation* que el resto. El incremento de esta categoría se acompaña de una mayor contribución de *Frontend*, lo que sugiere que las redirecciones del flujo de ejecución dificultan mantener un suministro continuo de instrucciones al *pipeline*.

Por otro lado, las aplicaciones de BC y PR presentan un comportamiento más dominado por la memoria que el de TC, y esto es coherente con el funcionamiento de ambos algoritmos. En el caso de BC, este recorre el grafo repetidas veces y accede continuamente tanto a listas de adyacencia como a diferentes estructuras auxiliares, como distancias, números de caminos y dependencias acumuladas [15]. Por su parte, PR también realiza múltiples iteraciones sobre el grafo para actualizar el valor de cada nodo a partir de las contribuciones de sus vecinos, lo que implica acceder a listas de adyacencia y a vectores de *ranks* [16]. Como consecuencia de estos comportamientos, la ejecución de ambos algoritmos está condicionada por el acceso a datos dispersos en la memoria [9]. Esto se ve reflejado en los porcentajes de *Memory Bound* que aparecen en el análisis *Top-Down*, lo que indica que una fracción significativa de los ciclos se pierde esperando datos del subsistema de memoria.

V. Comparativa entre Intel y AMD

Una vez analizadas ambas plataformas por separado, en esta sección se realiza una comparativa directa entre Intel y AMD a igual número de *cores*, en este caso 24, que es el máximo para AMD. En las Figuras 5a y 5b se muestra el tiempo de ejecución normalizado de Intel respecto a AMD; un tiempo normalizado inferior a 1 indica un mejor comportamiento de Intel, mientras que valores superiores favorecen a AMD. Como se ve en la Figura 5a en modo ST, la plataforma Intel obtiene los tiempos de ejecución más bajos para las aplicaciones de BC y PR, mientras que los tiempos de AMD son más bajos para TC. Si se observa la Figura 5b, donde se muestra el modo SMT, se observa que la misma tendencia se mantiene: Intel obtiene mejores resultados en BC y PR, pero AMD mantiene su comportamiento en TC. Por lo tanto, no se puede afirmar que existe una plataforma claramente superior para todas.

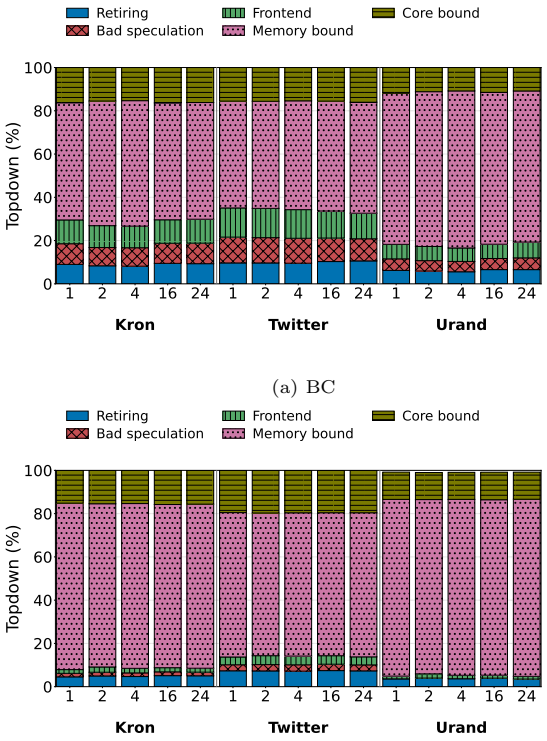
Analizando las diferencias entre los algoritmos vistos en el análisis de la microarquitectura, es coherente que la plataforma AMD obtenga mejores tiempos que Intel en TC, ya que el procesador AMD EPYC 9224 se basa en la microarquitectura Zen 4, para la que AMD destaca mejoras en *branch prediction*, una *micro-op cache* mayor y *buffers* internos más profundos [17]. Por lo tanto, para estas aplicaciones concretas, el funcionamiento de AMD parece más favorable para tolerar las irregularidades del flujo de control propias de TC, mientras que en Intel el impacto de



(a) ST

(b) SMT

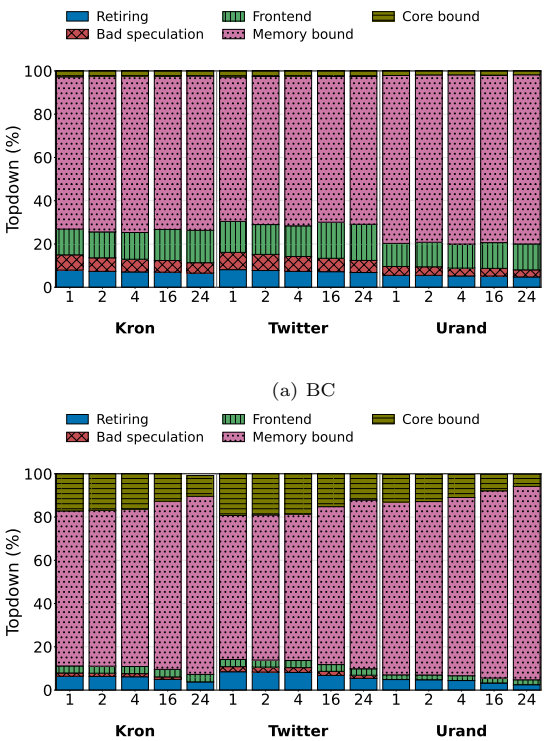
Fig. 5: Tiempo de ejecución obtenido en Intel normalizado respecto a AMD.



(a) BC

(b) PR

Fig. 6: Análisis Top-Down con el aumento de *cores* en Intel en modo ST.



(a) BC

(b) PR

Fig. 7: Análisis Top-Down con el aumento de *cores* en AMD en modo ST.

la especulación incorrecta resulta más visible. Por otro lado, el mejor comportamiento de Intel en BC y PR resulta coherente con las características de esta plataforma, ya que cuenta con una mayor capacidad de caché privada por núcleo que AMD, con 48 KB de L1D y 2 MB de L2 frente a los 32 KB de L1D y 1 MB de L2 de AMD, donde una mayor capacidad puede favorecer la reutilización de datos y reducir la presión sobre el resto de la jerarquía de memoria.

Las diferencias de escalabilidad de estas dos plataformas resultan coherentes si se analizan las Figuras 6 y 7 donde se ve la evolución de las métricas del método *Top-Down* al aumentar el número de *cores* con las configuraciones ST. Se observa que en AMD la categoría de *Memory Bound* alcanza valores elevados, especialmente en las aplicaciones de PR y de manera más moderada en BC, y que en varios casos aumenta progresivamente con el paralelismo. Mientras tanto, en Intel el *Top-Down* se mantiene más estable al aumentar el número de *cores*, de modo que la presión sobre la memoria no crece con la misma intensidad.

Finalmente, en la Figura 8 se observan las diferencias en *speedup* entre Intel y AMD. El valor se calcula dividiendo el *speedup* de Intel entre el *speedup* de AMD, ambos en 24 *cores*. Centrándonos en la comparación entre plataformas, los resultados vuelven a mostrar diferencias no uniformes. En las aplicaciones de BC y PR, el ratio es superior a 1, lo que indica que Intel alcanza un *speedup* mayor que el de AMD. Esta diferencia es más notable en PR, donde la ventaja de Intel es mucho mayor en comparación con BC, que obtiene resultados menos notorios, con algunos casos más cercanos a la igualdad. Por el contrario, las aplicaciones de TC presentan un comportamiento mucho más equilibrado entre plataformas, con los ratios situándose cerca de 1 e incluso por debajo en varias combinaciones, lo que indica que AMD iguala o supera ligeramente a Intel para este algoritmo. Estos resultados concuerdan con los que se han visto previamente, donde Intel mostraba unas mejores prestaciones para aquellas aplicaciones más dominadas por memoria y AMD un comportamiento más competitivo para TC, que depende más de la natu-

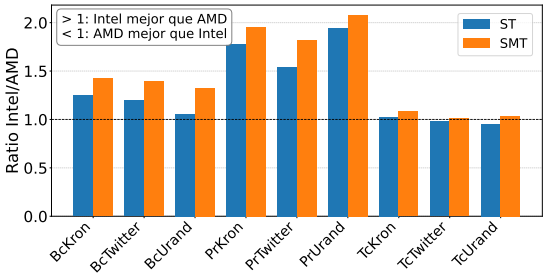


Fig. 8: Ratio *speedup* Intel/AMD para 24 *cores*.

raleza del algoritmo.

VI. Conclusiones

En este trabajo se ha realizado una caracterización del *benchmark* suite GAP en dos plataformas representativas de servidores actuales: Intel y AMD. Se han evaluado distintas configuraciones en términos de número de núcleos y modos de ejecución ST y SMT, analizando su comportamiento mediante métricas de tiempo de ejecución, *speedup* y el método de análisis microarquitectónico Top-Down, con el objetivo de identificar las principales tendencias de escalabilidad, la eficiencia del paralelismo y las limitaciones microarquitectónicas.

Los resultados del estudio muestran que las aplicaciones presentan comportamientos de escalabilidad muy diversos, altamente influenciados tanto por el algoritmo como por el grafo ejecutado. Asimismo, se observa que el efecto del SMT es, en general, positivo en ambas plataformas, aunque su impacto no es homogéneo. En particular, algoritmos como Triangle Counting (TC), con menores requisitos de memoria, se benefician en mayor medida de la activación del SMT.

En cuanto a la comparación entre plataformas, Intel muestra mejores resultados en las aplicaciones de Betweenness Centrality (BC) y PageRank (PR), cuyo comportamiento está más limitado por la memoria y se ve favorecido por una mayor cantidad de caché privada por núcleo frente a la arquitectura evaluada de AMD. Por el contrario, AMD presenta un comportamiento más favorable en TC, una aplicación con un mayor uso de los recursos del núcleo, lo que pone de manifiesto un *execution engine* más eficiente en esta arquitectura para este tipo de cargas.

En conclusión, este estudio evidencia que, dependiendo de las características de cada aplicación de grafos, los requisitos en términos de número de núcleos, uso de SMT y arquitectura pueden variar significativamente. Estos hallazgos abren nuevas oportunidades para futuras investigaciones centradas en el diseño de técnicas de optimización y de mecanismos de planificación capaces de adaptar dinámicamente estos factores, con el fin de mejorar la escalabilidad y la eficiencia del sistema.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación junto al FEDER europeo a través de los proyectos PID2021-123627OB-C51 y PID2024-158682OB-C31, y por la Generalitat Valenciana a través del proyecto CIPROM/2024/007.

Referencias

[1] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al., "Knowledge graphs," *ACM Computing Surveys (Csur)*, vol. 54, no. 4, pp. 1–37, 2021.

[2] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan, "One trillion edges: Graph processing at facebook-scale," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1804–1815, 2015.

[3] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui, "Graph neural networks in recommender systems: a survey," *ACM computing surveys*, vol. 55, no. 5, pp. 1–37, 2022.

[4] Sergey Brin and Lawrence Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[5] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang, "Knowledge vault: A web-scale approach to probabilistic knowledge fusion," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 601–610.

[6] Leslie F Sikos, "Cybersecurity knowledge graphs," *Knowledge and Information Systems*, vol. 65, no. 9, pp. 3511–3531, 2023.

[7] Stijn Eyerman, Wim Heirman, Kristof Du Bois, Joshua B. Fryman, and Ibrahim Hur, "Many-core graph workload analysis," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2018, pp. 282–292.

[8] Lucia Pons, Julio Sahuauillo, and Timothy M. Jones, "Dynamic allocation of processor cores to graph applications on commodity servers," in *2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2023, pp. 323–324.

[9] Alexandre Valentin Jamet, Georgios Vavouliotis, Daniel A Jiménez, Lluc Alvarez, and Marc Casas, "Practically tackling memory bottlenecks of graph-processing workloads," in *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2024, pp. 1034–1045.

[10] Scott Beamer, Krste Asanović, and David Patterson, "The gap benchmark suite," *arXiv preprint arXiv:1508.03619*, 2015.

[11] Ahmad Yasin, "A top-down method for performance analysis and counters architecture," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2014, pp. 35–44.

[12] Yan Kang, Sayan Ghosh, Mahmut Kandemir, and Andrés Márquez, "Studying cpu and memory utilization of applications on fujitsu a64fx and nvidia grace superchip," in *Proceedings of the International Symposium on Memory Systems*, 2024, pp. 198–207.

[13] Oded Green, James Fox, Alex Watkins, Alok Tripathy, Kasimir Gabert, Euna Kim, Xiaojing An, Kumar Aatish, and David A Bader, "Logarithmic radix binning and vectorized triangle counting," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 2018, pp. 1–7.

[14] Oded Green, Marat Dukhan, and Richard Vuduc, "Branch-avoiding graph algorithms," in *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*, 2015, pp. 212–223.

[15] Ulrik Brandes, "A faster algorithm for betweenness centrality," *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[16] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd, "The pagerank citation ranking: Bringing order to the web.," Tech. Rep., Stanford infolab, 1999.

[17] Advanced Micro Devices, "4th gen amd epyc processor architecture white paper," `https://www.amd.com/content/dam/amd/en/documents/products/epyc/4th-gen-epyc-processor-architecture-white-paper.pdf`, 2022, Accessed: 2026-04-05.

Computación cuántica (1/2)

Optimización de Circuitos para la Preparación de Estados Lógicos en Códigos Polares Cuánticos

Handy Kurniawan[1], Carmen G. Almudéver[2] y Francisco García Herrero[1]

Resumen—**La implementación de métodos eficientes para la corrección de errores cuánticos requiere la utilización de códigos de baja redundancia, cuya función es minimizar el número de qubits físicos necesarios por qubit lógico. Los códigos polares cuánticos son candidatos para este fin debido a que por definición pueden alcanzar la capacidad del canal mediante una codificación recursiva eficiente. Sin embargo, la fase de preparación del estado lógico constituye un cuello de botella, frecuentemente condicionado por un elevado sobrecoste de puertas y medidas. En el presente estudio, se expone una metodología para el código polar Q_1 que mitiga el sobrecoste de recursos mediante una optimización del circuito en el número de puertas y medidas. La propuesta ha sido evaluada mediante el entorno de simulación Stim, utilizando modelos de ruido a nivel de circuito sobre una topología que presenta conectividad *all-to-all*. El análisis realizado revela que la optimización de este trabajo aumenta la tasa de preparación de estados lógicos exitosos mientras mantiene o reduce las tasas de error lógico. A modo de ejemplo, para $N = 16$, las optimizaciones producen mejoras en ambos estados lógicos. Para el estado lógico $|+\rangle$, la propuesta incrementa la tasa de preparación hasta en un 34 %, y mejora las tasas de error lógico hasta en un 63 % en comparación con los valores de referencia. De igual manera, se evidencia un incremento en la tasa de preparación de hasta el 42 % y una reducción en la tasa de error lógico de hasta el 69 % para el estado lógico $|0\rangle$.**

Palabras clave— **Optimización de circuitos, Corrección de errores, Código polar cuántico**

I. Introducción

La corrección de errores cuánticos (QEC) constituye un componente fundamental para la consecución de una computación cuántica tolerante a fallos [1], [2], [3], [4], [5], [6], [7]. Las medidas en mitad del circuito cuántico (*mid-circuit measurements* o MCMs) y las operaciones de *feedforward* clásico—denominadas también circuitos cuánticos dinámicos—permiten implementar protocolos tolerantes a fallos en diversas plataformas, incluyendo iones atrapados [8], [9], [10] y procesadores cuánticos superconductores [11], [12]. Estas operaciones facilitan la preparación de estados entrelazados de largo alcance con una profundidad de circuito constante, lo que supone una ventaja de escalabilidad (y latencia) que, de forma demostrable, no pueden alcanzar los circuitos puramente unitarios limitados por su geometría [13], [14], [15], [16]. Las primeras implementaciones han demostrado el potencial de los circuitos dinámicos para reducir el sobrecoste de qubits, la profundidad del circuito y el número de puertas CNOT [16], [17], [18], [19], aspectos todos ellos beneficiosos para aumentar la precisión de los sistemas de QEC.

Por otra parte, mientras que los códigos cuánticos de comprobación de paridad de baja densidad (QLDPC)—como los códigos de superficie topológicos y los códigos *bivariate bicycle* (BB)—dominan actualmente el panorama de la tolerancia a fallos gracias a sus altos umbrales de corrección y su idoneidad para implementaciones a medio plazo [20], [21], los códigos polares cuánticos (QPC) siguen siendo un área de gran interés para futuras arquitecturas con alta conectividad. Este interés se ve impulsado al poder alcanzar una capacidad de corrección ideal, con tasas de codificaciones superiores [22], [23], [24], [25].

Trabajos recientes en compilación de circuitos cuánticos consciente del ruido han demostrado mejoras iniciales en la optimización de los QPC [26]; sin embargo, el sobrecoste de recursos sigue siendo sustancial. Por ello, este trabajo investiga una técnica de optimización aplicada a los QPC para reducir el sobrecoste de puertas CNOT. El objetivo no es desplazar a los códigos QLDPC a corto plazo, sino avanzar en la comprensión fundamental de las implementaciones de QPC demostrando que sus elevados requisitos de recursos pueden reducirse mediante técnicas de optimización, manteniendo o disminuyendo la tasa de error lógico (LER).

El protocolo estándar para la decodificación de los QPCs aplica extracciones de síndrome uniformes, incluso en las etapas iniciales de la preparación del estado donde los qubits se inicializan en $|0\rangle^{\otimes N}$. Dado que la tasa de error de la inicialización de los qubits es a menudo comparable o inferior a la de las puertas de dos qubits y las operaciones de lectura [27], realizar la extracción de síndromes en esta etapa resulta contraproducente; el ruido introducido por los propios operadores de comprobación a menudo supera los errores dispersos que se pretenden detectar [28], [29].

Este trabajo se basa en el fundamento teórico de los QPC tolerantes a fallos establecido por los autores de [30], quienes propusieron la construcción recursiva inicial y el *error-detection gadget*. Partiendo de esta base, proponemos una técnica de optimización que mitiga las ineficiencias de dicho diseño para mejorar la precisión de los QPC. Aprovechamos la inicialización fija de los qubits para eliminar las comprobaciones contraproducentes. Al aplicar el principio de medida diferida (PDM) [31] y técnicas de reducción

[1]Dpto. de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, e-mail: {handykur,francg18}@ucm.es.

[2]Dpto. de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mail: cargara2@disca.upv.es.

de puertas [29], simplificamos la primera etapa del circuito en operaciones unitarias estáticas. Esto reduce la magnitud del ruido generado en la primera capa de comprobación, aumentando así la tasa de preparación y reduciendo la LER.

Para validar la eficacia de esta técnica, realizamos simulaciones utilizando Stim [32] bajo un modelo de ruido a nivel de circuito sobre una topología con conectividad *all-to-all*. El enfoque propuesto mejora el rendimiento de los códigos QPC. Por ejemplo, para el estado lógico $|+\rangle$ con una longitud de código $N = 32$, nuestra estrategia mejora la tasa de preparación de estados hasta en un 34 % mientras que reduce el número de puertas cuánticas y la profundidad del circuito. Cabe destacar que, en términos de precisión, observamos reducciones de la LER de hasta el 63 % al comparar nuestro enfoque con la implementación estándar de los QPC. Se observan mejoras similares para el estado lógico $|0\rangle$, con aumentos en la tasa de preparación de hasta el 42 % y reducciones de la LER de hasta el 69 %.

El resto de este artículo se organiza de la siguiente manera. La Sección II establece el marco teórico de la extracción de síndromes y la propagación de errores en QEC, revisa las técnicas de optimización de circuitos y ofrece una visión general de los QPC. La Sección III describe las técnicas de optimización propuestas, el entorno experimental y las métricas de rendimiento. La Sección IV presenta los resultados de la simulación, analizando el rendimiento y la eficacia de la técnica. Finalmente, la Sección V concluye el artículo con un resumen de las principales contribuciones y una discusión sobre futuras líneas de investigación.

II. Contexto

Esta sección establece los fundamentos teóricos necesarios para la propuesta. Se comienza definiendo los mecanismos de propagación de errores y los protocolos de extracción de síndromes para los códigos QEC. Posteriormente, se revisan las técnicas de optimización de circuitos pertinentes. Finalmente, se ofrece una visión general de los QPC.

A. Extracción de síndromes y propagación de errores

El circuito de extracción de síndromes utiliza qubits auxiliares, denominados *ancilla*, para detectar errores en el sistema sin colapsar su estado cuántico de los datos. Este procedimiento se basa en la interacción entre los errores y las puertas CNOT. Los errores se propagan a través de las CNOT de la siguiente manera: un error X en el control se desplaza al target, mientras que un error Z en el target se propaga hacia el control (ver Fig. 1). Estas reglas sustentan el mecanismo de las medidas de estabilizadores [33], [34].

Tal como se ilustra en la Fig. 2, el qubit ancilla interactúa con los qubits de datos mediante puertas CNOT para extraer información de paridad. Para la detección de un bit-flip (error de tipo X), mostrada en la Fig. 2a, el qubit ancilla actúa como el target de

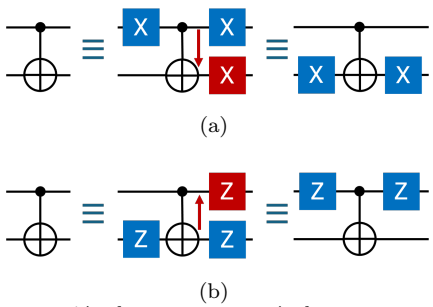


Fig. 1: Propagación de errores a través de una puerta CNOT. (a) Propagación de un error X de un solo qubit. Un error X en el control se propaga al *target*, mientras que un error X en el *target* no se propaga al control. (b) Propagación de un error Z. Un error Z en el target se propaga hacia atrás al control, mientras que un error Z en el control no se propaga al *target*.

las CNOT controladas por los qubits de datos. Según indican los marcadores rojos, un error X en el qubit de datos se propaga hacia adelante en el qubit ancilla (cambiando su estado de $|0\rangle$ a $|1\rangle$), lo que genera un error al realizar la medida. Por el contrario, para la detección de *phase-flip* (error de tipo Z), (Fig. 2b), la ancilla controla las operaciones CNOT. En este caso, un error de cambio de fase (Z) en el qubit de datos se propaga al qubit *ancilla*.

La extracción de los síndromes conlleva un coste de hardware elevado, dado que las puertas de dos qubits y las operaciones de medida presentan tasas de error superiores a las de las puertas de un solo qubit. Por tanto, la reducción del sobrecoste de este procedimiento es de utilidad ya que aumenta la precisión de esta operación. En este trabajo, se aborda este reto en el marco de los QPC aplicando las técnicas de optimización de circuitos cuánticos descritas en la Sec. II-B.

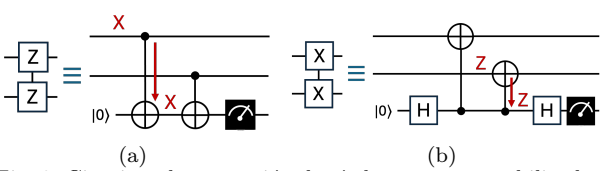


Fig. 2: Circuitos de extracción de síndrome para estabilizadores de *Pauli* de dos qubits. (a) Medida de *Pauli* $Z \otimes Z$ para la detección de errores de *bit-flip* (X). Los marcadores rojos muestran la propagación de un error X desde el qubit de datos (parte superior) a la *ancilla* (parte inferior) a través de la puerta CNOT. (b) Medida de *Pauli* $X \otimes X$ para la detección de errores de *phase-flip* (Z). En este caso, un error Z en el qubit de datos se propaga hacia el *target* de la CNOT a la *ancilla* de *control*.

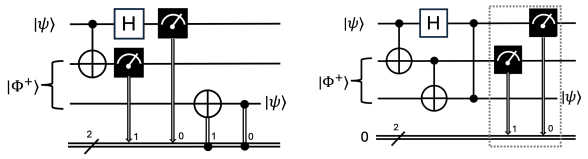
B. Optimización de circuitos cuánticos

La optimización de circuitos cuánticos permite mejorar la eficiencia computacional y reducir los errores introducidos por las operaciones cuánticas. En las últimas dos décadas, se han propuesto diversas técnicas de optimización que se clasifican en enfoques heurísticos [35], [36], basados en aprendizaje automático [37], [38], síntesis unitaria [39], [40], [41] y algorítmicos [42], [43]. La eficacia de estas técnicas se evalúa mediante métricas como la profundidad del circuito, el recuento total de puertas, el número de qubits, el número de puertas de dos qubits, la fideli-

dad de las puertas y el grado de paralelismo [44]. En este trabajo se emplean estrategias tanto algorítmicas como heurísticas; específicamente, se aplican: (i) la medida diferida [31] y (ii) la reducción de puertas [35], [36].

B.1 Medida diferida

El principio de medida diferida (PDM) establece que realizar una medida a mitad de un circuito cuántico equivale a posponerla hasta el final [31]. En diversos casos, los resultados de las MCM determinan operaciones futuras (*feedforward* clásico y flujo de *control*). Para aplazar las medidas, se sustituyen estos pasos controlados clásicamente por puertas controladas cuánticamente, donde el qubit no medido actúa como el *control*. Este enfoque permite optimizaciones posteriores de la profundidad del circuito y del recuento de puertas. La Fig. 3 ilustra un circuito de teletransporte implementado mediante la transformación PDM.



(a) (b)

Fig. 3: Circuito de teletransporte: (a) las medidas se sitúan en mitad del circuito donde se emplea la operación de *feedforward*, (b) medida al final del circuito basada en el principio de medida diferida, donde las medidas resultan opcionales [31].

B.2 Reducción de puertas

La reducción de puertas es una forma de optimización de circuitos cuánticos centrada en minimizar el número total de puertas necesarias para implementar una operación unitaria específica, a menudo mediante el uso de equivalencias entre puertas conocidas (p. ej., $HH = I$ or $XX = I$) o relaciones de conmutación [29].

En este trabajo, aplicamos estas técnicas a los circuitos de extracción de síndrome detallados en la Sección III-A. En concreto, se analiza el estado del circuito tras la aplicación de PDM para identificar operaciones condicionales que resultan funcionalmente equivalentes a la identidad; por ejemplo, operaciones de *feedforward* controladas por un qubit que se encuentra en el estado $|0\rangle$. Al reducir estas puertas redundantes, se comprime el recuento de puertas y la profundidad del circuito, lo que afecta a la eficiencia de la preparación de estado.

C. Códigos polares cuánticos

Los QPC son una familia de códigos QEC procedentes de la teoría de la información clásica [45], diseñados para alcanzar tasas de codificación elevadas mediante el fenómeno de la polarización de canal. Su procedimiento de preparación del estado cuántico tolerante a fallos [30] construye generadores de estabilizadores de forma recursiva mediante medidas de *Pauli* de dos qubits, lo que se denomina *error-detection gadget*.

Este protocolo está diseñado para restringir la propagación de errores. Mediante la estructura recursiva, las comprobaciones dinámicas se entrelazan de modo que un fallo en la preparación de estado se detecta antes de convertirse en un error lógico de peso alto, es decir, múltiples errores físicos generan un único error lógico. Esto permite que los fallos físicos individuales permanezcan localizados e identificables. Además, aunque esta estructura recursiva permite la tolerancia a fallos, implica el uso de estabilizadores de peso alto a medida que aumenta el tamaño del código. En consecuencia, la extracción de síndrome estándar requiere un número elevado de recursos, lo que constituye la motivación para las optimizaciones de circuito propuestas en este trabajo.

C.1 Preparación del estado

Este trabajo se centra en los códigos polares Q_1, que codifican un único qubit lógico en $N = 2^n$ qubits de datos, donde $n > 0$ es el nivel de polarización. La construcción del código depende de la posición en la que se codifica la información $i \in \{1, \ldots, N\}$, donde la posición para el qubit lógico es $i_{|0\rangle} = i$ e $i_{|+\rangle} = i + 1$. Sean $b_1, \ldots, b_n$ la expansión binaria de $i - 1$, siendo b_n el bit más significativo. Si $b_k = 0$ ($b_k = 1$), se aplica una medida de Pauli $X \otimes X$ ($Z \otimes Z$) en el nivel de recursión $k = 1, ..., n$. La Figura 4 muestra un ejemplo de la preparación basada en medidas de un estado lógico $|+\rangle$ para $N = 8$ y una posición de la información de $i_{|+\rangle} = 4$ ($b = (0, 1, 0)$).

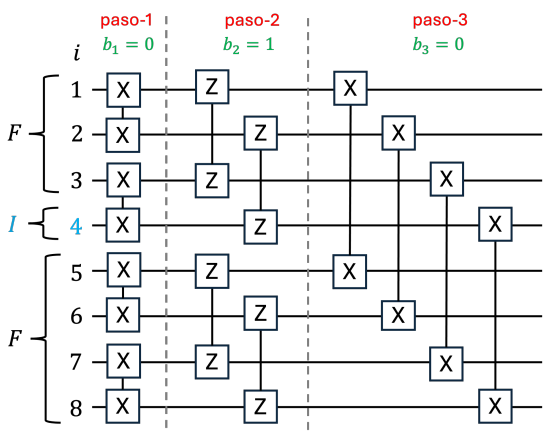


Fig. 4: Preparación basada en medidas del estado lógico $|+\rangle$ para el código Q_1 ($N = 8$, $i_{|+\rangle} = 4$, $b = (0, 1, 0)$) [30], donde i representa la posición del qubit de información en el código polar. F denota los *frozen qubits*, que generan la redundancia del código, I es el *information qubit*, y $b_k = 0$ ($b_k = 1$) indica la medida de *Pauli* $X \otimes X$ ($Z \otimes Z$) basada en la Fig. 2. Todos los qubits se inicializan en $|0\rangle$.

En la práctica, para las posiciones de información que comienzan con medidas $Z \otimes Z$ ($b_k = 1$), estos pasos iniciales pueden omitirse hasta que se produzca la primera medida $X \otimes X$ ($b_k = 0$). Esta simplificación es posible debido a que los qubits se inicializan en el estado $|0\rangle$, lo que hace que el resultado de la medida sea determinista y la operación de la puerta resulte redundante [33]. Por ejemplo, en el caso del código Q_1 ($N = 8, i_{|+\rangle} = 5$) con un patrón $b = (1, 1, 0)$, se

omiten las dos primeras comprobaciones de tipo Z (Z-*checks*) y la ejecución del circuito comienza directamente en el tercer paso ($X \otimes X$), tal como se ilustra en la Fig. 5.

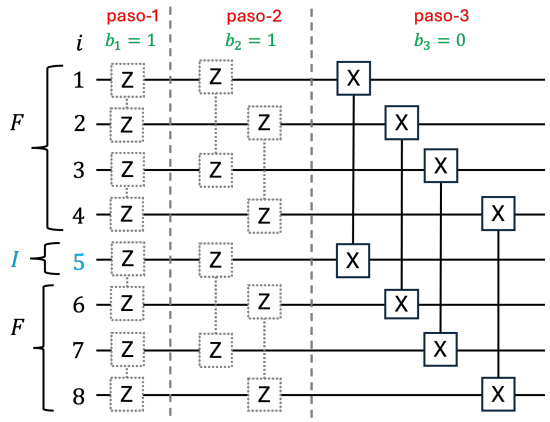


Fig. 5: Preparación basada en medidas del estado lógico $|+\rangle$ para el código Q_1 ($N = 8$, $i_{|+\rangle} = 5$, $b = (1, 1, 0)$). Las medidas de Pauli $Z \otimes Z$, indicadas por un patrón de puntos, pueden omitirse hasta las primeras medidas de Pauli $X \otimes X$.

III. METODOLOGÍA

En esta sección se define la técnica de optimización propuesta y se detalla el marco experimental utilizado para su validación. Se comienza definiendo el entorno de simulación y el modelo de ruido a nivel de circuito. Posteriormente, se describen las métricas de evaluación empleadas para cuantificar la tasa de preparación de estados lógicos y la LER.

A. Optimización de circuitos

Se implementa un flujo de optimización basado en la coincidencia de patrones que busca subcircuitos susceptibles de una representación equivalente con un número reducido de puertas y medidas [29]. En este trabajo, se aplican dos protocolos complementarios ilustrados en la Fig. 6: (i) *medida diferida* y (ii) *reducción de puertas*.

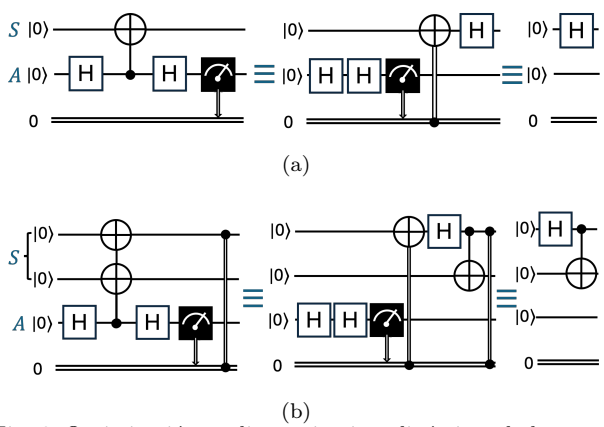


Fig. 6: Optimización mediante circuitos dinámicos de la syndrome extraction. (a) Ejemplo de una medida de X de un solo qubit, (b) extensión a una medida $X \otimes X$ de dos qubits. Ambas se muestran en tres etapas: la forma estándar (izquierda), PDM (centro) y la forma simplificada (derecha). Todos los qubits se inicializan en el estado $|0\rangle$.

A.1 Medida diferida

Se utiliza el PDM [31] para reestructurar el circuito para su optimización. La Fig. 6 ilustra este flujo de transformación para (a) una medida de X de un solo qubit y (b) una medida $X \otimes X$ de dos qubits. La transformación se realiza en tres etapas: la forma estándar (izquierda), la forma diferida (centro), donde los controles cuánticos se convierten en operaciones de *feedforward* clásico, y la forma reducida (derecha) tras la optimización.

Dado que la preparación de los QPC comienza con medidas $X \otimes X$ con qubits inicializados en $|0\rangle$, la primera capa de comprobación puede reestructurarse mediante PDM. Aunque desplazar la medida desacopla formalmente el qubit ancilla A de los qubits de datos S, la inicialización conocida asegura que se preserve la integridad lógica del circuito. En resumen, la aplicación del PDM tiene un propósito preparatorio: expone la estructura del circuito, haciéndola accesible para los pasos posteriores de reducción de puertas.

A.2 Reducción de puertas

La transformación PDM expone operaciones dentro del *error-detection gadget* de los QPC que resultan funcionalmente redundantes. En la configuración diferida (Fig. 6, centro), la reordenación de las puertas sitúa dos operaciones *Hadamard* en la línea del qubit *ancilla* en una secuencia directa. Dado que $HH = I$, estas puertas se cancelan resultando en la identidad, lo que simplifica la base de medida efectiva.

Debido a que los qubits se inicializan en $|0\rangle$, este circuito simplificado garantiza un resultado determinista de '0' (valor propio +1). Por tanto, la condición para activar las correcciones de *feedforward* (un resultado de '1') nunca se cumple. En consecuencia, las operaciones condicionales actúan de forma efectiva como el operador identidad y se eliminan del circuito, dando lugar a la forma optimizada que se muestra en la parte derecha de la Fig. 6.

Al aplicar esta lógica a las comprobaciones de estabilizadores de las etapas iniciales, se elimina la necesidad de las operaciones de MCM y *feedforward* asociadas. Esta optimización disminuye el número de recursos necesarios; para un bloque de tamaño n, esta optimización combinada elimina 2^{n-1} operaciones CNOT y *Hadamard* y 2^{n-1} operaciones de medida, reduciendo la profundidad efectiva del circuito para estos bloques de 5 a 2. Esta reducción preserva la equivalencia lógica exacta estrictamente bajo la condición de que el estado de entrada sea $|0\rangle^{\otimes n}$. Por el contrario, para un estado de entrada $|1\rangle^{\otimes n}$, los resultados de las medidas activarían las operaciones de *feedforward*, haciendo que esta optimización no sea aplicable.

B. Simulaciones

Los experimentos se realizan utilizando el simulador Stim [32], empleando un modelo de ruido Pauli estándar. Específicamente, se aplica ruido de depo-

larización a las puertas de un solo qubit (*Hadamard*) y de dos qubits (CNOT), mientras que las operaciones de inicialización, reinicio y medida están sujetas a errores de *bit-flip* (X). Los errores de medida se modelan mediante la inserción de un error X inmediatamente antes de la operación de medida. Para evaluar los protocolos, se considera el modelo de ruido a nivel de circuito. En este escenario, se asume una arquitectura con conectividad *all-to-all* entre qubits, asignando probabilidades de error idénticas a todos los qubits y operaciones. Se evalúa el rendimiento a través de un barrido de tasas de error físico: 10^{-2}, 5×10^{-3}, 10^{-3}, 5×10^{-4} y 10^{-4}.

B.1 Alcance experimental y reproducibilidad

El análisis se restringe a longitudes de código $N \in \{8, 16, 32\}$. Para asegurar la significación estadística, particularmente en el régimen de baja LER, se simulan entre 10^8 y 10^{10} lanzamientos por configuración, utilizando múltiples semillas aleatorias para promediar las realizaciones de ruido estocástico. Las simulaciones se ejecutaron en dos entornos de computación con Ubuntu 20.04: un sistema con 12 CPUs y 16 GB de RAM (Python 3.13.12) y un sistema con 24 CPUs y 125,6 GB de RAM (Python 3.11.13). Ambos entornos utilizaron Stim v1.15.0 [32].

C. Métricas de rendimiento

Para evaluar las técnicas propuestas, se utiliza un conjunto de métricas que categorizan el rendimiento en los dos dominios siguientes:

C.1 Tasa de preparación ($\boldsymbol{p_{prep}}$)

La tasa de preparación mide el porcentaje de casos en los que el proceso de preparación del estado es exitoso. Representa la probabilidad de que una ejecución supere todas las comprobaciones de detección de errores y sea aceptada como estado válido para la decodificación:

$$p_{prep} = \frac{n_{prep}}{n_{\text{tot}}}, \tag{1}$$

donde n_{prep} es el número de ejecuciones aceptadas y n_{tot} es el número total de ejecuciones realizadas.

C.2 Tasa de error lógico (LER)

La LER actúa como la medida de la calidad del estado válido. Cuantifica la probabilidad de que el estado sea decodificado correctamente, es decir, si presenta un error lógico o no, condicionado a haber superado las comprobaciones de preparación:

$$\text{LER} = \frac{n_{\text{err}}}{n_{prep}}, \tag{2}$$

donde n_{err} es el número de lanzamientos aceptados que fallan en la decodificación. Al realizar el cálculo de la tasa sobre n_{prep}, se aisla la precisión de los estados aceptados de la eficiencia del proceso de selección de dichos estados. La relación entre la LER y el sobrecoste de muestreo da lugar a un espacio de diseño en el que existe un compromiso entre el coste de preparación y la calidad del estado resultante.

IV. RESULTADOS Y DISCUSIÓN

En esta sección se evalúa la propuesta de este trabajo (identificada como *prop* en las figuras). La Figura 7 ofrece una visión de las mejoras de rendimiento. Los resultados demuestran que nuestras optimizaciones producen mejoras simultáneas en la tasa de preparación y en la LER, lo que valida la hipótesis de que las capas de comprobación iniciales en el protocolo estándar pueden optimizarse para reducir el sobrecoste de puerta sin comprometer la precisión de la detección de errores.

A. Tasa de preparación

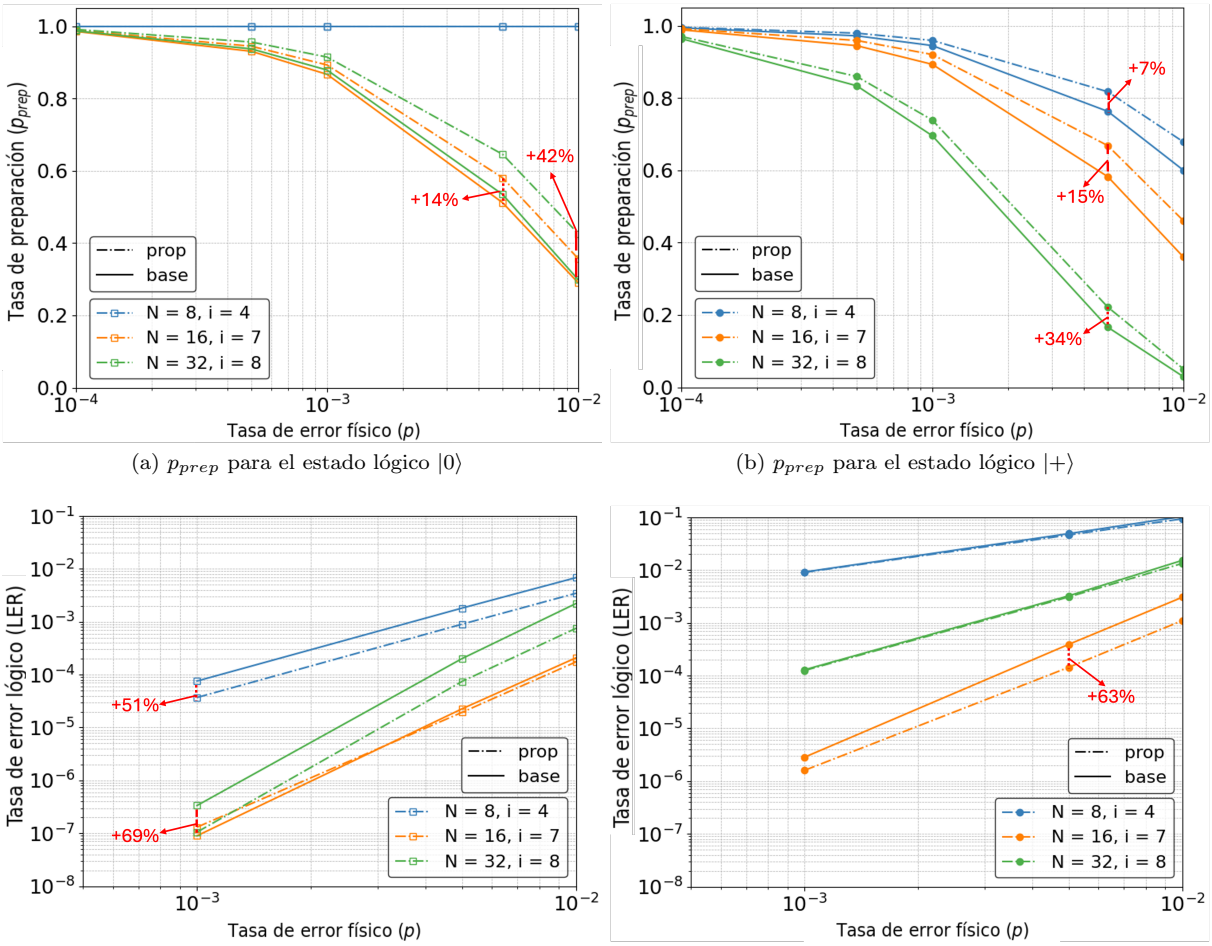
Las Figuras 7a y 7b ilustran las tasas de preparación para los estados lógicos $|0\rangle$ y $|+\rangle$, respectivamente. Se observa que la brecha de rendimiento entre los protocolos se reduce a medida que disminuye la tasa de error físico (p). Este comportamiento es el esperado, ya que el ruido acumulado para las longitudes de código $N \in \{8, 16, 32\}$ disminuye en niveles de ruido físico más bajos, lo que resulta en un menor número de estados rechazados en ambos protocolos.

La técnica propuesta demuestra mejoras en casi todas las configuraciones. Cabe destacar que, para el estado lógico $|+\rangle$ con $p = 5 \times 10^{-3}$, la tasa de preparación aumenta un 7 %, 15 % y 34 % para $N = 8, 16,$ y 32, respectivamente. Se observa una tendencia similar para el estado $|0\rangle$, donde la ganancia alcanza el 42 % para $N = 32$. La única excepción es $N = 8$ para el estado $|0\rangle$, donde la tasa de preparación ya está saturada en el caso base y no puede optimizarse. Estos resultados confirman que la técnica propuesta reduce el sobrecoste de las puertas CNOT, lo que se traduce directamente en una mayor tasa de preparación.

B. Tasa de error lógico

Esta ganancia en el rendimiento no se produce a expensas de la precisión en términos de LER. Las Figuras 7c y 7d muestran la LER para los estados $|0\rangle$ y $|+\rangle$. Para $p = 10^{-3}$ en el estado lógico $|0\rangle$, la propuesta reduce la LER en un 51 % para $N = 8$ y en un 80 % para $N = 32$. Para el estado lógico $|+\rangle$ con $p = 5 \times 10^{-3}$, N=16 presenta una mejora del 63 %. Si bien la LER para algunas configuraciones (p. ej., $N = 8$ y $N = 32$ en el estado $|+\rangle$) sigue siendo comparable a la del caso base, la ausencia de degradación del rendimiento, combinada con la mejora en la tasa de preparación, aumenta la eficiencia del protocolo optimizado.

Una observación destacable en las Figuras 7c y 7d es que la LER para $N = 32$ es superior a la de $N = 16$, a pesar del mayor tamaño del código. Este comportamiento se atribuye a la posición de la información i específica elegida para cada código, ya que en la preparación de estados basada en códigos polares, la eficacia de la detección de errores es altamente sensible al índice seleccionado para situar el qubit de datos. Para $N = 32$ con $i = 8$, la profundidad del circuito y el número de puertas CNOT aumentan en comparación con $N = 16$ con $i = 7$. Este incremento en la complejidad introduce más ruido físico en el sis-



(a) p_{prep} para el estado lógico $|0\rangle$

(b) p_{prep} para el estado lógico $|+\rangle$

(c) LER para el estado lógico $|0\rangle$

(d) LER para el estado lógico $|+\rangle$

Fig. 7: Evolución de la tasa de preparación y de la LER ante variaciones en el ruido físico. Los resultados se detallan para la posición de la información i. Las líneas continuas representan el protocolo de referencia, mientras que las líneas de punto y raya corresponden al protocolo optimizado propuesto en este trabajo. Los colores distinguen la longitud de código N y la posición de la información i específicas.

tema; si las propiedades de detección de errores de la posición $i = 8$ no escalan lo suficiente para suprimir este ruido adicional, el resultado es una tasa de error lógico residual más elevada. Esto pone de manifiesto la importancia de un compromiso entre la distancia teórica de un código y el sobrecoste en complejidad de circuito cuántico requerido para preparar sus estados lógicos.

V. Conclusión y trabajo futuro

Este estudio ha presentado una técnica de optimización para la preparación de estado de los QPC. Utilizando el simulador Stim bajo un modelo de ruido a nivel de circuito con conectividad *all-to-all*, hemos demostrado que la extracción de síndromes estándar en las etapas iniciales del circuito de preparación puede resultar contraproducente debido al ruido inducido por las puertas y las medidas. Al eliminar estas comprobaciones redundantes y simplificar las capas iniciales en operaciones unitarias optimizadas, hemos logrado un incremento en el rendimiento de la preparación de estados sin comprometer la precisión en términos de tasa de error lógico. Nuestros resultados indican que el protocolo propuesto escala eficazmente con la longitud del código, alcanzando mejoras en la tasa de preparación de hasta el 42 % y reducciones de la LER de hasta el 69 % para $N = 32$ en el estado lógico $|0\rangle$.

Como trabajo futuro, se plantean diversas líneas. En primer lugar, investigaremos el impacto de la optimización del circuito con una variación de las posiciones de los qubits de datos i, ya que nuestros resultados sugieren que la elección de i influye en la eficacia de la detección de errores en códigos más grandes. En segundo lugar, nuestro objetivo es implementar circuitos dinámicos que empleen medidas en MCM para omitir ejecuciones de puertas posteriores una vez detectado un error; este enfoque permitiría reducir recursos tanto cuánticos como clásicos. Por último, se estudiará la extensión de esta optimización a arquitecturas con conectividad limitada que supondrá un paso necesario para la implementación de QPCs de alta tasa en el hardware actual.

Agradecimientos

Esta investigación forma parte del proyecto PID2023-147059OB-I00 financiado por MCIU/AEI/10.13039/501100011033/FEDER, UE. Handy Kurniawan agradece el apoyo de la Comunidad de Madrid bajo el número de beca

PIPF-2023/COM-30051. Carmen G. Almudéver agradece el apoyo del Ministerio de Ciencia, Innovación y Universidades y del FEDER de la Unión Europea bajo la subvención PID2024-158682OB-C31, así como del Ministerio para la Transformación Digital y de la Función Pública a través de la convocatoria del proyecto QUANTUM ENIA - proyecto Quantum Spain, y de la Unión Europea a través del Plan de Recuperación, Transformación y Resiliencia - NextGenerationEU en el marco de la Agenda España Digital 2026.

Referencias

[1] Peter W Shor, "Scheme for reducing decoherence in quantum computer memory," *Physical review A*, vol. 52, no. 4, pp. R2493, 1995.

[2] A Robert Calderbank and Peter W Shor, "Good quantum error-correcting codes exist," *Physical Review A*, vol. 54, no. 2, pp. 1098, 1996.

[3] Peter W Shor, "Fault-tolerant quantum computation," in *Proceedings of 37th conference on foundations of computer science*. IEEE, 1996, pp. 56–65.

[4] Sergey Bravyi, Oliver Dial, Jay M Gambetta, Darío Gil, and Zaira Nazario, "The future of quantum computing with superconducting qubits," *Journal of Applied Physics*, vol. 132, no. 16, 2022.

[5] David Aasen, Morteza Aghaee, Zulfi Alam, Mariusz Andrzejczuk, Andrey Antipov, Mikhail Astafev, Lukas Avilovas, Amin Barzegar, Bela Bauer, Jonathan Becker, et al., "Roadmap to fault tolerant quantum computation using topological qubit arrays," *arXiv preprint arXiv:2502.12252*, 2025.

[6] IBM Quantum, "IBM lays out clear path to fault-tolerant quantum computing — IBM Quantum Computing Blog," 2025, Available: https://www.ibm.com/quantum/blog/large-scale-ftqc (Accessed Jan. 16, 2026).

[7] Muhammad AbuGhanem, "Toward scalable fault-tolerant photonic quantum computers," *The Journal of Supercomputing*, vol. 82, no. 2, pp. 51, 2026.

[8] C Ryan-Anderson, NC Brown, MS Allman, B Arkin, G Asa-Attuah, C Baldwin, J Berg, JG Bohnet, S Braxton, N Burdick, et al., "Implementing fault-tolerant entangling gates on the five-qubit code and the color code," *arXiv preprint arXiv:2208.01863*, 2022.

[9] Matthew DeCross, Eli Chertkov, Megan Kohagen, and Michael Foss-Feig, "Qubit-reuse compilation with mid-circuit measurement and reset," *Physical Review X*, vol. 13, no. 4, pp. 041057, 2023.

[10] Mohsin Iqbal, Nathanan Tantivasadakarn, Thomas M Gatterman, Justin A Gerber, Kevin Gilmore, Dan Gresh, Aaron Hankin, Nathan Hewitt, Chandler V Horst, Mitchell Matheny, et al., "Topological order from measurements and feed-forward on a trapped ion quantum computer," *Communications Physics*, vol. 7, no. 1, pp. 205, 2024.

[11] Antonio D Córcoles, Maika Takita, Ken Inoue, Scott Lekuch, Zlatko K Minev, Jerry M Chow, and Jay M Gambetta, "Exploiting dynamic quantum circuits in a quantum algorithm with superconducting qubits," *Physical Review Letters*, vol. 127, no. 10, pp. 100501, 2021.

[12] Daniel Hothem, Jordan Hines, Charles Baldwin, Dan Gresh, Robin Blume-Kohout, and Timothy Proctor, "Measuring error rates of mid-circuit measurements," *Nature Communications*, vol. 16, no. 1, pp. 5761, 2025.

[13] Sergey Bravyi, David Gosset, and Robert König, "Quantum advantage with shallow circuits," *Science*, vol. 362, no. 6412, pp. 308–311, 2018.

[14] Michael Foss-Feig, Arkin Tikku, Tsung-Cheng Lu, Karl Mayer, Mohsin Iqbal, Thomas M Gatterman, Justin A Gerber, Kevin Gilmore, Dan Gresh, Aaron Hankin, et al., "Experimental demonstration of the advantage of adaptive quantum circuits," *arXiv preprint arXiv:2302.03029*, 2023.

[15] Michael Foss-Feig, Guido Pagano, Andrew C Potter, and Norman Y Yao, "Progress in trapped-ion quantum simulation," *Annual Review of Condensed Matter Physics*, vol. 16, no. 1, pp. 145–172, 2025.

[16] Elisa Bäumer and Stefan Woerner, "Measurement-based long-range entangling gates in constant depth," *Physical Review Research*, vol. 7, no. 2, pp. 023120, 2025.

[17] Fei Hua, Yuwei Jin, Yanhao Chen, Suhas Vittal, Kevin Krsulich, Lev S Bishop, John Lapeyre, Ali Javadi-Abhari, and Eddy Z Zhang, "CaQR: A compiler-assisted approach for qubit reuse through dynamic circuit," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 59–71.

[18] Siyuan Niu, Efekan Kokcu, Anupam Mitra, Aaron Szasz, Akel Hashim, Justin Kalloor, Wibe Albert de Jong, Costin Iancu, and Ed Younis, "AC/DC: Automated compilation for dynamic circuits," *arXiv preprint arXiv:2412.07969*, 2024.

[19] Elisa Bäumer, Vinay Tripathi, Derek S Wang, Patrick Rall, Edward H Chen, Swarnadeep Majumder, Alireza Seif, and Zlatko K Minev, "Efficient long-range entanglement using dynamic circuits," *PRX Quantum*, vol. 5, no. 3, pp. 030339, 2024.

[20] Theodore J Yoder, Eddie Schoute, Patrick Rall, Emily Pritchett, Jay M Gambetta, Andrew W Cross, Malcolm Carroll, and Michael E Beverland, "Tour de gross: A modular quantum computer based on bivariate bicycle codes," *arXiv preprint arXiv:2506.03094*, 2025.

[21] Alexander J Malcolm, Andrew N Glaudell, Patricio Fuentes, Daryus Chandra, Alexis Schotte, Colby DeLisle, Rafael Haenel, Amir Ebrahimi, Joschka Roffe, Armanda O Quintavalle, et al., "Computing efficiently in QLDPC codes," *arXiv preprint arXiv:2502.07150*, 2025.

[22] Joseph M Renes, Frédéric Dupuis, and Renato Renner, "Efficient polar coding of quantum information," *Physical Review Letters*, vol. 109, no. 5, pp. 050504, 2012.

[23] Mark M Wilde and Saikat Guha, "Polar codes for degradable quantum channels," *IEEE Transactions on Information Theory*, vol. 59, no. 7, pp. 4718–4729, 2013.

[24] Anqi Gong and Joseph M Renes, "Improved logical error rate via list decoding of quantum polar codes," in *2024 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2024, pp. 2496–2501.

[25] Xanthe Croot, Kasra Nowrouzi, Christopher Spitzer, Carmen G Almudever, Alexandre Blais, Malcolm Carroll, Jerry Chow, Daniel Friedman, Masao Tokunari, Edoardo Charbon, et al., "Enabling technologies for scalable superconducting quantum computing," *arXiv preprint arXiv:2512.15001*, 2025.

[26] Handy Kurniawan, Valentin Savin, Carmen G Almudéver, and Francisco García Herrero, "Implementing quantum polar codes in a superconducting processor: From state preparation to decoding," *IEEE Software*, 2025.

[27] IBM Quantum, "Compute resources — IBM Quantum Platform," 2026, Available: https://quantum.cloud.ibm.com/computers (Accessed Jan. 16, 2026).

[28] John Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, pp. 79, 2018.

[29] Krishnageetha Karuppasamy, Varun Puram, Stevens Johnson, and Johnson P Thomas, "A comprehensive review of quantum circuit optimization: Current trends and future directions," *Quantum Reports*, vol. 7, no. 1, pp. 2, 2025.

[30] Ashutosh Goswami, Mehdi Mhalla, and Valentin Savin, "Fault-tolerant preparation of quantum polar codes encoding one logical qubit," *Physical Review A*, vol. 108, no. 4, pp. 042605, 2023.

[31] Michael A Nielsen and Isaac L Chuang, *Quantum computation and quantum information*, Cambridge university press, 2010.

[32] Craig Gidney, "Stim: a fast stabilizer circuit simulator," *Quantum*, vol. 5, pp. 497, July 2021.

[33] Daniel Gottesman, *Stabilizer codes and quantum error correction*, California Institute of Technology, 1997.

[34] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland, "Surface codes: Towards practical large-scale quantum computation," *Physical Review A—Atomic, Molecular, and Optical Physics*, vol. 86, no. 3, pp. 032324, 2012.

[35] Dmitri Maslov, Gerhard W Dueck, D Michael Miller, and Camille Negrevergne, "Quantum circuit simplification and level compaction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 3, pp. 436–444, 2008.

[36] Nabila Abdessaied, Mathias Soeken, and Rolf Drechsler,

"Quantum circuit optimization by hadamard gate reduction," in *International Conference on Reversible Computation*. Springer, 2014, pp. 149–162.

[37] Thomas Fösel, Murphy Yuezhen Niu, Florian Marquardt, and Li Li, "Quantum circuit optimization with deep reinforcement learning," *arXiv preprint arXiv:2103.07585*, 2021.

[38] J Riu, J Nogué, G Vilaplana, A Garcia-Saez, and MP Estarellas, "Reinforcement learning based quantum circuit optimization via ZX-calculus (2024)," *arXiv preprint arXiv:2312.11597*, 2025.

[39] Michal Sedlák and Martin Plesch, "Towards optimization of quantum circuits," *Central European Journal of Physics*, vol. 6, no. 1, pp. 128–134, 2008.

[40] Igor L Markov and Yaoyun Shi, "Simulating quantum computation by contracting tensor networks," *SIAM Journal on Computing*, vol. 38, no. 3, pp. 963–981, 2008.

[41] Jessica Pointing, Oded Padon, Zhihao Jia, Henry Ma, Auguste Hirth, Jens Palsberg, and Alex Aiken, "Quanto: Optimizing quantum circuits with automatic generation of circuit identities," *Quantum Science and Technology*, vol. 9, no. 4, pp. 045009, 2024.

[42] Mansoor Niaz and William R Robinson, "Teaching algorithmic problem solving or conceptual understanding: Role of developmental level, mental capacity, and cognitive style," *Journal of Science Education and Technology*, vol. 2, no. 2, pp. 407–416, 1993.

[43] Joao F Ferreira, Alexandra Mendes, Alcino Cunha, Carlos Baquero, Paulo Silva, Luís Soares Barbosa, and José Nuno Oliveira, "Logic training through algorithmic problem solving," in *International Congress on Tools for Teaching Logic*. Springer, 2011, pp. 62–69.

[44] Yunseong Nam, Neil J Ross, Yuan Su, Andrew M Childs, and Dmitri Maslov, "Automated optimization of large quantum circuits with continuous parameters," *npj Quantum Information*, vol. 4, no. 1, pp. 23, 2018.

[45] Erdal Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[46] Ryan Mandelbaum, Antonio D Córcoles, and Jay Gambetta, "Ibm's big bet on the quantum-centric supercomputer: Recent advances point the way to useful classical-quantum hybrids," *IEEE Spectrum*, vol. 61, no. 9, pp. 24–33, 2024.

[47] Sergey Bravyi and Alexei Kitaev, "Universal quantum computation with ideal Clifford gates and noisy ancillas," *Physical Review A—Atomic, Molecular, and Optical Physics*, vol. 71, no. 2, pp. 022316, 2005.

[48] Christopher Chamberland and Michael E Beverland, "Flag fault-tolerant error correction with arbitrary distance codes," *Quantum*, vol. 2, pp. 53, 2018.

[49] Dhruv Bhatnagar, Matthew Steinberg, David Elkouss, Carmen G Almudever, and Sebastian Feld, "Low-depth flag-style syndrome extraction for small quantum error-correction codes," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2023, vol. 1, pp. 63–69.

[50] Younghun Kim, Hansol Kim, Jeongsoo Kang, Wonjae Choi, and Younghun Kwon, "Effectiveness of the syndrome extraction circuit with flag qubits on IBM quantum hardware," *Quantum*, vol. 9, pp. 1893, 2025.

[51] Matteo Pompili, Sophie LN Hermans, Simon Baier, Hans KC Beukers, Peter C Humphreys, Raymond N Schouten, Raymond FL Vermeulen, Marijn J Tiggelman, Laura dos Santos Martins, Bas Dirkse, et al., "Realization of a multinode quantum network of remote solid-state qubits," *Science*, vol. 372, no. 6539, pp. 259–264, 2021.

[52] Wonho Jang, Koji Terashi, Masahiko Saito, Christian W Bauer, Benjamin Nachman, Yutaro Iiyama, Ryunosuke Okubo, and Ryu Sawada, "Initial-state dependent optimization of controlled gate operations with quantum computer," *Quantum*, vol. 6, pp. 798, 2022.

[53] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D Nation, Lev S Bishop, Andrew W Cross, et al., "Quantum computing with Qiskit," *arXiv preprint arXiv:2405.08810*, 2024.

Scaling Quantum Circuit Simulation: Optimizations for the FTDD Framework

Vicente López[1] , Jose M. Badia[1], Maribel Castillo[1]

Resumen— **Classical simulation remains essential for the development of quantum algorithms; however, traditional matrix-based methods struggle with scalability. While the Fast Tensor Decision Diagram (FTDD) framework offers a hybrid approach by combining the strengths of Tensor Networks and Decision Diagrams (DDs) for memory-efficient simulation, its core operations are often bottlenecked by memory management structures. Our primary motivation is to address the high computational and spatial cost associated with the hash tables, as well as to simplify the complexity of the simulation workflow. This work introduces systematic, low-level optimizations to the framework. Our main contribution is the enhancement of the memory management layer for the internal tables. We implemented a node reuse mechanism and designed a controlled strategy for hash table growth. These improvements minimize the memory footprint and reduce the costly hashing overhead inherent in DD-based simulators. We performed a thorough experimental analysis using a diverse set of quantum circuits, which were characterized by well-established metrics. Experimental results demonstrate that our enhancements improve the tool's performance and scalability. Notably, these advancements enable the simulation of complex circuits, such as Quantum Fourier Transform, with hundreds of qubits, on a standard server with 2-Tb of memory.**

Palabras clave— **Quantum Computing Simulation, Decision Diagrams, Tensor Network, Statevector Simulation.**

I. Introduction

THE field of Quantum Computing (QC) has attracted attention due to its transformative potential in various fields [1]. However, the capabilities of current quantum devices are limited by the constraints of the Noisy Intermediate-Scale Quantum (NISQ) era [2]. That includes a restricted number of qubits and susceptibility to noise. These factors hinder the reliable execution of advanced quantum algorithms. Consequently, classical simulators are indispensable for designing, validating and optimising quantum algorithms. However, traditional simulation approaches based on matrix and vector multiplications face severe scalability challenges due to the exponential increase in computational and memory requirements [3]. It is important to emphasise that the fundamental factor limiting the size of quantum circuits that can be simulated on classical hardware is memory rather than time or the raw computational capabilities of modern supercomputers.

To address the limitations of circuit simulation, several methodologies have been introduced, including full wavefunction evolution [4] and Tensor Network (TN) [5]. The last one has demonstrated remarkable efficacy in simulating random quantum circuits [6]. However, their efficiency is strongly dependant on the order the tensors are contracted. Finding the optimal order is an NP-hard problem [7], so it is important to develop heuristics to efficiently approximate the optimal order.

Alternative ways of representing, storing and manipulating states and operations have been proposed, such as Decision Diagrams (DDs) [8]. This type of structure was designed to enhance the efficiency of quantum operations by minimising memory usage. This is achieved by exploiting mathematical properties to reduce data storage and avoiding their repetition of equivalent operations [9]. The prevailing state-of-the-art tool is known as DDSIM [10] and has shown to be competitive with matrices.

In this study, we use FTDD [11], a tool that combines TNs and DDs in a unique way, making use of the strengths of each. This hybrid approach establishes FTDD as a highly promising framework with the potential to rival well-established simulators such as DDSIM (DD-based) and GTN [12] (matrix-based). This DD-based tool relies on two hash tables (`UniqueTable` and `ComputeTable`) to take advantage of their benefits. While the implementation of these structures is crucial for the sequential performance of the tool, several studies indicate that these global tables often act as a significant barrier to parallelization, limiting the scalability of DD-based structures in high-performance environments [13]. This work aims to optimize the implementation of these tables to enhance the tool's performance by enabling the simulation of larger circuits and reducing computational time [14].

We extend the functionality of the FTDD framework by introducing a unified simulation interface that standardises the execution workflow and abstracts internal complexities from the user. In addition, the management of the `UniqueTable` and `ComputeTable` has been substantially improved through two core enhancements: preventing the premature deletion of nodes that remain in use, and enabling the safe reuse of nodes marked as dead by the garbage collector, thereby reducing memory overhead. We further introduce a heuristic that regulates the size of both tables to mitigate uncontrolled growth as circuit size increases. Beyond these architectural improvements, we conduct an extensive experimental evaluation using a diverse benchmark suite comprising some of the most widely studied circuits in the literature, such as RQC, QFT, QPE, QNN, QWalk, GHZ, and Graph states. This allows us to analyse the behaviour of the tool across markedly different circuit families. Our results de-

[1]Dpt. de Ing. y Ciencia de Comp., Universitat Jaume I, Spain, e-mail: {voliva,badia,castillo}@uji.es

monstrate that FTDD can simulate a broad range of circuits at sizes significantly larger than those achievable with matrix-based tools such as GTN. In particular, FTDD is capable of simulating QFT circuits with more than 120 qubits on a machine with 2 TB of memory, clearly surpassing the limits reported for tools such as DDSIM and GTN.

The paper is structured as follows: Section II briefly reviews some related work. Section III summaries the theoretical background of our work, including basic aspects of quantum computation, tensor networks, decision diagrams, and hash tables. Section IV provides definitions of metrics that will be used to characterise and select the set of quantum circuits to be used. In section V, we detail the improvements that have been made to enhance the tool's performance. In section VI, we provide a detailed description of the experiments conducted and discuss the results. Finally, Section VII presents the conclusions and suggests directions for future research.

II. Related work

There exists a wide variety of quantum simulators that operate both sequentially and in parallel [15], and most of them are based on the evolution of the full quantum state using matrices. Nevertheless, the exponential increase in spatial and temporal costs associated with simulation means that the number of qubits is limited to a few tens, even in the most powerful supercomputer available [16].

The use of TN for simulating quantum circuits was firstly proposed by Markov and Shi [17], which lead to the development of quantum simulators based on this idea, like QTensor [18]. But the performance of this kind of simulators strongly depends of the order in which the contractions are made. One of the most widely used tools for this purpose is `cotengra` [19], which besides tree decompositions, also implements more sophisticated optimization techniques such as heuristic cost models, hyper-graph partitioning, and stochastic or evolutionary search methods. `cotengra` can yield contraction orders close to the optimal when enough time is given. The problem with these simulators based on matrix representation of the information is that, even with the final tensors, the dimensions of the matrices grows exponentially with the number of qubits, which is a major constraint, especially in terms of space cost. To assess this problem, other representations of the information have been proposed.

DDs were adopted to represent data and operations for various applications, due to their ability to reduce both temporal and spatial costs. In [20], the authors provide the first description of how Binary Decision Diagrams (BDDs) should be implemented. This article laid the foundations for the development of various types of DDs with diverse applications [21,22]. In [23] the first attempt can be found to use this representation to simulate quantum circuits. Since then, many tools have been developed using this type of diagrams. The most well-known library is MQT [10], which is utilised for both simulating and verifying quantum circuits. This tool uses Quantum Multiple-Valued Decision Diagram (QMDD) as their representation, a variant of the DDs designed specifically for quantum circuits [10]. QMDD partitions a transformation matrix into four sub-matrices of equal size. These sub-matrices are then partitioned in a similar way until the matrix's dimension is 1×1. This method of representing matrices is somewhat rigid as it does not allow for a different order of variables. To address this limitation, Tensor Decision Diagrams (TDDs) were introduced to combine the best features of DDs and TNs [24].

III. Background

This section provides a brief overview of the fundamental theoretical aspects. For an in-depth analysis of QC, readers are directed to consult [25] or [26].

A. Quantum Circuits as TNs

Similar to the classical bit, the basic unit of information in QC is the qubit, which can be written as a linear combination of two basic states as $\alpha|0\rangle + \beta|1\rangle$ with $\alpha, \beta \in \mathbb{C}$ [27]. A quantum system is formed by a set of n qubits, $\phi_1, \cdots, \phi_n$. This constitutes a Hilbert space of dimension 2^n, whose basic states are in $\{0,1\}^n$. Therefore, the state of a quantum system $|\psi\rangle$ can be described as:

$$|\psi\rangle = \sum_{i \in \{0,1\}^n} \alpha_i |i\rangle \text{ with } \forall i \in \{0,1\}^n, \alpha_i \in \mathbb{C}, \quad (1)$$

where the probability of measuring $|i\rangle$ is given by $|\alpha_i|^2$. Quantum systems are manipulated through quantum operators that are unitary matrices of dimension $2^n \times 2^n$. Typically, these operators only act on a subset of the system's qubits. Quantum circuits are commonly used to represent the successive application of quantum operators $g_1, \cdots, g_k$. Quantum algorithms can be represented by a quantum circuit, and have an associate unitary operator that represents their complete functionality.

Tensors are a natural generalisation of matrices. A tensor of rank r is defined as an element $d_1 \times \cdots \times d_r$ existing in the space $\mathbb{C}^{d_1,\cdots,d_r}$. A complex value can be seen as a tensor of rank 0, while a vector is a tensor of rank 1. Similarly, a matrix of dimension $n \times m$ is a tensor of rank 2. Tensors have a defined set of operations, including the tensor product, trace, contraction, and partitioning [28]. One of the most significant operations is the contraction. Contraction of two tensors $R_{x,z}$ and $S_{y,z}$ is a tensor obtained by summing up over the shared indices, defined as follows:

$$T_{x,y} = \sum_{z \in \{0,1\}} R_{x,z} \cdot S_{y,z} \quad (2)$$

In general, the successive contraction between pairs of tensors of a network allows to obtain a single tensor representing its full functionality. The quantum state $|\psi\rangle$ can also be represented as a vector

$|\psi\rangle = [\alpha_0, \alpha_1]^T$, where $\alpha_0, \alpha_1 \in \mathbb{C}$. Therefore, it can be represented as a rank 1 tensor T_q. Similarly, the operators associated with the gates applied to a single qubit can be represented as rank 2 tensors T_{q_1,q_2}. In general, a gate acting on n qubits can be represented as a tensor of rank $2n$.

A quantum circuit can be modeled as a TN, where each gate is represented as a tensor. Consequently, since the behavior of a quantum circuit can be simulated using the matrix representing its functionality, it can similarly be simulated using its tensor representation. This implies that simulating a quantum circuit is equivalent to contracting its associated TN. This principle underpins the approach proposed by Markov and Shi to reduce the cost of simulating quantum circuits [17].

B. Decision Diagrams

Quantum circuits are usually implemented by using matrices, whose main advantage is the availability of efficient algorithms for matrix multiplication. But this is not the only type of structure that we can use. DDs are acyclic undirected graphs that represent the gates that compose the circuits and the unitary matrices used during its processing (simulation, verification, etc).

Two main data structures are usually employed to efficiently store the DDs. On the one hand, the `UniqueTable` stores the information about the graph nodes and on the other hand, the `ComputeTable` stores the operations as they are performed. This implementation is advantageous due to the operations on tree-like DDs are usually performed recursively based on the subtrees of each node [24]. If an operation between subtrees has already been performed, it is retrieved directly from the compute table.

Tensors can also be represented as a DDs using TDD [24], which allows the benefits of both techniques in the handling of quantum circuits. In order to achieve this objective, a number of methods have been developed that are essential for DDs to be able to work with TNs, including the contraction operation.

TDDs use nodes to represent the indices of TN, with each one having two successors linked by edges. One of the successors is associated with the value 0 of the node, while the other successor is associated with the value 1. Each edge has a weight indicating the value by which the TDD must be multiplied. In the graphical representation, the absence of a weight on the edge indicates that its weight is 1. The TDDs introduced in [24] were implemented in Python. The improved C++ version, designated Fast Tensor Decision Diagram (FTDD), will be employed [11]. In this new version, the authors make improvements to the TDD which includes Edge-centric Data Structures for TDD, employment of Key BDD Optimizations [20] or the execution of Near-Optimal Contraction Order (`cotengra` in this case), with which the authors can obtain speedups up to 175x respect to the Python version.

C. Hash Tables

A hash table is a data structure designed to store and retrieve (key, value) pairs with an average-time complexity of O(1) for lookup, insertion, and deletion operations [29]. They have become a ubiquitous component in high-performance software, including compilers, interpreters, symbolic manipulation tools, and, most notably, DD packages used in formal verification and symbolic simulation. Its efficiency relies on the use of a hash function, which maps an arbitrary key to an index in an underlying array (bucket), combined with a collision-resolution strategy to handle cases where multiple keys map to the same bucket. This design enables significantly faster access times than classical search structures such as linked lists. FTDDs are derived from BDD, and the general architecture used to implement BDD use two hash tables in his core [20], which fulfill distinct but complementary roles:

1. `UniqueTable`: enforces the canonicity of the decision diagram. Typically indexed by the tuple describing a node (key, childs) and ensures that only one node with a given structure exists. This mechanism is critical for reducing the storage cost.
2. `ComputeTable`: functions as a memorization cache for intermediate computation results and stores entries indexed by the operation and its operands. This avoids redundant work and is essential for performance when executing complex or deeply nested operations on decision diagrams.

IV. Benchmarks and Metrics

In order to evaluate the performance of the simulators, it is imperative to employ a set of metrics and quantum circuits that are both relevant and well-established. As was the case in [30], the present study will rely on a list of metrics that characterise the topological nature of static quantum circuits. These metrics will be used to select quantum circuits that differ in their topologies. We will use them to select a total of eight circuits from MQTBench [31]. This yields a test suite that is representative of practical quantum-computing applications.

First, we use metrics to characterise the structural properties of the circuits analysed in this work. We consider: the *Effort* (E) metric [32], which captures the structural complexity of the circuit; the *Entanglement Ratio* (ER), *Critical Depth* (Cd), *Parallelism* (Pa), and *Program Communication* (Pc) metrics from SupermarQ [33], which quantify the density of two-qubit interactions, serial bottlenecks, parallel structure, and communication patterns among qubits, respectively; and the *Entanglement Variance* (EV) from QASMBench [34], which measures how uniformly entanglement is distributed.

The benchmark suite employed in this study comprises a broad and heterogeneous collection of circuits, selected to span a wide variety of structural

behaviours according to the metrics above. These include several of the most representative and widely studied circuits in the quantum-simulation literature, such as QFT, QPE, RQC, GHZ, QNN, QWalk, RA and Graph. Together, they cover many of the key algorithmic families used in practice and in benchmarking frameworks. This diversity is essential for assessing the robustness of the simulator across different topological characteristics, rather than for a narrow or specially favourable subset of circuits. Figure 1 shows the evolution of the effort with the number of qubits for each circuit.

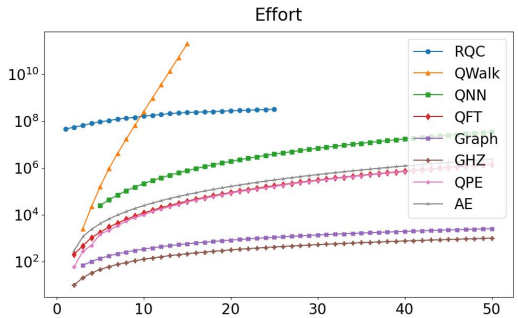


Fig. 1: Evolution of the Effort metric with respect to the number of qubits for each of the circuits.

V. FTDD optimization

In a previous work [35], FTDDs were tested across a broad variety of quantum circuits and contraction–order selection methods. That revealed several implementation issues, along with functional and performance limitations that affected the robustness and scalability of the tool. The present work analyzes the results of an improved version of FTDD that addresses these limitations, correcting identified errors, and improving performance, particularly with respect to the memory cost of TN contractions.

In this section, we describe the implementation details of the revised version. Firstly, several previously undetected issues had hindered the correct construction and manipulation of the underlying TN components. In particular, recurrent errors in the detection and propagation of open indices led to malformed intermediate representations, compromising the reliability. These issues have now been corrected: open indices are consistently recognised, validated, and updated throughout all stages of the FTDD pipeline, ensuring stable behaviour.

We also have introduced a new high-level simulation function designed to streamline and standardize the execution of quantum-circuit simulations within the FTDD framework. Previously, users were required to manually invoke a sequence of low-level operations (ranging from the initialization of the tool to circuit preprocessing and TN construction to contract the network), making the simulation workflow prone to inconsistencies and difficult to reproduce. The newly implemented function encapsulates this complexity by enforcing a canonical order of operations and abstracting internal procedural details. This not only reduces user-side overhead but also ensures methodological uniformity. As a result, FTDD now offers a clearer and more robust user interface, further strengthening its value as a practical simulation tool.

Additionally, we revised the behaviour of the two core data structures that underpin the FTDD representation, namely the `ComputeTable` and the `UniqueTable`, in order to improve memory usage and reduce unnecessary resource consumption. First, the `ComputeTable` has been updated to ensure that nodes currently referenced during intermediate computations cannot be prematurely deleted. In the previous implementation, the garbage collector could remove nodes that were still in active use, potentially leading to invalid pointer accesses or inconsistencies in the DD layer. The updated mechanism introduces explicit reference tracking, preventing the garbage collector from reclaiming nodes whose lifecycle is tied to active entries.

Second, modifications to `UniqueTable` now allow nodes that have been marked as dead to be safely reused. Earlier versions relied on a one-use lifecycle for `UniqueTable` entries, resulting in unnecessary memory pressure when large numbers of short-lived nodes were created during highly entangled simulations. The new reuse strategy enables the system to repurpose entries that are no longer semantically active, thereby reducing allocations, and improving the overall memory footprint of the simulator. Together, these enhancements strengthen the efficiency and reliability of the FTDD memory-management model, particularly for large-scale simulations where node growth is a critical performance factor.

Finally, we redesigned the sizing policy of the `UniqueTable` and `ComputeTable` to address the exponential memory growth that previously occurred. In the original FTDD version, allocated memory size of the `UniqueTable` depended exponentially of the number of qubits, and the `ComputeTable` was setted into a fixed value of 2^{15} buckets. This behaviour often led to excessive and unpredictable memory usage, making simulations infeasible for circuits of even moderate size. To overcome this limitation, we now enforce a constant memory footprint for both tables by constraining their combined capacity to a fixed and small fraction of the available memory.

The hash function employed is a 32-bit FNV scheme. Prior work has shown that table sizes of the form $2^x - 1$ yield particularly robust distribution properties for this type of hash function [36]. For this reason, we introduce a parameter N that determines the number of buckets assigned to each table: the `UniqueTable` is allocated with $2^N - 1$ buckets, while the `ComputeTable` is assigned $2^{N-1} - 1$ buckets. The value of N is chosen so that the combined memory footprint of all tables accounts for approximately 10 % of the total system memory. The decision to adopt this sizing strategy arises from extensive empirical analysis indicating that table-growth patterns vary significantly across circuit families. Consequently, no single static configuration provides con-

sistently optimal behaviour for all benchmarks. In particular, increasing the size of the `ComputeTable` is essential for storing a sufficiently large number of intermediate operations, thereby enabling the detection of repeated computations. At the same time, the table must remain small enough to ensure that node removal can occur effectively and that memory usage does not become intractable. The selected configuration provides a practical balance between these competing requirements.

By tying the table sizes to a fixed portion of system memory, FTDD adapts automatically to the available memory while ensuring controlled memory usage. This dynamic, resource-aware allocation strategy markedly improves scalability and prevents the runaway table expansion that previously hindered large-scale simulations. Although this approach introduces a minor drawback—namely, that the tables retain a fixed and relatively large size even for small circuits, thereby reserving more memory than strictly necessary, its practical impact is limited. Small circuits do not come close to exhausting available memory, and the allocation of larger tables does not noticeably increase contraction time when memory-reservation overhead is taken into account. More importantly, the primary aim of this work is to ensure spatial efficiency for large-scale simulations, where memory is the dominant bottleneck. In this context, the advantages of the new sizing policy decisively outweigh the overhead incurred for small circuits, and the overall behaviour of FTDD becomes significantly more robust and scalable.

VI. Experimental results

This section presents the methodology used to conduct the experiments, together with the resulting data and its analysis. Our objective is to compare the spatial and temporal costs of our enhanced FTDD implementation with those of the previous version, as well as with GTN, a well-established framework that realizes TN through matrix representations.

A. Experimental Environment and Methodology

All experiments have been run on a single node including 2 Xeon 6418H with 24 cores each, 2Tb of RAM and 4 GPU H100-80Gb. The operating system used was Linux 5.4.0-72-generic 80-Ubuntu. Two types of experiments were conducted. First, a small set of circuits was used to evaluate the improvements of the latest FTDD version (section VI-B). Subsequently, all circuits described in Section IV were employed to perform a comparative analysis between the FTDD and GTN tools (section VI-C).

Unless noted otherwise, all experiments used the same workflow. All simulations were performed using statevector evaluation, meaning that the input was fixed (in our case, to state $|0\rangle$) and the output remained open. Two contraction plans were used: the plan obtained with the `sequential` method (selected for its simplicity and because it is known to be among the least effective orders for statevector simulation with DDs) and the plan obtained with the `cotengra` method, one of the best-known optimization strategies. No prior pre-processing was applied. `cotengra` had one hour to compute the order with the *minimize* parameter set to *combo* with 56 threads. All simulations have a time constraint of one hour to perform the simulation. They started with circuit with the smallest number of qubits and continued with larger ones until the tool could no longer complete the simulation due to memory or time constraints.

B. Evaluation of Memory Optimization Strategies

In this section, we evaluate the efficiency of the improvements with respect to the original version. We employ the `sequential` method in conjunction with three representative circuits that highlight the variability. Regarding the versions under consideration, we denote the original FTDD implementation as `original` , the one that incorporates only the mechanism for correctly deleting nodes and reusing it as `Node-Reuse` (`NR`) , and the version with all modifications as `Node-Reuse and Table-Resize` (`NRTR`) . For each version, we evaluate both execution time and memory consumption, which was measured using the python function *psutil.virtual_memory*.

An examination of the memory usage presented in Figure 2 reveals two clear patterns. First, the `NR` configuration consistently exhibits lower memory consumption than the `original` implementation in all scenarios, highlighting the benefits of explicitly deleting and reusing nodes correctly. The second pattern concerns the comparatively high memory usage observed for the `NRTR` version across all circuits at smaller sizes. This behavior arises from its use of fixed-size tables: regardless of how few nodes are stored, the memory allocated for these tables imposes a substantial lower bound on overall memory consumption. As the number of qubits increases, this difference gradually decreases, as the table sizes required by the other implementations progressively approach those of the `NRTR` version. It is also important to note that not all circuits reach the same number of qubits (particularly in the QNN and RQC cases) because each simulation was executed only up to the point at which either the one-hour time limit or the available system memory (2 Tb) was exceeded.

The advantages of fixing the table size become especially evident in the case of the QFT circuit. In the figure 2, both the `original` and `NR` configurations exhibit exponential growth, whereas the `NRTR` version shows no increase in memory consumption. Starting at approximately 25 qubits, the number of nodes required by the simulation grows only marginally; the exponential behaviour observed in the other versions is therefore attributable to the growth of the tables themselves rather than to the memory needed to represent intermediate contraction results. Our experiments were terminated at 120 qubits due to the lack of larger benchmark circuits, but the memory profile of the `NRTR` version shows no indication of significant growth beyond this point.

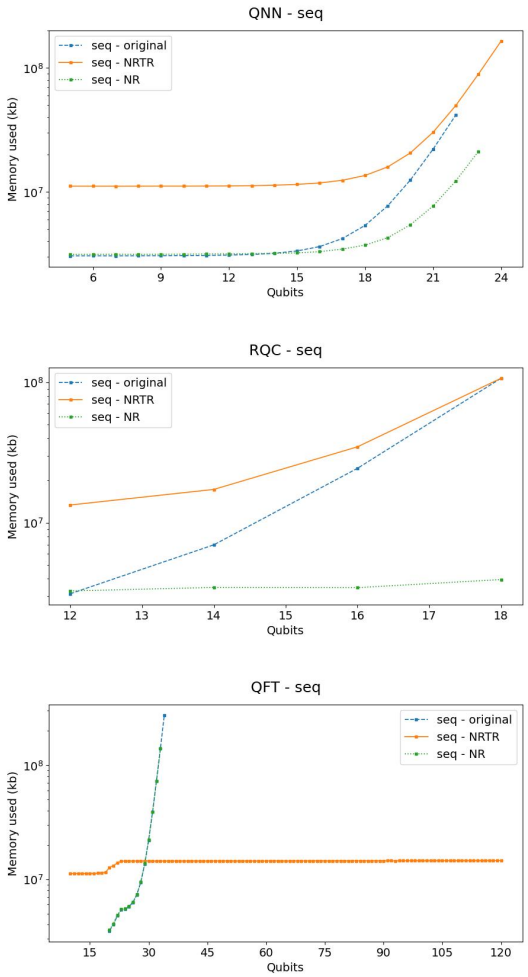


Fig. 2: Evolution of the memory usage of the original, `NR` and `NRTR` versions of the FTDD with the `sequential` method.

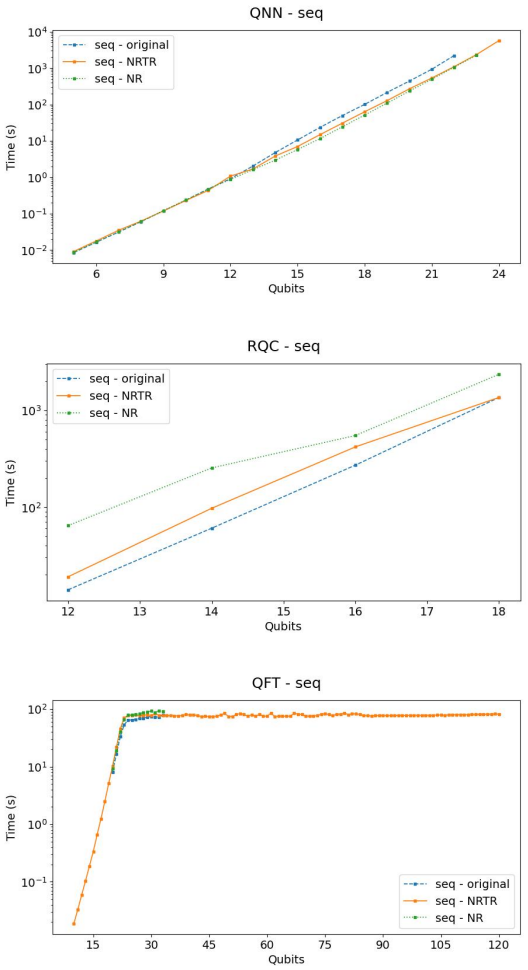


Fig. 3: Evolution of the Contraction time of the original, `NR` and `NRTR` versions of the FTDD with the `sequential` method.

An analysis of the execution times in logarithmic scale presented in Figure 3, reveals a consistent pattern. Notably, the behaviour of the algorithms varies significantly depending on the type of circuit, which also influences the relative performance of the three versions evaluated. In general, the `original` version exhibits lower runtime than the `NR` implementation. This difference arises from the additional overhead associated with maintaining the status of each node. The `NRTR` version almost always shows better performance than the `NR` version. This improvement is attributable to the increased size of the `ComputeTable` structures, which, being larger, can store more intermediate results and thereby reduce the number of repeated operations performed. However, this benefit is not sufficient to compensate fully for the additional bookkeeping overhead, and the execution times of the `NRTR` version remain, in most cases, higher than those of the `original` version. Nevertheless, the average observed difference in runtime is below 5 %.

C. Comparing DD and Matrix-based simulation

The objective of this section is to compare the performance of FTDDs (using the `NRTR` version), in terms of both execution time and memory consumption, with that of a well-established tool for matrix-based TN simulations, GTN. All circuits were executed following the procedure described above, with the exception of QPE, QNN and QWalk, for which a longer time limit of 10 hours was imposed. This adjustment was necessary because these circuits exhibit significantly higher runtime compared to the others, while both GTN and FTDD showed modest memory usage under the one-hour limit. Extending the time window therefore enabled a more informative analysis of their memory behaviour.

In Figure 4, the memory usage of both tools is presented, in which can be seen a large variety of behaviour across the tools and contractions methods. It is important to highlight that FTDD exhibits relatively high memory usage even for small circuits. Recall that this behaviour originates from the fixed size of its internal tables, which imposes a constant minimum memory overhead, as previously noted. Based on the observed performance, two groups of circuits can be identified based on their performance. The first group corresponds to the circuits QFT, Graph, GHZ and QPE. In this group, FTDD's memory consumption avoids the exponential growth observed for GTN. FTDD's memory usage appears to stabilise for all circuits in this group except QPE, whose memory behaviour is notably irregular as the number of qubits increases.

The second group includes the RQC, QNN,

QWalk, and RA circuits, within which two distinct cases arise. For QWalk, the limiting factor for FTDD was execution time, whereas for GTN it was memory usage. Thus, although GTN may perform better for small qubit counts, FTDD would be able to simulate larger circuits in the absence of time constraints. In contrast, for RQC, QNN and RA, GTN consistently achieves better memory performance than FTDD.

In Figure 5, the execution times of each circuit across the different tools and methods are shown. As anticipated from the diversity of circuits selected through the topological metrics, the behaviour of the tools varies substantially across the circuits, making it difficult to identify large groups of circuits with similar performance profiles. Nevertheless, two distinct groups emerge based on the relative performance of FTDD compared to GTN. The first group includes the QFT, Graph, and GHZ circuits, which is almost the same first group as in the memory analysis, with the exception of QPE. In this group FTDD does not exhibit exponential growth in simulation time. For QFT, the execution time grows exponentially up to 23 qubits and then stabilizes around 80 seconds, remaining nearly constant as the number of qubits increases. The time does not stabilizes for Graph and GHZ. In this group, FTDD clearly outperforms GTN, which cannot simulate circuits beyond 40 qubits due to insufficient memory, whereas FTDD handles the maximum number of qubits available in the MQT test suite.

The second group consists of the RQC, QNN, QWalk, QPE, and RA circuits. In this group, both FTDD and GTN exhibit exponential growth in execution time. Within it, three cases can be identified. The first includes the QNN and RA circuits, for which the increase in execution time is exponential and stable for both tools, and where the best ordering for GTN is able to simulate consistently larger circuits than the best ordering for FTDD. The second case corresponds to the QPE and QWalk circuits, where both tools show comparable performance, reaching approximately the same number of qubits. Finally, the third case comprises the RQC circuits, for which execution times increase in a similar way than QWalk and QPE, but where GTN achieves a better performance than FTDD.

Finally, it is worth examining the influence of the two contraction-ordering strategies employed throughout the experiments. The behaviour of the `sequential` and `cotengra` ordering differs markedly across both tools and circuit families. In general, `cotengra` could not simulate circuits as big as the `sequential` method, due to the lack of memory to calculate the path. For instance, with FTDD simulating QNN, `cotengra` gives best results but stops at 16 qubits while `sequential` can simulate up to 24 qubits. For FTDD, `cotengra` often yields faster simulations for a fixed circuit size; however, this tendency is far from uniform. In circuits such as QFT, RA, or QPE, the `sequential` method achieves lower execution times for GTN, which may be partly explained by the limited one-hour time budget available for `cotengra` to compute the contraction plan, but also could be explained for `cotengra` not optimizing the time, but instead optimizing both memory and time. In contrast, `cotengra` with FTDD performs better than `sequential` in general in the cases where there is enough memory to get a plan. That means that `cotengra` is giving a path that can effectively reduce the size of the DDs used during the contraction process.

Considering the limiting factors of FTDD during simulations, two groups can be distinguished. The first includes the RQC, QNN, RA, QPE, and QWalk circuits, for which execution time was the limiting factor. Within this group, two cases arise: for QPE and QWalk, FTDD could reach larger qubit counts than GTN in the absence of time constraints, whereas for RQC, RA, and QNN, GTN outperformed FTDD in both time and memory. The second group consists of the QFT, GHZ, and Graph circuits, where neither time nor memory limited FTDD; rather, the constraint was the maximum number of qubits available in the dataset we used. In this group, FTDD achieves substantially better performance than GTN in both metrics, and simulations could be extended to larger qubit counts if they were available.

We attempted to establish a correlation between the previously defined metrics and the computational performance of the simulators. However, our analysis reveals that none of these metrics consistently correlate with the runtime behavior exhibited by DD-based structures, contrasting with the results observed for matrix-based methods. This discrepancy arises because the complexity of DDs is not solely a function of the circuit's topological structure or the treewidth of a given contraction order. Instead, the efficiency of DD-based simulation is heavily dependent on the algebraic regularity and redundancy inherent in the matrices encountered throughout the contraction process—features that topological metrics fail to capture. Consequently, while structural metrics provide a reliable proxy for the workload in traditional simulators, they prove insufficient for predicting the more volatile performance profile of DD-based frameworks. An example can be seen in Figure 6a, which provides a summary of the performance of the GTN simulator using the `sequential` ordering. As can be observed, when the circuits are ordered according to their execution time, the resulting ordering matches exactly the one induced by the *effort* metric shown previously in Figure 1. This indicates that *effort* serves as a reliable predictor of GTN's runtime behavior. However, this correspondence does not hold for FTDD, as illustrated in Figure 6a. The performance of FTDD depends strongly on how effectively the structure of the underlying matrices can be exploited, resulting in a more irregular and less predictable runtime profile. Consequently, the *effort* metric fails to capture the dominant factors governing FTDD's performance and therefore cannot be used to predict its behavior.

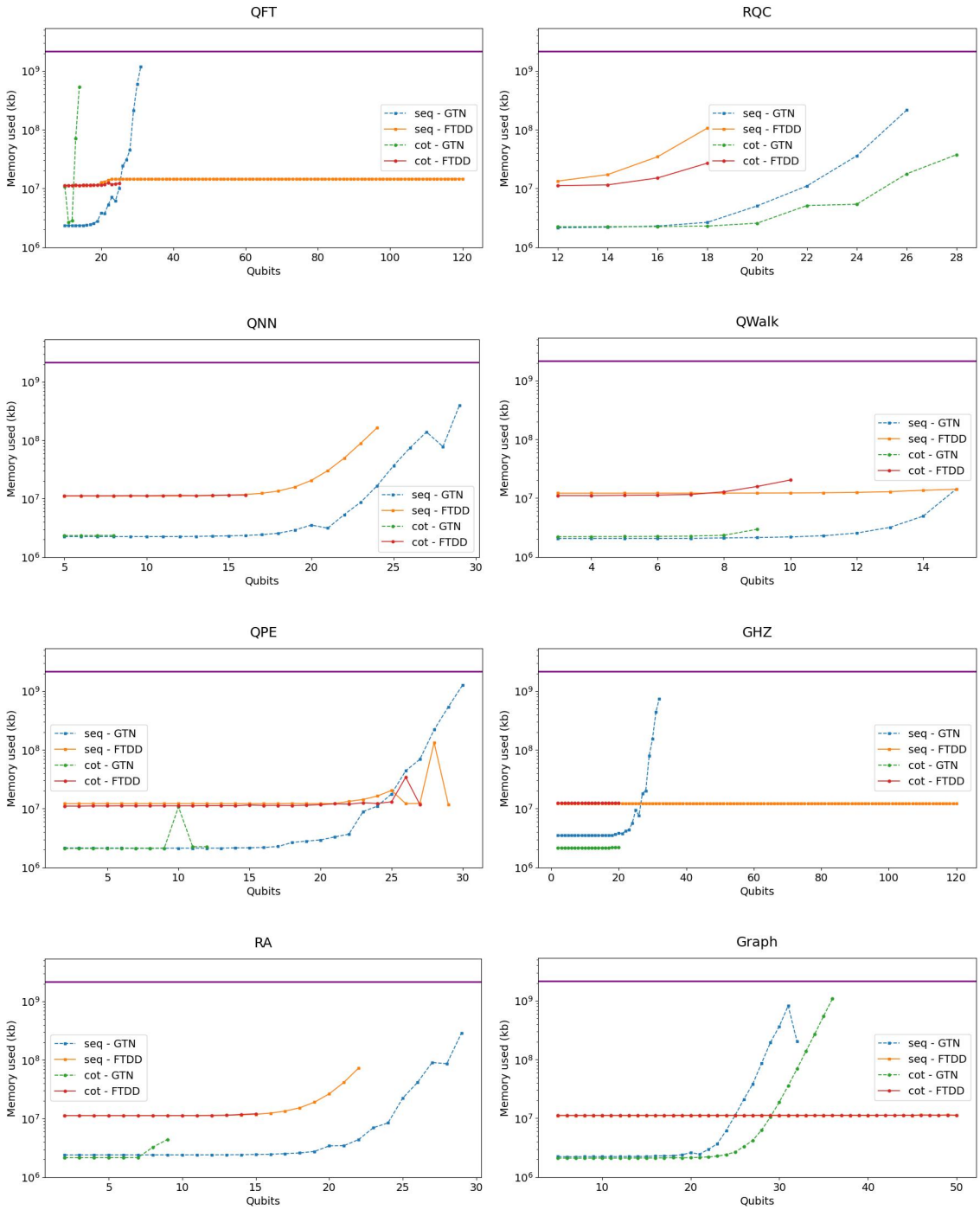


Fig. 4: Comparative between the tools GTN and FTDD and the contraction orders `sequential` and `cotengra` of the evolution of the memory used in the simulations in logarithmic scale measure in Kb respect the number of qubits of each circuit.

VII. Conclusions

This work has introduced a set of enhancements to the FTDD simulation tool, targeting its practical performance. First, we proposed a unified simulation interface that standardizes the execution workflow, thus improving usability and reproducibility. Second, we redesigned the internal `UniqueTable` and `ComputeTable` structures, preventing premature node deletion, enabling the safe reuse of nodes collected as dead, and introducing a heuristic mechanism that regulates table growth. These achieve more efficient memory utilization and substantially reduce memory overhead in large-scale simulations.

Our extensive experimental evaluation demonstrates the effectiveness of these improvements. The enhanced FTDD implementation exhibits a markedly superior memory profile compared with the original version, and in several circuit families—including QFT, GHZ, and Graph states—it achieves stable memory usage even as qubit counts increase. As a consequence, FTDD is able to simulate circuits with more than 120 qubits on a 2 TB machine, exceeding the capabilities of the matrix-based GTN simulator. These results position FTDD as a competitive alternative for large-scale quantum-circuit simulation, particularly in regimes where traditional tensor-network

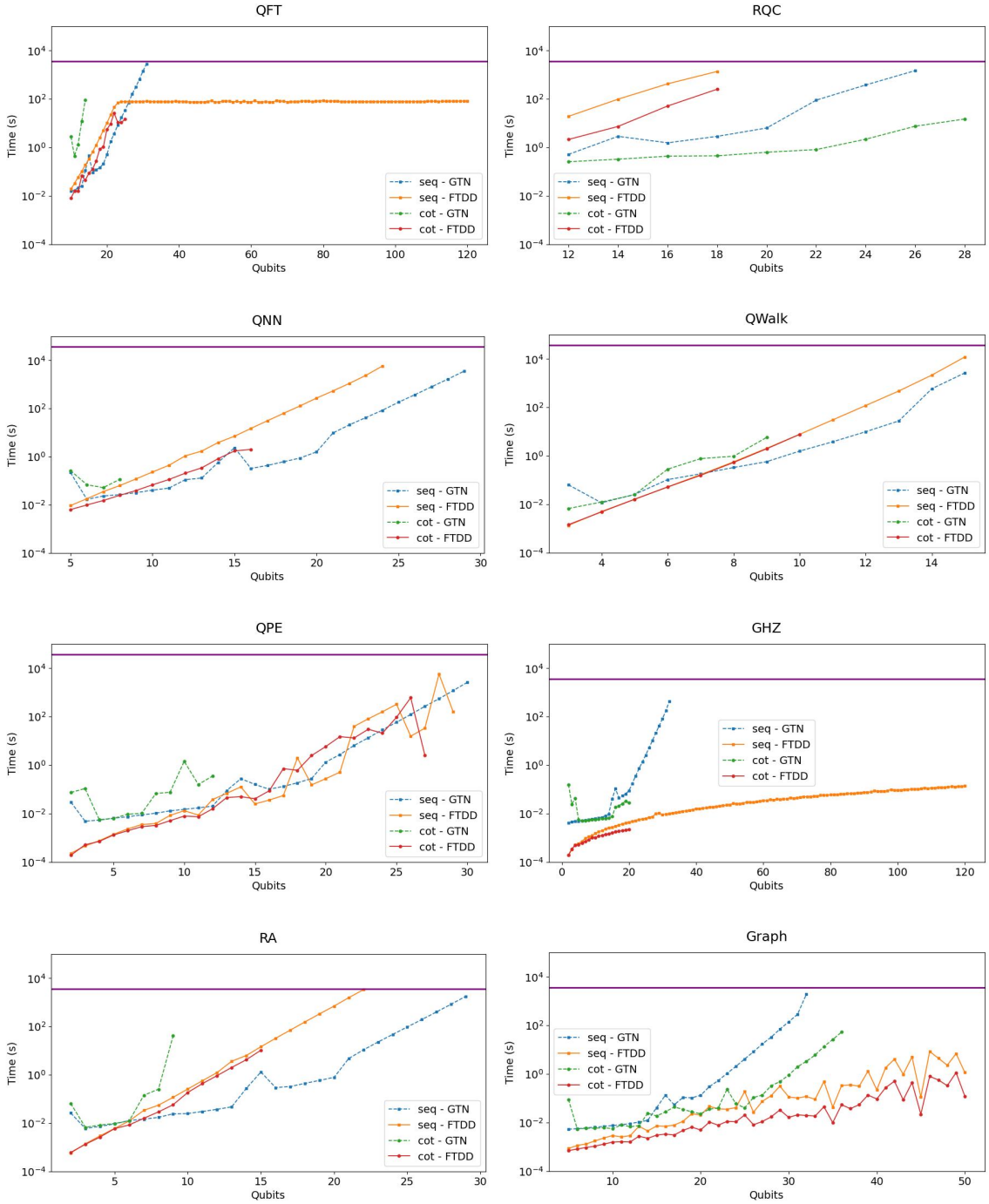


Fig. 5: Comparative between the tools GTN and FTDD and the contraction orders `sequential` and `cotengra` of the evolution of the time used in the simulations in logarithmic scale measure in second respect the number of qubits of each circuit.

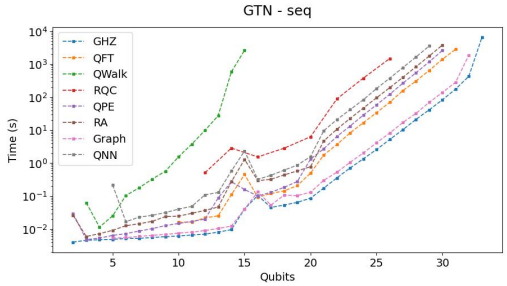
techniques fail due to exponential memory growth.

The comparative analysis further shows that FTDD and GTN exhibit complementary performance characteristics: FTDD excels for circuits with strong structural regularities, while GTN remains advantageous for circuits that generate highly irregular contraction patterns. Additionally, our study confirms that the *effort* metric provides an accurate predictor of GTN's runtime but does not capture the factors that dominate FTDD's behavior. This observation highlights the need for new metrics specifically tailored to DD simulation approaches.

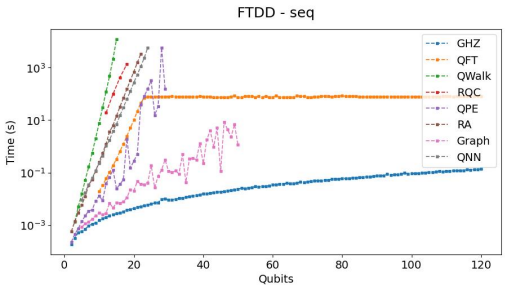
Future work will explore advanced contraction-planning strategies tailored to DD-based structures, extend the memory-management techniques to distributed memory and develop predictive metrics capable of characterising hybrid TN-DD behaviour. These directions are expected to further expand the applicability of FTDD.

Agradecimientos

This research was funded by the project PID2023-146569NB-C22 supported by MICIU/AEI/10.13039/501100011033 and ERDF/UE. Author V.L-O. is supported by grant ACIF/2021/260 (GVA and European Social Fund).



(a) Evolution of the time in seconds with logarithmic scale respect the number of qubits of the circuit for simulations done with GTN for each circuit group.



(b) Evolution of the time in seconds with logarithmic scale respect the number of qubits of the circuit for simulations done with FTDD for each circuit group.

Referencias

[1] X. Yuan, "A quantum-computing advantage for chemistry," *Science*, vol. 369, no. 6507, pp. 1054–1055, 2020.

[2] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[3] C. Huang, F. Zhang, M. Newman, X. Ni *et al.*, "Efficient parallelization of tensor network contraction for simulating quantum computation," *Nature Computational Science*, vol. 1, no. 9, pp. 578–587, 2021.

[4] X.-C. Wu, S. Di, E. M. Dasgupta, F. Cappello *et al.*, "Full-state quantum circuit simulation by using data compression," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–24.

[5] T. Vincent, L. J. O'Riordan, M. Andrenkov, J. Brown *et al.*, "Jet: Fast quantum circuit simulations with parallel task-based tensor-network contraction," *Quantum*, vol. 6, p. 709, 2022.

[6] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush *et al.*, "Characterizing quantum supremacy in near-term devices," *Nature Physics*, vol. 14, no. 6, pp. 595–600, 2018.

[7] L. Chi-Chung, P. Sadayappan, and R. Wenger, "On optimizing a class of multi-dimensional loops with reduction for parallel execution," *Parallel Processing Letters*, vol. 7, no. 02, pp. 157–168, 1997.

[8] S. B. Akers, "Binary decision diagrams," *IEEE Transactions on computers*, vol. 27, no. 06, pp. 509–516, 1978.

[9] A. Zulehner and R. Wille, "Advanced simulation of quantum computations," *tcad*, vol. 38, no. 5, pp. 848–859, 2019.

[10] ——, "Advanced simulation of quantum computations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 848–859, 2018.

[11] Q. Zhang, M. Saligane, H.-S. Kim, D. Blaauw *et al.*, "Quantum circuit simulation with fast tensor decision diagram," *arXiv preprint arXiv:2401.11362*, 2024.

[12] C. Roberts, A. Milsted, M. Ganahl, A. Zalcman *et al.*, "Tensornetwork: A library for physics and machine learning," 2019.

[13] S. Hillmich, A. Zulehner, and R. Wille, "Concurrency in dd-based quantum circuit simulation," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 115–120.

[14] B. Zhang, B. Fang, F. Ye, Y. Gu *et al.*, "Overcoming memory constraints in quantum circuit simulation with a high-fidelity compression framework," *arXiv preprint arXiv:2410.14088*, 2024.

[15] Z. Dahi, E. Alba, R. Gil-Merino, F. Chicano *et al.*, "A survey on quantum computer simulators," 2023.

[16] F. Arute, K. Arya, R. Babbush, D. Bacon *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[17] I. L. Markov and Y. Shi, "Simulating quantum computation by contracting tensor networks," *SIAM Journal on Computing*, vol. 38, no. 3, pp. 963–981, 2008.

[18] D. Lykov, R. Schutski, A. Galda, V. Vinokur *et al.*, "Tensor network quantum simulator with step-dependent parallelization," in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2022, pp. 582–593.

[19] J. Gray and S. Kourtis, "Hyper-optimized tensor network contraction," *Quantum*, vol. 5, p. 410, 2021.

[20] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a bdd package," in *Proceedings of the 27th ACM/IEEE design automation conference*, 1991, pp. 40–45.

[21] N. Husung, C. Dubslaff, H. Hermanns, and M. A. Köhl, "Oxidd: a safe, concurrent, modular, and performant decision diagram framework in rust," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2024, pp. 255–275.

[22] I. Filippidis, S. Dathathri, S. C. Livingston, N. Ozay *et al.*, "Control design for hybrid systems with tulip: The temporal logic planning toolbox," in *2016 IEEE Conference on Control Applications (CCA)*. IEEE, 2016, pp. 1030–1041.

[23] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986.

[24] X. Hong, X. Zhou, S. Li, Y. Feng *et al.*, "A tensor network based decision diagram for representation of quantum circuits," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 27, no. 6, pp. 1–30, 2022.

[25] R. S. Sutor, *Dancing with Qubits: How quantum computing works and how it can change the world*. Packt Publishing Ltd, 2019.

[26] V. L. Oliva, "Fundamentos de la computación cuántica," *TEMat*, vol. 6, pp. 31–47, 2022.

[27] P. A. M. Dirac, "A new notation for quantum mechanics," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 35, no. 3, p. 416–418, 1939.

[28] J. C. Bridgeman and C. T. Chubb, "Hand-waving and interpretive dance: an introductory course on tensor networks," *Journal of physics A: Mathematical and theoretical*, vol. 50, no. 22, p. 223001, 2017.

[29] H. P. Luhn, "A new method of recording and searching information," *American Documentation*, vol. 4, no. 1, pp. 14–16, 1953.

[30] M. Vallero, P. Rech, and F. Vella, "State of practice: evaluating gpu performance of state vector and tensor network methods," *Future Generation Computer Systems*, vol. 174, p. 107927, 2026.

[31] N. Quetschlich, L. Burgholzer, and R. Wille, "MQT Bench: Benchmarking software and design automation tools for quantum computing," 2022, MQT Bench is available at https://www.cda.cit.tum.de/mqtbench/.

[32] J. Zhao, "Some size and structure metrics for quantum software," in *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*. IEEE, 2021, pp. 22–27.

[33] T. Tomesh, P. Gokhale, V. Omole, G. S. Ravi *et al.*, "Supermarq: A scalable quantum benchmark suite," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 587–603.

[34] A. Li, S. Stein, S. Krishnamoorthy, and J. Ang, "Qasmbench: A low-level quantum benchmark suite for nisq evaluation and simulation," *ACM Transactions on Quantum Computing*, vol. 4, no. 2, pp. 1–26, 2023.

[35] V. Lopez-Oliva, J. M. Badia, and M. Castillo, "Efficient quantum circuit contraction using tensor decision diagrams," *The Journal of Supercomputing*, vol. 81, no. 1, p. 354, 2025.

[36] C. Hayes and D. Malone, "An evaluation of fnv non-cryptographic hash functions," in *2024 35th Irish Signals and Systems Conference (ISSC)*. IEEE, 2024, pp. 1–8.

Comparación de métodos clásicos y cuánticos para la clasificación del peso en corderos

Elena Ondicol, Virginia Riego y Lidia Sánchez-González[1]

Resumen— **Este trabajo aborda la clasificación de corderos según su peso a partir de un dataset previamente disponible con características morfológicas calculadas a partir de imágenes cenitales. Primero, se seleccionan automáticamente las cuatro características más discriminativas mediante Random Forest. Sobre este conjunto reducido de características, se evalúan distintos modelos clásicos (Random Forest, SVM, XGBoost, Regresión Logística y KNN) empleando validación cruzada estratificada y técnicas de preprocesamiento estándar. Los resultados muestran que SVM alcanza la mejor capacidad de generalización (90,5 % en el conjunto de prueba), mientras que Random Forest destaca por su estabilidad en validación cruzada. Además, se analizan modelos cuánticos dentro del paradigma de aprendizaje híbrido, concretamente el Quantum Support Vector Classifier (QSVC) y el Variational Quantum Classifier (VQC). QSVC demuestra ser capaz de capturar parcialmente la estructura de los datos mediante kernels cuánticos, aunque con un rendimiento inferior al de los modelos clásicos en el conjunto de prueba (71,6 %). Por el contrario, VQC presenta un comportamiento inestable y bajo rendimiento, atribuible a las dificultades en la optimización variacional. El análisis de matrices de confusión confirma que los modelos clásicos logran una mejor separación entre categorías de peso (bajo, medio y alto), mientras que los cuánticos muestran mayores niveles de confusión. En conjunto, los resultados indican que, si bien los métodos cuánticos presentan potencial, los enfoques clásicos siguen siendo superiores en problemas de clasificación con datos estructurados y de baja dimensionalidad.**

Palabras clave— **Clasificación supervisada, clasificación cuántica, estimación peso corderos, características morfométricas.**

I. Introducción

La ganadería de precisión (Precision Livestock Farming - PLF) se basa en el uso de nuevas tecnologías y avances recientes para automatizar ciertas tareas y facilitar las más exigentes [1]. Entre las tareas que se pueden llevar a cabo están el conteo de ganado, análisis del comportamiento de los animales o la detección de los animales. Todos estos aspectos tienen una influencia crucial en los beneficios obtenidos [2], así como en el bienestar animal [3], [4].

Una de las cuestiones más importantes en la ganadería es el control en el crecimiento de los animales, ya que permite prevenir enfermedades, optimizar la cría y el consumo de piensos [5]. En el caso del pesaje en ovejas, hay trabajos donde se propone un sistema de estereovisión que estima el tamaño y el peso de las ovejas a partir de una vista frontal con dos cámaras [6]; aunque la muestra está formada por solo 27 ovejas vivas y el sistema requiere que las cámaras estén perfectamente alineadas, por lo que no es fácil de configurar. En [7], se predice la masa muscular utilizando modelos de regresión múltiple considerando características calculadas a partir de imágenes de ultrasonido de cabras y ovejas vivas e incluyendo el peso como entrada. El requisito de dispositivos complejos y costosos [8], [9] hace que estos enfoques sean menos atractivos para un uso generalizado.

Por otra parte, la computación cuántica (QC) presenta un paradigma revolucionario en el procesamiento de la información, ya que aprovecha los fenómenos de la mecánica cuántica para realizar cálculos de formas fundamentalmente diferentes a las de la computación clásica (CC). A pesar de las afirmaciones sobre la "supremacía cuántica", la QC se encuentra todavía en sus primeras etapas de desarrollo, aunque existen numerosos trabajos que abordan aspectos tanto hardware como software, proponiendo soluciones que se podrán implementar a medida que los computadores cuánticos vayan evolucionando. Aquí se enmarca este trabajo.

La viabilidad de su aplicación en el ámbito agrícola [10] dependerá, en última instancia, de la caracterización formal de los problemas en términos de complejidad computacional. En particular, será relevante determinar si dichos problemas implican la simulación de sistemas físicos gobernados por dinámicas cuánticas cuya resolución eficiente excede las capacidades de algoritmos clásicos en tiempo polinómico (i.e., fuera de clases tratables como P), o si pueden formularse como problemas de optimización combinatoria pertenecientes a clases de elevada complejidad (por ejemplo, NP-hard), para los cuales determinados algoritmos cuánticos puedan ofrecer una aceleración cuadrática/exponencial frente a los métodos clásicos convencionales.

La aplicación cuántica en visión por computador [11], establece la necesidad de crear algoritmos novedosos y convertir y recrear los algoritmos clásicos con el paradigma cuántico, ya que no existen mecanismos entre los algoritmos clásicos existentes y los algoritmos cuánticos que puedan reducir la tarea de crear algoritmos cuánticos desde cero [12], [13], [14]. Además, no es trivial mapear los algoritmos clásicos de visión artificial y sus equivalentes cuánticos debido a las diferencias en los métodos de computación, reduciendo la portabilidad de los algoritmos. Trabajos previos como [15], procesan imágenes de piezas mecanizadas con el fin de detectar los bordes de la máscara, lo que puede determinar la presencia de rebabas generadas en los procesos de fabricación industrial. Los resultados de la prueba muestran que, en comparación con la computación tradicional, los algoritmos cuánticos aún no son lo suficientemente

[1]Robotics Group, I4 Institute, Department of Mechanical, Computer Science and Aerospace Engineering, Universidad de León, León, Spain, e-mail: vriec,lidia.sanchez@unileon.es

eficientes como para igualar o incluso superar a los algoritmos clásicos de visión por computador. Existen soluciones que utilizan la computación cuántica en el campo de la visión para realizar tareas de correspondencia estereoscópica [16], para optimizar el problema de optimización binaria cuadrática sin restricciones (QUBO) que elimina detecciones redundantes en el algoritmo *Non-maximum suppression (NMS)* [17] o para la segmentación del movimiento [18]. También se emplea para realizar tareas de clasificación y regresión con datasets típicos de imagen como MNIST y Fashion-MNIST [19] o para clasificar sonidos [20], entre otras aplicaciones.

Considerando estos trabajos, se propone una solución para clasificar imágenes de cordero según su peso a partir de un conjunto de características morfométricas extraídas previamente, comparando los resultados de los clasificadores clásicos frente a las soluciones cuánticas existentes.

El resto del artículo se organiza como sigue: la sección II explica el dataset utilizado. En la sección III se describen los métodos considerados, tanto clásicos como cuánticos. Los resultados experimentales se comentan en la sección IV y finalmente, la sección V recoge las conclusiones alcanzadas.

II. Dataset

En un estudio previo, se disponía de un dataset con información sobre corderos de ambos sexos de raza Rasa Aragonesa que pesaban entre 14 y 27,7 kilos [21] y se usaban sistemas 3D para estimar el peso de los corderos. El experimento se basó en tomar imágenes RGB-D con una cámara Intel Real Sense D435, las cuales se segmentaban utilizando la distancia calculada a partir de la información de profundidad y se calculaba el área del cordero segmentado. Comparando distintos métodos, se realizaba una estimación del peso considerando como características el área de esa zona y el sexo del animal; en este trabajo, se obtuvo un error medio absoluto (MAE) de 1,37 kilos y un error medio absoluto relativo (MRAE) menor del 6 %, así como un R2 del 80 %.

Basándose en el experimento anterior, en [22] se propone un sistema de visión por computador capaz de describir el contorno de la máscara de la imagen cenital de un cordero con las siguientes características: área, *bounding box* (ancho y alto de la misma), elipse (eje mayor, menor, simetría y excentricidad de la misma), perímetro del contorno y porcentaje de cobertura de la máscara en la *bounding box*. Dichas características se utilizaron para entrenar sistemas de regresión para la estimación del peso, obteniendo un R2 del 90 %, un MAE de 0,75 kilos y un MRAE del 4 %.

En la preparación de los datos de entrada, se contempló que múltiples capturas pueden pertenecer a un mismo identificador de animal, y que el área registrada en píxeles posee una dependencia directa de la altura de la cámara obtenida mediante el sensor de profundidad. Cabe destacar que, si bien el sexo del animal estaba contemplado inicialmente en el registro, se descartó del vector de entrada al observar que su impacto predictivo era irrelevante. En la Figura 1 se puede observar la distribución de pesos y valores que se tendrán en cuenta para la experimentación.

Dado que la variable objetivo original es el peso continuo del animal, el conjunto de datos presentaba naturaleza de regresión. Con el objetivo de simplificar el problema y facilitar su abordaje inicial mediante clasificadores cuánticos, la tarea se transformó en clasificación. Para ello, la distribución continua del peso se discretizó en tres categorías (terciles), definiendo las clases de peso bajo, medio y alto. A continuación, se balancearon los conjuntos de datos para asegurar un número parejo de ejemplos por clase y evitar una distribución desbalanceada que generara sesgos durante el entrenamiento. En la Figura 2 se puede observar la distribución final de las categorías con la separación de datos para entrenamiento y test.

El sistema original extrae 9 características morfológicas del contorno de corderos. Con PCA (*Principal Component Analysis*) se transforman estas 9 variables originales (potencialmente redundantes) en 6 componentes principales ortogonales que capturan la máxima varianza: PC1 (tamaño general), PC2 (forma vertical/horizontal), PC3 (perímetro complejo), PC4 (simetría), PC5 y PC6 (combinaciones residuales), reduciendo dimensionalidad sin perder información esencial para la clasificación. Después se utilizó un clasificador Random Forest que permitió obtener las características de mayor importancia de cara a la clasificación, pudiendo reducir el dataset final a 4 características como se puede ver en la Figura 3: PC2 (forma vertical/horizontal), PC3 (perímetro complejo), PC4 (simetría), PC5 (combinaciones residuales). Todos estos valores fueron normalizados para facilitar la convergencia de los clasificadores.

III. Materiales y métodos

En esta parte se han llevado a cabo dos líneas de experimentación: por un lado el análisis de cómo se comportan los clasificadores tradicionales y por otra, el análisis del rendimiento de los métodos cuánticos con los mismos datos.

A. Clasificadores tradicionales

Se evaluaron cinco modelos clásicos de clasificación supervisada (Random Forest, SVM, XGBoost, Regresión Logística y KNN) para predecir las categorías de peso de corderos (bajo/medio/alto) utilizando las 4 características PCA más importantes identificadas por Random Forest.

La evaluación se realizó mediante validación cruzada estratificada de 5-fold, que divide sistemáticamente los datos de entrenamiento en 5 particiones iguales manteniendo la proporción de clases, entrenando 5 veces y promediando la precisión para obtener una estimación robusta del rendimiento generalizado y reducir el sesgo de partición única.

Para ello se utilizó la técnica de Grid Search que evalua los mejores hiperparámetros de entrenamiento de cada uno de estos modelos. Posteriormente, cada

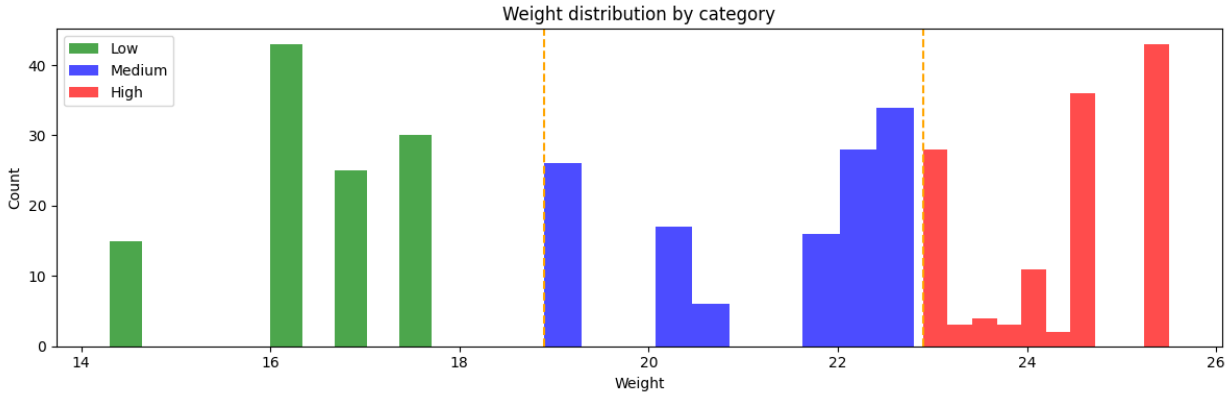


Fig. 1: Distribución del número de corderos del dataset según su peso

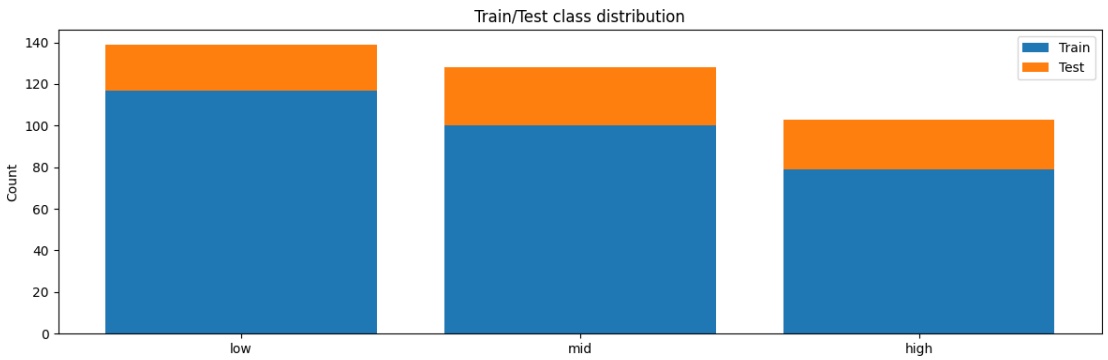


Fig. 2: Distribución del número de datos por categoría que se usan para entrenamiento y prueba

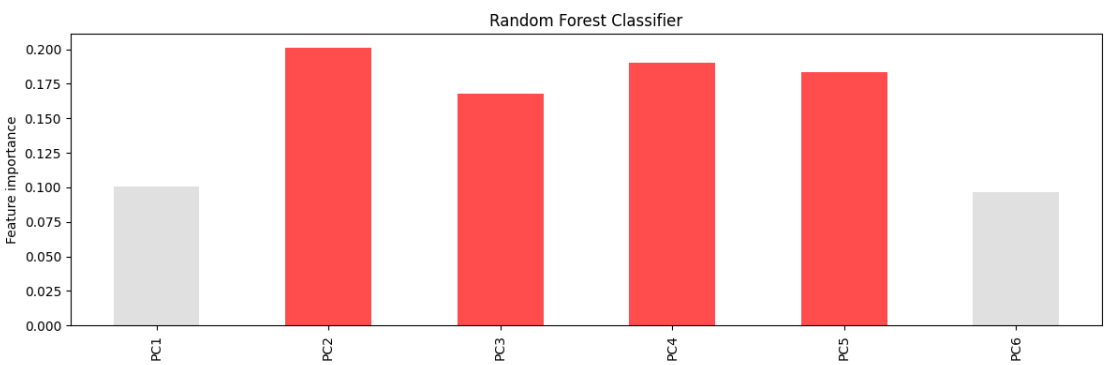


Fig. 3: Importancia de las características de PCA para un clasificador Random Forest

modelo se entrenó con todos los datos de entrenamiento y evaluó en el conjunto de prueba independiente, calculando la exactitud tanto en el entrenamiento como en el test para detectar sobreajuste.

B. Clasificadores cuánticos

B.1 Quantum Support Vector Classifier (QSVC)

El Quantum Support Vector Classifier (QSVC) [23], [24], [25] constituye una extensión cuántica del algoritmo clásico Support Vector Machine basada en la evaluación de kernels cuánticos en espacios de Hilbert de alta dimensión. Las características anteriormente descritas se codifican en estados cuánticos mediante un mapa de características $\phi(x)$, implementado a través de circuitos cuánticos parametrizados, lo que permite proyectar los datos en un espacio donde la separabilidad puede mejorar significativamente. El kernel cuántico $K(x_i, x_j) = |\langle\phi(x_i)|\phi(x_j)\rangle|^2$ captura correlaciones no lineales complejas entre las variables, facilitando la construcción de un hiperplano óptimo que permite discriminar entre las tres clases de peso (bajo, medio y alto). A partir de la matriz de kernel, se resuelve el problema de optimización para identificar los vectores de soporte y maximizar el margen de separación entre categorías.

B.2 Algoritmo Variacional Cuántico (QVA)

Los Algoritmos Variacionales Cuánticos (QVA) [26], [27] constituyen un marco general de métodos híbridos cuántico-clásicos basados en circuitos cuánticos parametrizados y procedimientos de optimización clásica. Dentro de este marco, el modelo empleado en este trabajo corresponde a un *Variational Quantum Classifier* (VQC) [28], [29], es decir, una instancia particular de QVA orientada a tareas de clasificación supervisada. El conjunto de datos anteriormente descrito, clasifica en tres categorías: bajo, medio y alto. Dado que el VQC implementa-

do presenta una naturaleza intrínsecamente binaria, se adopta una estrategia *one-vs-rest*, entrenando tres clasificadores independientes, cada uno encargado de discriminar una clase frente al resto. La salida del circuito variacional, obtenida mediante la medición de observables, se interpreta como una probabilidad de pertenencia a la clase objetivo. Los parámetros optimizados permiten realizar la clasificación multiclase y proporcionan información sobre la contribución relativa de las características de entrada, favoreciendo la interpretabilidad del modelo.

IV. Resultados experimentales

Se evaluaron inicialmente cinco modelos tradicionales (Random Forest, SVM, XGBoost, Regresión Logística, KNN) utilizando las 4 componentes principales más discriminativas seleccionadas automáticamente por Random Forest sobre las 6 PCs morfológicas derivadas del contorno de corderos. La metodología empleó validación cruzada 5-fold estratificada sobre datos de entrenamiento balanceados, codificación one-hot de categorías peso (bajo/medio/alto), y normalización estándar para equiparar escalas dispares. La Tabla I resume los resultados de la validación cruzada y del entrenamiento llevado a cabo con todos los datos de entrenamiento y evaluado con los datos de prueba. SVM logra la máxima precisión en el conjunto de prueba (90,5 %) con menor sobreajuste (en el conjunto de entrenamiento alcanza el 94,6 %), mientras que Random Forest destaca en los resultados de la validación cruzada (92,9 %) por su estabilidad.

Posteriormente se evaluaron modelos cuánticos sobre el mismo conjunto de datos, comparando directamente con los métodos clásicos que dieron los mejores resultados. Entre los modelos cuánticos, destaca QSVC con un 94,3 % para los datos de entrenamiento y un 71,6 % para los de prueba, capturando estructuras relevantes en las 4 PCs sin un exceso sobreajuste, aunque queda por debajo del SVM clásico (90,5 %). Este comportamiento, visible en la Tabla II es consistente con estudios que muestran que el QSVC es competitivo en datasets tabulares. En contraste, los resultados de VQC son mucho peores (44,6 %/52,7 %), debido a la inestabilidad del ansatz variacional y problemas de optimización, un problema recurrente en la literatura cuántica. En conclusión, QSVC representa una alternativa cuántica viable, aunque los modelos clásicos siguen proporcionando resultados superiores en este problema de clasificación.

Las matrices de confusión de la Figura 4 revelan diferencias notables entre modelos clásicos y cuánticos en test. Random Forest y SVM obtienen matrices diagonalmente dominantes con alta precisión en la diagonal principal, pocos falsos positivos y correcta separación bajo/medio/alto, confirmando su exactitud en un 87,8 % y un 90,5 % para los datos de prueba. En contraste, QSVC muestra una peor diagonal con una confusión notable de la categoría medio-alto, indicando que el kernel cuántico captura parcialmente pero no refina fronteras complejas tan bien como el SVM clásico. VQC presenta la peor matriz con una distribución uniforme y una alta confusión global, evidenciando el fallo del ansatz variacional. Por tanto, los métodos clásicos superan consistentemente a los cuánticos en generalización real.

V. Conclusiones

En conjunto, los resultados obtenidos evidencian que los modelos clásicos continúan ofreciendo el mejor rendimiento en la tarea de clasificación del peso de corderos a partir de características morfológicas reducidas mediante PCA. En particular, SVM logra el mejor equilibrio entre capacidad de generalización y control del sobreajuste, alcanzando la mayor precisión en el conjunto de prueba (un 90,5 %), mientras que Random Forest destaca por su estabilidad en validación cruzada, aunque con algo de sobreajuste en el entrenamiento. Estos resultados confirman que, en espacios de baja dimensionalidad y con características bien seleccionadas, los métodos clásicos siguen siendo altamente competitivos y robustos.

Por otro lado, los modelos cuánticos muestran un comportamiento desigual. QSVC demuestra ser capaz de capturar cierta estructura relevante en los datos, obteniendo un rendimiento razonable sin sobreajuste severo, lo que sugiere que los kernels cuánticos pueden aportar valor en la modelización de relaciones no lineales. Sin embargo, su rendimiento con los datos de prueba (un 71,6 %) queda significativamente por debajo de su equivalente clásico, evidenciando limitaciones en la capacidad de generalización en este contexto. En cuanto al VQC, presenta un desempeño claramente insuficiente, con resultados cercanos al azar, lo que pone de manifiesto problemas inherentes a los algoritmos variacionales actuales, como la inestabilidad del ansatz y la dificultad en la optimización de parámetros en espacios de alta complejidad.

El análisis de las matrices de confusión refuerza estas conclusiones, mostrando que los modelos clásicos logran una separación clara entre las categorías de peso, mientras que QSVC presenta confusiones notables (especialmente entre las clases media y alta) y el VQC falla en capturar patrones discriminativos. En resumen, aunque los enfoques cuánticos, y en particular QSVC, muestran cierto potencial, los resultados indican que, en el estado actual de la tecnología y para este tipo de datos estructurados y de baja dimensión, los modelos clásicos siguen siendo la opción más eficaz y fiable.

Agradecimientos

Proyecto PID2024-161761OB-C21 financiado por MICIU/AEI/10.13039/501100011033 y por FEDER/UE.

Referencias

[1] Parisa Niloofar, Deena P. Francis, Sanja Lazarova-Molnar, Alexandru Vulpe, Marius-Constantin Vochin, George Suciu, Mihaela Balanescu, Vasileios Anestis, and Thomas Bartzanas, "Data-driven decision support in livestock farming for improved animal health, welfare and greenhouse gas emissions: Overview and challenges,"

Tabla I: Comparación de la exactitud en los Modelos Clásicos de Clasificación

Modelo	Validación Cruzada	Entrenamiento	Prueba
Random Forest	**0,929**	1,000	0,878
SVM	0,909	0,946	**0,905**
XGBoost	0,912	1,000	0,892
Regresión Logística	0,895	0,892	0,865
KNN	0,895	0,946	0,878

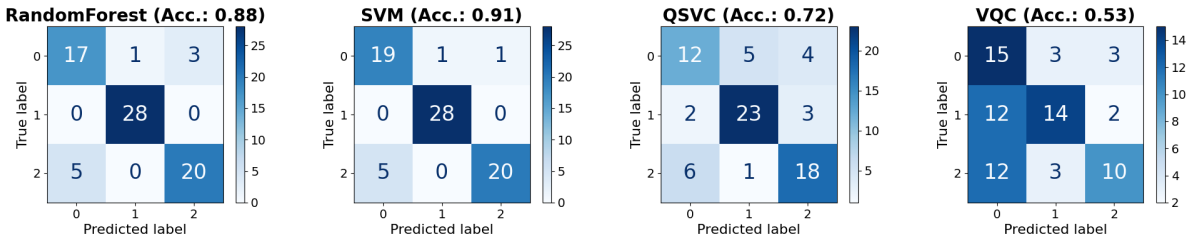


Fig. 4: Matrices de confusión de modelos clásicos (SVM y Random Forest) y cuánticos (QSVC y VQC) para los datos de prueba

Tabla II: Comparación de la exactitud en los Modelos Cuánticos de Clasificación

Modelo	Entrenamiento	Prueba
QSVC	**0,943**	**0,716**
VQC	0,446	0,527

Computers and Electronics in Agriculture, vol. 190, pp. 106406, 2021.

[2] Nathalie Hostiou, Jocelyn Fagon, Sophie Chauvat, Amélie Turlot, Florence Kling-Eveillard, Xavier Boivin, and Clément Allain, "Impact of precision livestock farming on work and human-animal interactions on dairy farms. a review," *Biotechnology, Agronomy and Society and Environment*, vol. 21, no. 4, pp. 268–275, 2017.

[3] Chen Chen, Weixing Zhu, Yizheng Guo, Changhua Ma, Weijia Huang, and Chengzhi Ruan, "A kinetic energy model based on machine vision for recognition of aggressive behaviours among group-housed pigs," *Livestock Science*, vol. 218, no. April 2017, pp. 70–78, 2018.

[4] D. T. Byrne, H. Esmonde, D. P. Berry, F. McGovern, P. Creighton, and N. McHugh, "Sheep lameness detection from individual hoof load," *Computers and Electronics in Agriculture*, vol. 158, no. August 2018, pp. 241–248, 2019.

[5] Nirav Alpesh Shah, Jaydeep Thik, Chintan Bhatt, and Aboul-Ella Hassanien, "A deep convolutional encoder-decoder architecture approach for sheep weight estimation," in *Advances in Artificial Intelligence and Data Engineering*, Niranjan N. Chiplunkar and Takanori Fukao, Eds., Singapore, 2021, pp. 43–53, Springer Singapore.

[6] Paolo Menesatti, Corrado Costa, Francesca Antonucci, Roberto Steri, Federico Pallottino, and Gennaro Catillo, "A low-cost stereovision system to estimate size and weight of live sheep," *Computers and Electronics in Agriculture*, vol. 103, pp. 33–38, 2014.

[7] Luís G. Dias, Severiano R. Silva, and Alfredo Teixeira, "Simultaneously prediction of sheep and goat carcass composition and body fat depots using in vivo ultrasound measurements and live weight," *Research in Veterinary Science*, vol. 133, pp. 180–187, 2020.

[8] Diego André Sant'Ana, Marcio Carneiro Brito Pache, José Martins, Wellington Pereira Soares, Sebasti ao Lucas Neves de Melo, Vanir Garcia, Vanessa Aparecida de Moares Weber, Natália da Silva Heimbach, Rodrigo Gonçalves Mateus, and Hemerson Pistori, "Weighing live sheep using computer vision techniques and regression machine learning," *Machine Learning with Applications*, vol. 5, pp. 100076, 2021.

[9] E. González-García, M. Alhamada, J. Pradel, S. Douls, S. Parisot, F. Bocquier, J.B. Menassol, I. Llach, and L.A. González, "A mobile and automated walk-over-weighing system for a close and remote monitoring of liveweight in sheep," *Computers and Electronics in Agriculture*, vol. 153, pp. 226–238, 2018.

[10] Luis O Tedeschi, "Asas-nanp symposium: Mathematical modeling in animal nutrition: quantum computing in agricultural sciences: from theory to reality," *Journal of Animal Science*, vol. 104, pp. skaf445, 2026.

[11] Naoual El Djouher Mebtouche and Sarah Sahnoune, "Quantum computing for computer vision: Applications, challenges, and research tracks," in *International Symposium on Quantum Sciences: Applications and Challenges*. Springer, 2024, pp. 152–166.

[12] Sebastian Raubitzek and Kevin Mallinger, "On the applicability of quantum machine learning," *Entropy*, vol. 25, no. 7, 2023.

[13] Cesarino Bertini and Roberto Leporini, "Quantum-inspired applications for classification problems," *Entropy*, vol. 25, no. 3, 2023.

[14] Vojtěch Havlíček, Antonio D. Córcoles, and Kristan et al. Temme, "Supervised learning with quantum-enhanced feature spaces," *Nature*, vol. 567, pp. 209–212, 2019.

[15] Santiago Merino-Bajo, Lidia Sánchez-González, Virginia Riego, and Vicente Matellán, "Effectiveness of quantum computing in image processing for burr detection," in *International Conference on Soft Computing Models in Industrial and Environmental Applications*. Springer, 2023, pp. 102–111.

[16] Shahrokh Heidari, Michael J. Dinneen, and Patrice Delmas, "Quantum annealing for computer vision minimization problems," *Future Generation Computer Systems*, vol. 160, pp. 54–64, 2024.

[17] Junde Li and Swaroop Ghosh, "Quantum-soft qubo suppression for accurate object detection," in *Computer Vision – ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, Eds., Cham, 2020, pp. 158–173, Springer International Publishing.

[18] Federica Arrigoni, Willi Menapace, Marcel Seelbach Benkner, Elisa Ricci, and Vladislav Golyanik, "Quantum motion segmentation," in *Computer Vision – ECCV 2022*, Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, Eds., Cham, 2022, pp. 506–523, Springer Nature Switzerland.

[19] Teppei Suzuki, Takashi Hasebe, and Tsubasa Miyazaki, "Quantum support vector machines for classification and regression on a trapped-ion quantum computer," *Quantum Machine Intelligence*, vol. 6, no. 1, pp. 31, May 2024.

[20] Sandra Ranilla-Cortina, Antonio J. Muñoz-Montoro, Elías F. Combarro, Sebastián García-Galán, and José Ranilla, "A quantum machine learning approach to cardiopulmonary sound classification," in *Artificial Intelligence in Biomedicine*, Aurelio López Fernández, Alejandro Rodríguez-González, Raquel Leirós-Rodríguez, Christian Mata Miquel, and Víctor Manuel González Suárez, Eds., Cham, 2026, pp. 228–238, Springer Nature Switzerland.

[21] E. Samperio, I. Lidón, R. Rebollar, M. Castejón-Limas, and C. Álvarez-Aparicio, "Lambs' live weight estimation using 3d images," *Animal*, vol. 15, no. 5, pp. 100212, 2021.

[22] Virginia Riego del Castillo, Lidia Sánchez-González, Laura Fernández, Ruben Rebollar, and Enrique Samperio, "A non-stressful vision-based method for weighing live lambs," *Logic Journal of the IGPL*, vol. 33, no. 5, pp. jzae059, 2025.

[23] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd, "Quantum support vector machine for big data classification," *Phys. Rev. Lett.*, vol. 113, pp. 130503, Sep 2014.
[24] Anekait Kariya and Bikash K. Behera, "Investigation of quantum support vector machine for classification in nisq era," 2021.
[25] Zhaokai Li, Xiaomei Liu, Nanyang Xu, and Jiangfeng Du, "Experimental realization of a quantum support vector machine," *Phys. Rev. Lett.*, vol. 114, pp. 140504, Apr 2015.
[26] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature Communications*, vol. 5, no. 1, pp. 4213, Jul 2014.
[27] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann, "A quantum approximate optimization algorithm," 2014.
[28] Maria Schuld, Alex Bocharov, Krysta M. Svore, and Nathan Wiebe, "Circuit-centric quantum classifiers," *Phys. Rev. A*, vol. 101, pp. 032308, Mar 2020.
[29] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, "Quantum circuit learning," *Phys. Rev. A*, vol. 98, pp. 032309, Sep 2018.

Clasificador Cuántico Variacional para Datos Desbalanceados y No Lineales en Simulaciones Climáticas

Laura M. Donaire, Elena R. Rull, Marouan Khollala, Gloria Ortega, Francisco Orts y Ester M. Garzón[1]

Resumen— **Las simulaciones climáticas son herramientas fundamentales para comprender el cambio climático, pero el uso de configuraciones inadecuadas de sus parámetros pueden provocar fallos críticos (crashes) durante su ejecución. La predicción temprana de estos eventos es vital para optimizar los recursos de supercomputación y mejorar la fiabilidad de los modelos. Este trabajo presenta el desarrollo de un Clasificador Cuántico Variacional optimizado para el conjunto de datos *Climate Model Simulation Crashes*, caracterizado por relaciones altamente no lineales y un severo desbalanceo de clases (8,52 % de fallos). Se describe una metodología integral que abarca desde la selección de las tres características más determinantes mediante análisis de relevancia, permitiendo reducir la dimensionalidad a solo tres qubits, hasta un estudio comparativo del impacto del balanceo sintético mediante la técnica SMOTE. Los resultados experimentales demuestran que la arquitectura propuesta alcanza una exactitud del 95,37 %, una precisión del 97,60 % y un F1-Score del 79,54 %, superando el rendimiento de los enfoques clásicos (SVM) y cuánticos (QKE) documentados en el estado del arte. Además, la eficiencia y baja profundidad del circuito diseñado han hecho posible su validación utilizando un procesador cuántico real de IBM. Estos hallazgos evidencian el potencial práctico y la viabilidad del *Quantum Machine Learning* para resolver problemas científicos complejos bajo las restricciones de la actual era NISQ.**

Palabras clave— **Quantum Machine Learning, Simulación climática, Machine Learning.**

I. Introducción

LAS simulaciones climáticas son herramientas computacionales fundamentales para el estudio y la comprensión del cambio climático, permitiendo proyectar escenarios futuros y evaluar el impacto de diversas variables ambientales. Sin embargo, la complejidad de estos modelos radica en la inmensa cantidad de parámetros involucrados en su configuración. Una combinación inadecuada de estas variables puede provocar fallos críticos o caídas durante la ejecución de la simulación [1]. Estos fallos no solo interrumpen el análisis científico, sino que también representan un desperdicio significativo de tiempo y recursos computacionales, dada la gran envergadura de los cálculos requeridos [2].

En este contexto, la detección y predicción temprana de configuraciones propensas a fallos es crucial para mejorar la fiabilidad de los modelos climáticos y optimizar los recursos de supercomputación. Sin embargo, los métodos tradicionales de evaluación y ajuste manual de parámetros a menudo resultan ineficientes frente a la alta dimensionalidad y las relaciones no lineales de los datos subyacentes. Como resultado, el uso de técnicas de inteligencia artificial y aprendizaje predictivo ha ganado relevancia para automatizar la identificación de configuraciones problemáticas antes de su ejecución.

El aprendizaje automático clásico (*Machine Learning*, ML) ha transformado la predicción y clasificación en múltiples disciplinas mediante el reconocimiento de patrones en grandes volúmenes de datos. En el análisis de simulaciones, los modelos clásicos como las Máquinas de Soporte Vectorial (SVM) y los algoritmos basados en conjuntos de árboles de decisión han demostrado ser herramientas útiles para predecir fallos [1], [3]. Sin embargo, estos enfoques tradicionales enfrentan limitaciones físicas que frenan su avance, un fenómeno anticipado por el declive de la Ley de Moore [4]. Además, los algoritmos clásicos a menudo requieren conjuntos de datos de gran tamaño y perfectamente balanceados para lograr un rendimiento óptimo. Sin embargo, en problemas reales, como los fallos en simulaciones climáticas, los datos suelen estar altamente desbalanceados, lo que degrada severamente la capacidad del modelo para identificar la clase minoritaria.

El aprendizaje automático cuántico (*Quantum Machine Learning*, QML) es un campo emergente que surge como una alternativa prometedora, aprovechando principios de la mecánica cuántica como la superposición y el entrelazamiento para acelerar el procesamiento de información [5]. La capacidad de los sistemas cuánticos para mapear características en espacios de Hilbert de alta dimensionalidad permite aprender patrones complejos con conjuntos de datos más reducidos y sin la estricta necesidad de balancear las clases de manera artificial [6]. Actualmente, la disciplina se encuentra en la era cuántica de escala intermedia ruidosa (*Noisy Intermediate-Scale Quantum*, NISQ) [7]. Esta era se caracteriza por disponer de un número limitado de recursos (qubits) y por la susceptibilidad de los sistemas a la decoherencia y errores en las puertas lógicas [8]. Para superar estas barreras, han surgido los enfoques híbridos, que combinan el poder de representación cuántica con la optimización clásica. Dentro de estos, los Clasificadores Cuánticos Variacionales (VQC) se posicionan como soluciones óptimas en la era NISQ, ya que requieren un número reducido de qubits y circuitos de poca profundidad [9].

[1]Dpto. de Informática, Universidad de Almería, e-mail: {laura.donaire, err984, mk350, gloriaortega, francisco.orts, gmartin}@ual.es.

En este trabajo, se diseña y optimiza un modelo híbrido basado en VQC para la predicción de fallos en simulaciones climáticas. El modelo propuesto integra capas de codificación cuántica y circuitos parametrizados, aprovechando las ventajas de la computación cuántica mientras mantiene una arquitectura resistente a los errores de hardware. Las principales contribuciones de este estudio son: (1) El desarrollo de un modelo híbrido eficiente y compacto que reduce la dimensionalidad del problema para operar con tan solo tres qubits, haciéndolo factible para los dispositivos actuales en la era NISQ. (2) La evaluación del modelo utilizando el *Climate Model Simulation Crashes Dataset* (3) Una comparativa de rendimiento detallada frente a enfoques del estado del arte, abarcando tanto modelos clásicos como cuánticos basados en estimación de kernel. (4) La validación empírica de la arquitectura propuesta mediante su ejecución no solo en simuladores ideales, sino también en hardware cuántico real de IBM, demostrando la viabilidad de su implementación práctica.

II. Quantum Machine Learning

El QML investiga cómo los algoritmos cuánticos pueden optimizar y acelerar las tareas de aprendizaje automático al aprovechar principios fundamentales de la mecánica cuántica, como la superposición y el entrelazamiento [5]. Este paradigma se fundamenta en la representación de datos mediante estados cuánticos y su posterior procesamiento a través de circuitos cuánticos parametrizados [4], [9]. El campo del QML se divide principalmente en dos grandes vertientes: los enfoques puramente cuánticos y los enfoques híbridos cuántico-clásicos.

A. Enfoques Cuánticos Puros

Los algoritmos puramente cuánticos ejecutan la totalidad de sus operaciones computacionales en hardware cuántico, con el objetivo de obtener aceleraciones exponenciales respecto a sus contrapartes clásicas. Entre los modelos más relevantes en la literatura destacan las Máquinas de Soporte Vectorial Cuánticas (QSVM). Al igual que las SVM tradicionales, las QSVM buscan un hiperplano óptimo para separar datos en distintas categorías [10]. La diferencia fundamental reside en que las QSVM transforman los datos clásicos en estados cuánticos a través de mapas de características cuánticos (*Feature Maps*). Estos mapas aplican rotaciones parametrizadas y puertas de entrelazamiento para codificar las características en un estado cuántico complejo [11]. Posteriormente, se emplea un circuito cuántico para evaluar las similitudes entre dichos estados, definiendo así un kernel cuántico [12], el cual permite identificar patrones no lineales de forma intrínsecamente más eficiente [13].

Otro desarrollo reciente en esta categoría son las Redes Neuronales Completamente Cuánticas (FQNN). A diferencia de las redes cuánticas convencionales que combinan infraestructuras clásicas y cuánticas [14], las FQNN operan de manera exclusiva en el dominio cuántico. El entrenamiento de estas arquitecturas se lleva a cabo mediante la estimación de gradientes direccionales, aproximando los gradientes de los parámetros cuánticos a través de diferencias finitas, habiendo demostrado alta eficacia en conjuntos de datos estándar [15].

B. Enfoques Híbridos

Para mitigar las limitaciones físicas inherentes a la era NISQ, como la decoherencia y los errores de puerta, los enfoques híbridos proponen una sinergia entre recursos cuánticos y clásicos. En este esquema, los circuitos cuánticos variacionales (*Ansatz*) actúan como evaluadores que se optimizan iterativamente mediante algoritmos clásicos de retropropagación.

El VQC es uno de los exponentes principales de este enfoque para tareas de clasificación binaria. Su arquitectura consta de tres fases fundamentales:

- **Codificación con Mapa de Características:** Los datos clásicos se proyectan en el espacio de Hilbert de los qubits utilizando un mapa de características no lineal.
- **Circuito Variacional:** Los datos codificados se procesan a través del *Ansatz*, que consiste en una secuencia parametrizada de puertas cuánticas orientada a minimizar una función de pérdida definida clásicamente.
- **Medición y Optimización:** Se mide el estado final de los qubits para evaluar la pérdida y, mediante un optimizador clásico, se actualizan los parámetros del circuito para la siguiente iteración.

La Figura 1 ilustra el flujo de trabajo del clasificador descrito.

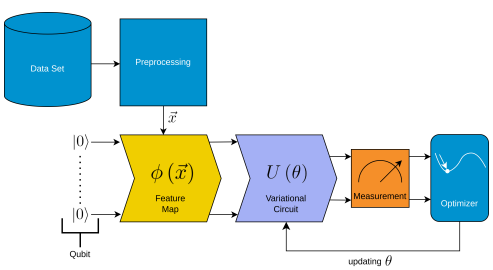


Fig. 1: Diagrama de flujo del funcionamiento de VQC.

Adicionalmente, la Estimación Cuántica de Kernel (QKE) representa otra técnica híbrida que utiliza el cálculo cuántico de funciones kernel para potenciar métodos clásicos como las SVM. En lugar de calcular explícitamente la matriz de kernel computacionalmente costosa, los circuitos cuánticos evalúan empíricamente la superposición entre los estados que representan cada dato. Este kernel precalculado cuánticamente alimenta posteriormente al algoritmo clásico de clasificación para trazar las fronteras de decisión [13].

III. Metodología e Implementación

En esta sección se presenta la metodología detallada que se ha utilizado para el desarrollo y evaluación del modelo propuesto. Todo el entorno experimental

se ha implementado utilizando el lenguaje de programación Python. Para el manejo, exploración y preprocesamiento de los datos, se hizo uso de librerías estándar como *Pandas*, mientras que el diseño, simulación y ejecución de los circuitos cuánticos se llevó a cabo mediante *Qiskit*, un entorno de desarrollo de código abierto impulsado por IBM para la computación cuántica.

La Figura 2 resume, de forma general, las distintas fases llevadas a cabo en el desarrollo de esta investigación. En primer lugar, se ejecutó un proceso de preprocesamiento de los datos con el objetivo de prepararlos adecuadamente para su posterior clasificación. A continuación, se empleó la interfaz *Qiskit* como entorno de desarrollo para la construcción del clasificador, sobre el cual se procedió a la optimización de diversos hiperparámetros. Posteriormente, el modelo fue entrenado utilizando la base de datos *Climate Model Simulation Crashes*, y evaluado mediante métricas robustas y ampliamente aceptadas en el ámbito del aprendizaje automático. Finalmente, los resultados obtenidos fueron comparados con aquellos reportados en el estado del arte, con el fin de valorar el rendimiento del enfoque propuesto.

A. Descripción del Conjunto de Datos

La investigación utiliza el conjunto de datos *Climate Model Simulation Crashes* [2], que contiene 540 registros con 18 parámetros descriptivos obtenidos mediante muestreo de hipercubo latino [1]. El problema se define como una clasificación binaria (éxito o fallo de la simulación). El conjunto presenta un desbalanceo intrínseco severo: el 91.48 % de las muestras corresponden a éxitos (494 casos) y solo el 8.52 % a fallos (46 casos), lo que representa un desafío crítico para la detección de la clase minoritaria.

B. Preprocesamiento y Selección de Características

Dada la limitación de recursos en la era NISQ, hemos priorizado la reducción de dimensionalidad. Tras descartar el Análisis de Componentes Principales (PCA) por la baja varianza explicada (46.40 % con 8 componentes), se optó por un análisis de relevancia basado en Información Mutua y test ANOVA. Este proceso permitió identificar las tres características con mayor poder predictivo: *vconst_corr*, *vconst_2* y *bckgrnd_vdc1*. Esta reducción al "Top-3" facilitó la implementación de un circuito compacto de solo 3 qubits, minimizando el ruido cuántico.

C. Fase Experimental de Balanceo

Con el objetivo de analizar exhaustivamente cómo el desbalanceo de clases afectaba a la capacidad predictiva del clasificador, se llevó a cabo una fase experimental utilizando la técnica de sobremuestreo sintético SMOTE (*Synthetic Minority Oversampling Technique*) [16]. Mediante este procedimiento, se generaron muestras artificiales para la clase minoritaria hasta alcanzar una distribución equitativa de 494 instancias por clase (988 en total). Esta etapa se concibió como un estudio comparativo para evaluar si el equilibrio artificial de la base de datos permitía al modelo capturar mejor los patrones de fallo.

D. Proceso de Entrenamiento y Validación

Para determinar la topología óptima del VQC, se aplicó una búsqueda exhaustiva en cuadrícula (*GridSearch*) sobre el espacio de hiperparámetros. En la Tabla I se recogen las opciones exploradas: tipo de optimizador, Feature Map, *Ansatz*, función de pérdida y tasas de aprendizaje.

Para optimizar el tiempo de cómputo en esta fase exploratoria, cada ejecución se limitó a 25 iteraciones para los optimizadores basados en gradiente (por ejemplo, ADAM [17]) y a 50 iteraciones para los optimizadores sin gradiente (COBYLA) [18]. Todos los detalles, métricas y tiempos de ejecución se registraron exhaustivamente, tal como se ilustra en la Figura 3.

Se consideraron los dos escenarios previamente descritos: (1) utilizando el conjunto original altamente desbalanceado, y (2) utilizando el conjunto balanceado mediante SMOTE. En total se realizaron 672 ejecuciones por escenario (1344 en total), de las cuales se seleccionaron las tres mejores en cada caso, sumando seis configuraciones finales. El diagrama de flujo de este proceso de validación se detalla en la Figura 4.

Estas seis configuraciones seleccionadas se sometieron a un reentrenamiento con un mayor número de iteraciones, hasta 300 para optimizadores sin gradiente y 100 para los basados en gradiente, ya que, tras la fase exploratoria, incrementar las iteraciones favorece la convergencia del modelo [19].

E. Arquitectura del Modelo Final

Tras el proceso de reentrenamiento, se determinó la arquitectura más robusta y con mayor capacidad de generalización para el conjunto de datos original, prescindiendo del balanceo sintético. El modelo definitivo emplea un total de 3 qubits, correspondientes a las características seleccionadas en las fases previas. En concreto, la configuración cuántica que alcanzó el mejor rendimiento global utiliza la plantilla *ZFeatureMap* con dos repeticiones para la codificación de los datos clásicos, seguida de un *Ansatz* tipo *EfficientSU2*, también configurado con dos repeticiones. El ajuste de los parámetros se llevó a cabo empleando el optimizador clásico L-BFGS-B con una tasa de aprendizaje de 0.0010, estableciendo un límite de 100 iteraciones para garantizar la convergencia óptima del sistema.

En la Figura 5 se muestra la estructura del circuito empleado. Este implementa una codificación de características donde cada qubit recibe información de entrada mediante una puerta de Hadamard seguida de una puerta de fase, equivalente a una rotación $R_Z(2x_i)$. Después de la codificación, se aplica el *Ansatz EfficientSU2*, repitiendo dos veces un bloque compuesto por rotaciones R_Y y R_Z parametrizadas sobre cada qubit, junto con una secuencia lineal de

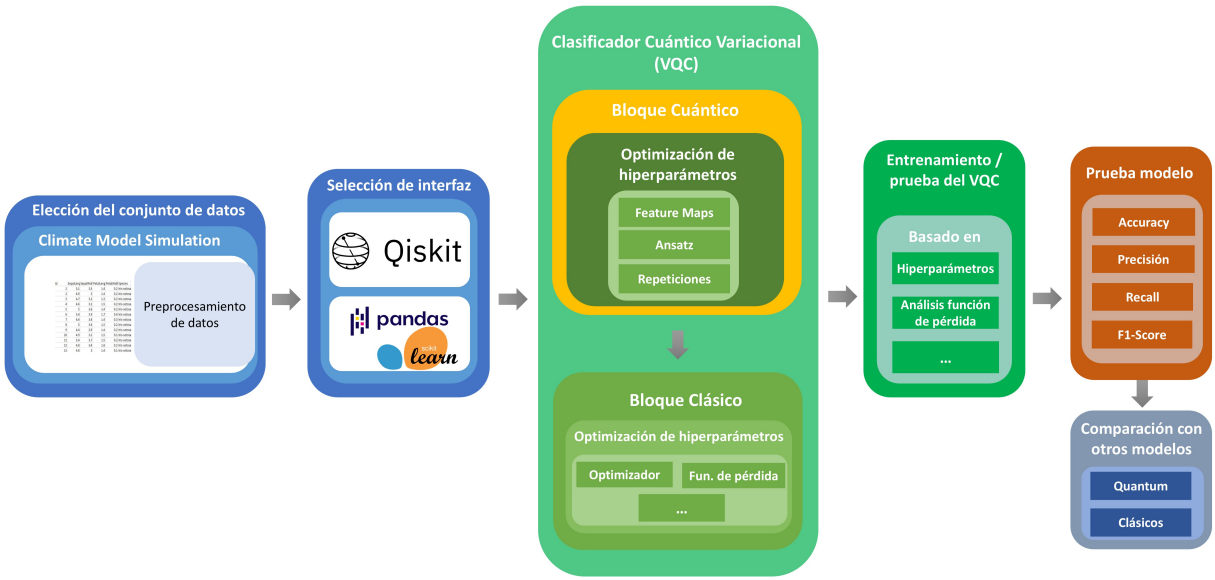


Fig. 2: Flujo completo del VQC propuesto para el conjunto de datos elegido.

Tabla I: Hiperparámetros evaluados en la búsqueda de GridSearch. Un guion (–) indica que no se evaluaron configuraciones adicionales para dicho hiperparámetro.

Hiperparámetro	Valor 1	Valor 2	Valor 3	Valor 4
Optimizador	SPSA	COBYLA	Adam	L-BFGS-B
Función de pérdida	cross_entropy	weightedcross_error	absolute_error	squared_error
Feature Map	ZFeatureMap	ZZFeatureMap	PauliFeatureMap	–
Ansatz	EfficientSU2	RealAmplitudes	–	–
Reps. Feature Map	1	2	–	–
Reps. Ansatz	2	3	–	–
Learning Rate	0.05	0.10	0.001	0.0001

	feature_map	ansatz	rep_fm	rep_var	learning_rate	loss_func	accuracy	precision	recall	f1	durat
26	ZFeatureMap	EfficientSU2	2	2	0.0001	cross_entropy	0.944444	0.971429	0.666667	0.735294	124.977
74	ZFeatureMap	EfficientSU2	2	2	0.0001	squared_error	0.944444	0.971429	0.666667	0.735294	124.779
24	ZFeatureMap	EfficientSU2	1	2	0.0001	cross_entropy	0.925926	0.962617	0.555556	0.580583	154.093
25	ZFeatureMap	EfficientSU2	1	3	0.0001	cross_entropy	0.925926	0.962617	0.555556	0.580583	153.667
73	ZFeatureMap	EfficientSU2	1	3	0.0001	squared_error	0.925926	0.962617	0.555556	0.580583	167.836
...	...	...	...	...	...	...	...	...	...	...	
94	PauliFeatureMap	RealAmplitudes	2	2	0.0001	squared_error	0.731481	0.521107	0.550505	0.505604	44.412
14	ZZFeatureMap	RealAmplitudes	2	2	0.0001	weightedcross_entropy	0.648148	0.595745	0.808081	0.541964	73.403
15	ZZFeatureMap	RealAmplitudes	2	3	0.0001	weightedcross_entropy	0.583333	0.552759	0.671717	0.475332	67.544
23	PauliFeatureMap	RealAmplitudes	2	3	0.0001	weightedcross_entropy	0.564815	0.580357	0.762626	0.482832	85.553
22	PauliFeatureMap	RealAmplitudes	2	2	0.0001	weightedcross_entropy	0.546296	0.561813	0.702020	0.460825	68.824

Fig. 3: Matriz de resultados de las distintas ejecuciones para cada combinación de hiperparámetros.

puertas CNOT encargadas de generar el entrelazamiento.

Finalmente, se realiza la medición del estado final de los qubits para colapsar la superposición y obtener la predicción del circuito. Esta salida cuántica es interpretada por el sistema clásico para evaluar la función de pérdida *Cross-Entropy* y actualizar iterativamente los parámetros. Cabe destacar que la configuración óptima resultante no solo se evaluó en simulación ideal, sino que se probó empíricamente en un dispositivo cuántico real de IBM Quantum para medir su rendimiento sobre hardware físico y analizar la afectación del ruido.

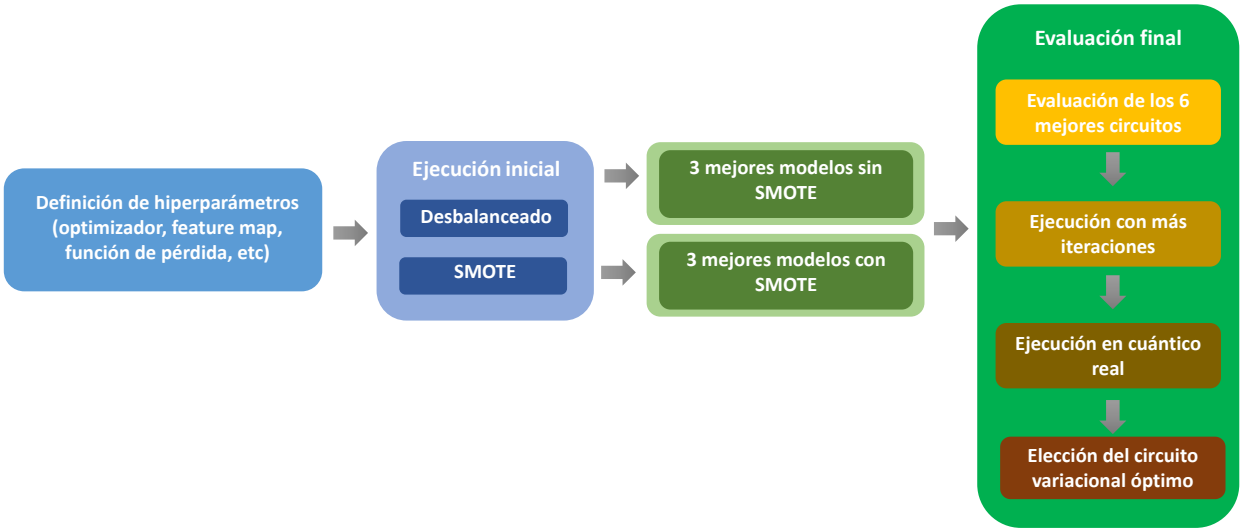


Fig. 4: Diagrama de flujo del proceso de entrenamiento y validación del modelo VQC.

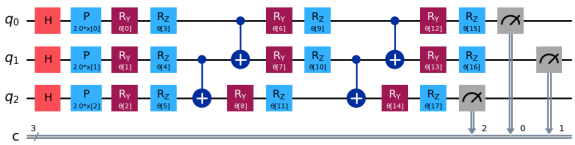


Fig. 5: Topología del circuito cuántico variacional empleado para la tarea de clasificación.

IV. Resultados y discusión

A. Métricas de Evaluación

En escenarios con un desbalanceo de clases tan severo (91.48 % frente a 8.52 %), el uso aislado de la métrica *Accuracy* resulta engañoso. Un modelo trivial que prediga sistemáticamente la clase mayoritaria (simulación exitosa) obtendría una exactitud superior al 91 % sin identificar correctamente ni un solo fallo climático. Por este motivo, la evaluación del rendimiento se ha centrado en métricas más robustas frente a clases desequilibradas: la Precisión, el *Recall* y el F1-Score (la media armónica de ambas). La maximización del F1-Score se ha establecido como objetivo prioritario para garantizar un equilibrio óptimo entre la detección efectiva de anomalías (verdaderos positivos) y la minimización de falsas alarmas (falsos positivos).

B. Evaluación del Modelo Propuesto

El rendimiento del VQC con la arquitectura final configurada fue evaluado inicialmente sobre el conjunto de datos original sin aplicar técnicas de balanceo sintético.

Bajo estas condiciones naturales de desbalanceo, el modelo propuesto demostró una notable capacidad de generalización. Alcanzó un *Accuracy* del 95.37 % y una Precisión del 97.60 %. Su tasa de detección de la clase minoritaria (*Recall*) se situó en el 72.22 %, consolidando un F1-Score del 79.54 %. Estos valores confirman que el circuito de 3 qubits diseñado es capaz de identificar patrones no lineales subyacentes a las configuraciones que provocan fallos, sin verse excesivamente sesgado por el volumen de la clase mayoritaria.

Para contrastar estos resultados y aislar el efecto del desbalanceo, se evaluó la misma arquitectura sobre el conjunto balanceado artificialmente mediante SMOTE. En este escenario, el *Accuracy* descendió al 80.40 %, un efecto esperado al aumentar drásticamente el peso de la clase minoritaria en la función de coste. Sin embargo, la sensibilidad del modelo se incrementó de forma sustancial, logrando un *Recall* del 96.96 % y un F1-Score del 90.28 %. Esto demuestra que, en contextos operativos donde omitir un fallo en la simulación resulte computacionalmente inasumible, el preprocesamiento con SMOTE ofrece un modelo de altísima detección.

C. Comparativa con el Estado del Arte

Para validar empíricamente las contribuciones de esta investigación, se ha llevado a cabo un estudio comparativo frente a los resultados reportados en la literatura para el mismo conjunto de datos *Climate Model Simulation Crashes*. La comparativa incluye enfoques clásicos de alta capacidad como SVM [1] y ensambles avanzados de árboles de decisión (NODE, CatBoost) [3], así como alternativas cuánticas recientes basadas en QKE [20] (ver Tabla II).

En la Tabla II, se observa que el modelo SVM de Lucas et al. [1] es el modelo clásico más competitivo para este conjunto de datos, alcanzando un *Accuracy* del 96.70 %. Sin embargo, el enfoque propuesto en esta investigación logra un F1-Score superior (79.54 % frente al 75.00 % del SVM), lo que demuestra una mayor capacidad para equilibrar la detección de fallos en un entorno de datos altamente desbalanceados. Además, el modelo SVM clásico depende de la totalidad de las dimensiones del problema, mientras que nuestro método optimiza el espacio de características de manera más eficientze.

Respecto a los enfoques cuánticos, los métodos más relevantes en la literatura son los de Raubitzek et al. [20], específicamente los modelos QKE-Z y QKE-SL, que utilizan estimación de kernel cuántico. Nuestro objetivo es analizar qué parámetros mejo-

ra la arquitectura VQC-Climate en comparación con estos referentes del estado del arte.

En términos de *Accuracy*, el modelo propuesto alcanza un 95.37 %, superando significativamente al modelo QKE-Z (84.40 %) y al modelo QKE-SL (91.30 %). En cuanto a la Precisión, nuestro modelo logra un 97.60 %, lo que garantiza una mínima tasa de falsos positivos en la predicción de éxitos de simulación. En el análisis del *Recall*, nuestra propuesta alcanza un 72.22 %, superando tanto al SVM clásico (64.30 %) como a los modelos QKE (56.40 %). Asimismo, el modelo alcanza un F1-Score de 79.54 %, superando el 58.80 % del mejor modelo cuántico previo (QKE-SL) y el 76.60 % del modelo clásico de ensamble NODE [3].

Para analizar el rendimiento de la propuesta con más detalle, la Figura 6 presenta la matriz de confusión. En ella se observa que el modelo clasifica correctamente 99 instancias de la clase mayoritaria y 5 instancias de la clase minoritaria (fallos críticos), cometiendo únicamente 4 errores de clasificación en la clase minoritaria.

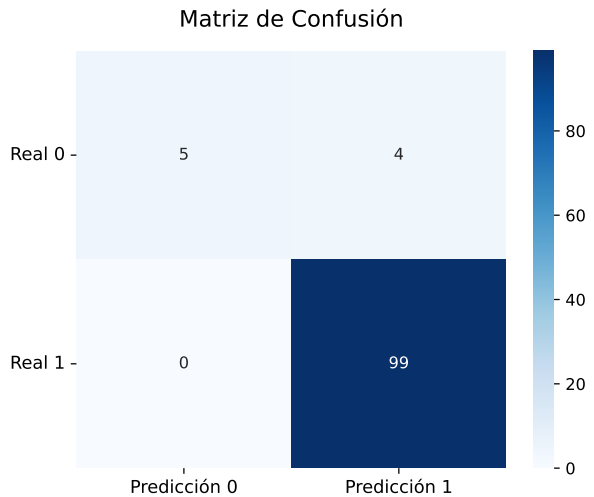


Fig. 6: Matriz de confusión del modelo propuesto, donde se muestra el número de instancias clasificadas correctamente e incorrectamente. El valor '0' representa las simulaciones exitosas y el '1' representa los fallos en la simulación climática.

Una ventaja clave de este modelo es su eficiencia y viabilidad técnica: solo requiere 3 qubits para procesar la información, frente a las 18 dimensiones que manejan los modelos competitivos del estado del arte. Esta reducción drástica de la complejidad permite que el circuito mantenga una profundidad baja, lo que minimiza el impacto del ruido y la decoherencia. Esto convierte a este enfoque en una solución robusta y aplicable en la actual era NISQ. Como se muestra en la Tabla II, los resultados obtenidos a partir de la ejecución en un ordenador cuántico real confirman que el modelo propuesto demuestra un rendimiento altamente competitivo, destacando especialmente por su superioridad en la métrica clave F1-Score.

V. Conclusiones y trabajo futuro

En esta investigación se ha llevado a cabo un análisis exhaustivo sobre la aplicación de VQCs en conjuntos de datos eviy no lineales. Para ello, se ha utilizado el conjunto de datos *Climate Model Simulation Crashes*, cuyo objetivo es clasificar entre exitosas o fallidas las configuraciones empleadas para simular modelos de cambio climático. Esta predicción temprana es fundamental para mejorar la fiabilidad de las simulaciones y permitir a los investigadores identificar y corregir posibles fallos, optimizando así los recursos computacionales.

Durante el estudio, se han puesto en evidencia las limitaciones de métricas convencionales como el *Accuracy*, las cuales pueden resultar engañosas al beneficiar desproporcionadamente a la clase mayoritaria. Por consiguiente, la evaluación se ha fundamentado en métricas más robustas frente al desbalanceo, tales como la Precisión, el *Recall* y, especialmente, el F1-Score, complementadas con un análisis detallado de las matrices de confusión para obtener una perspectiva completa y real de la capacidad del modelo para clasificar correctamente ambas clases.

A través de una búsqueda exhaustiva en cuadrícula (*GridSearch*), se evaluaron diversas combinaciones de hiperparámetros utilizando distintos optimizadores, funciones de pérdida, mapas de características y circuitos parametrizados. Los resultados obtenidos han revelado que la configuración óptima está conformada por el optimizador L-BFGS-B, la función de pérdida *Cross-Entropy*, el mapa de características *ZFeatureMap* y el circuito *EfficientSU2*. Este modelo destacó al alcanzar una precisión sobresaliente del 97,60 % y un equilibrio adecuado entre falsos positivos y negativos, reflejado en un F1-Score del 79,54 %.

Además, al comparar estos resultados con modelos clásicos como las SVM y otros enfoques cuánticos publicados previamente, la propuesta desarrollada basada en VQC ha demostrado claras ventajas. Si bien la aplicación de técnicas de balanceo de clases (SMOTE) mejoró significativamente métricas como el *Recall* (96,90 %) y el F1-Score (90,20 %), el modelo VQC original sin balanceo también alcanzó resultados muy competitivos, confirmando que posee una capacidad intrínseca para detectar correctamente la clase minoritaria.

Cabe destacar que estos circuitos compactos, al requerir un número mínimo de recursos, pudieron ejecutarse de manera exitosa en hardware cuántico real de IBM sin deteriorar las métricas obtenidas en simulación. Esto es posible gracias a que la reducida profundidad del circuito disminuye considerablemente los efectos negativos del ruido y la decoherencia propios de la actual era NISQ.

Como trabajo futuro, se plantea testear el comportamiento, la escalabilidad y la robustez del clasificador cuántico desarrollado utilizando otras bases de datos altamente desbalanceadas en diversos dominios de aplicación.

Agradecimientos

Este trabajo ha sido financiado por los proyectos PID2021-123278OB-I00 y PDC2022-133370-I00, financiados por MCIN/AEI/10.13039/501100011033 y por "FEDER Una manera de hacer Europa", así

Tabla II: Comparativa de resultados de diferentes modelos cuánticos y clásicos para el conjunto de datos propuesto. La columna "Cuántico" indica si el modelo utiliza un enfoque de computación cuántica, mientras que "No" en la columna "Balanceo" indica que no se especificó si se aplicaron técnicas de balanceo de datos. Un guion (–) indica que la métrica correspondiente no fue reportada. Las mejores métricas se han destacado en negrita. Los resultados del modelo propuesto se obtuvieron mediante ejecución directa en hardware cuántico real.

Autor	Modelo	Cuántico	Balanceo	Accuracy (%)	Precisión (%)	Recall (%)	F1-Score (%)
[20]	QKE-Z	Sí	No	84.4	55	55.9	55.4
	QKE-SO	Sí	No	35.1	53	58	32.3
	QKE-SL	Sí	No	91.3	71.2	56.4	58.8
	QKE-SU	Sí	No	90.7	45.6	49.6	47.5
	QKE-GL	Sí	No	91.3	45.6	50	47.7
	QKE-U	Sí	No	90.7	45.6	49.6	47.5
	QKE-O	Sí	No	35.1	53	58	32.3
	CatBoost	No	No	91.3	45.6	50	47.7
[3]	GRANDE	No	No	87.5	–	–	75.8
	XGB	No	No	76.2	–	–	78.1
	NODE	No	No	77.3	–	–	76.6
[1]	SVM	No	No	**96.7**	90	64.3	75
Propuesta	VQC	Si	No	95.3	**97.6**	72.2	79.5
	VQC	Si	Si	80.4	84.2	**96.9**	**90.2**

como por el Plan de Investigación y Transferencia 2024-2025 de la Universidad de Almería.

Referencias

[1] D. D. Lucas, R. Klein, J. Tannahill, D. Ivanova, S. Brandon, D. Domyancic, and Y. Zhang, "Failure analysis of parameter-induced simulation crashes in climate models," *Geoscientific Model Development*, vol. 6, no. 4, aug 2013.

[2] D. Lucas, R. Klein, J. Tannahill, D. Ivanova, S. Brandon, D. Domyancic, and Y. Zhang, "Climate model simulation crashes [dataset]," UCI Machine Learning Repository, 2013.

[3] S. Marton, S. Lüdtke, C. Bartelt, and H. Stuckenschmidt, "Grande: Gradient-based decision tree ensembles for tabular data," *arXiv preprint arXiv:2309.17130*, sep 2023.

[4] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*, Cambridge University Press, Cambridge, UK, 10th edition, dec 2010.

[5] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.

[6] L. M. Donaire, G. Ortega, F. Orts, E. M. Garzón, and E. Filatovas, "A hybrid quantum-classical approach for liver disease detection using quantum machine learning," *Engineering Applications of Artificial Intelligence*, vol. 164, pp. 113240, 2026.

[7] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, pp. 195–202, aug 2018.

[8] D. V. Babukhin and W. V. Pogosov, "The effect of quantum noise on algorithmic perfect quantum state transfer on nisq processors," *Quantum Information Processing*, vol. 21, no. 1, pp. 7, 2022.

[9] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, "Parameterized quantum circuits as machine learning models," *Quantum Science and Technology*, vol. 4, no. 4, pp. 043001, nov 2019.

[10] P. Rebentrost, M. Mohseni, and S. Lloyd, "Quantum support vector machine for big data classification," *Physical Review Letters*, vol. 113, no. 13, pp. 130503, sep 2014.

[11] E. F. Combarro, S. González Castillo, and A. Di Meglio, *A Practical Guide to Quantum Machine Learning and Quantum Optimization: Hands-on Approach to Modern Quantum Algorithms*, Packt Publishing Ltd, 1 edition, 2023.

[12] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, "Supervised learning with quantum-enhanced feature spaces," *Nature*, vol. 567, no. 7747, pp. 209–212, 2019.

[13] M. Schuld and N. Killoran, "Quantum machine learning in feature hilbert spaces," *Physical Review Letters*, vol. 122, no. 4, pp. 040504, 2019.

[14] M. Amin, E. Andriyash, J. Rolfe, and et al., "Quantum boltzmann machine," *Physical Review X*, vol. 8, no. 2, pp. 021050, June 2018.

[15] D. Ewald, "The proposal of a fully quantum neural network and fidelity-driven training using directional gradients for multi-class classification," *Electronics*, vol. 14, no. 11, pp. 2189, 2025.

[16] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[18] M. JD. Powell, "A direct search optimization method that models the objective and constraint functions by linear interpolation," in *Advances in optimization and numerical analysis*, pp. 51–67. Springer, 1994.

[19] J. Hunout, S. Laizet, and L. Iannucci, "Variational quantum algorithm based on lagrange polynomial encoding to solve differential equations," *Physical Review A*, vol. 111, pp. 062404, 2025.

[20] S. Raubitzek, S. Schrittwieser, A. Schatten, and K. Mallinger, "Quantum inspired kernel matrices: Exploring symmetry in machine learning," *Physics Letters A*, vol. 525, pp. 129895, nov 2024.

Evaluación de prestaciones (2/3)

Scalable Spike Transmission in Brain Network Simulations

Mario Ibáñez [1], Marvin Kaster [2], Borja Pérez[1], Han Lu[3], Fabian Czappa[2], Sandra Díaz Pier[3], Jose Luis Bosque[1], Felix Wolf[2], Thorsten Hater[3]

Abstract— **Large-scale simulations of biological neural networks model cells, their connections, and the exchange of information through action potentials, or spikes. As network size grows, spike transmission over MPI becomes a major bottleneck. The traditional approach broadcasts all spikes to all ranks, simplifying preprocessing but introducing costly postprocessing and becoming inefficient at scale.**

To address this, we propose two complementary methods. The first, Targeted Spike Delivery, is a generic, network-independent strategy that sends spikes only to their intended receivers, reducing redundant communication and eliminating expensive postprocessing. The second, Hierarchical Neighborhood Communication, exploits cell positions and MPI neighborhood communication. By partitioning the network into spatial subdomains, it leverages the predominance of local neuronal connectivity to further improve scalability.

Our methods achieve substantial performance gains, with speedups of up to 2.5 on CPUs, 6 on GPUs, and 3.7 over the Baseline strategy when neighborhood communication is applied. These results improve the efficiency of large-scale brain simulations and support their execution on current and future exascale systems.

Key Words— **Brain simulation, HPC, Scalability, Communication, Spiking transmission, neighborhood communication.**

I. Introduction

UNderstanding the human brain remains one of the grand scientific challenges. Because experiments cannot simultaneously monitor large and complex brain cell populations in detail, computational modeling has become an essential tool across multiple levels of resolution: from neural mass models [1], to point neuron models [2], cable cell models [3, 4], and molecular dynamics models [5]. These approaches are used either independently or in co-simulation [6, 7] to study learning and memory [8, 9], disease and rehabilitation [10, 11], and emerging applications such as bio-inspired AI and personalized treatments for disorders including Parkinson's disease, epilepsy [12], neuropathy [13], and Alzheimer's disease [14].

Current simulation technology still falls short of the long-term goal of whole-brain simulation. The adult human brain contains about 86 billion neurons [15]; the cerebral cortex alone has around 16 billion cells [16] and about 10000 synapses per cell [17]. Simulating networks at this scale therefore requires handling extremely large connectivity data efficiently.

Even with a minimal 16B per connection, storing the full connectome would require about 2.5PB of memory, far beyond the capacity of current systems. On JUWELS Booster, for example, this would correspond to about 640GB of connection data per GPU, exceeding both device and host memory limits.

However, at realistic connection degrees, increasing the number of cells makes the network increasingly sparse. For a network with k synapses per cell and n cells distributed over r ranks, assuming uniformly distributed targets, the probability that a cell connects to a given rank is $Q = 1 - (1 - 1/r)^k$. As shown in Fig. 1, this probability decreases with system scale, indicating that modern large-scale simulations operate in a sparse regime even at cortical connection densities.

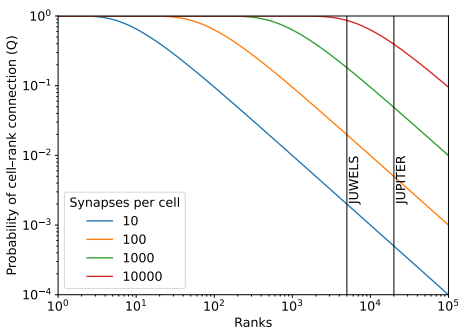


Fig. 1: Probability of a connection between a cell and a given rank, Vertical lines mark the full-scale configurations of the European JUWELS and JUPITER systems. Setting the synapse count per cell $k = 10000$ yields $Q \sim 0{,}91$ for JUWELS Booster ($r = 4000$) and $Q \sim 0{,}35$ for JUPITER Booster ($r = 24000$). Both assume the full system, using all GPUs, and one rank per GPU.

Besides the memory demands of storing connectivity, large-scale brain simulations require efficient communication. Each action potential generated by a neuron may need to be delivered to many targets, often distributed across different MPI ranks. The number of communicated events grows with the number of neurons, synapses, and the chosen communication scheme, making communication a major runtime bottleneck [18].

This challenge is especially relevant on modern large-scale clusters and GPU-accelerated systems [19], where increasing system size also increases communication and synchronization costs. Since traditional spike communication schemes were not designed for this scale, improving scalability is essential. To address this, this work proposes two alternative communication strategies in Arbor to improve

[1]Dept. of Computer Engineering and Electronics, University de Cantabria, Santander, Spain
[2]Dept. of Computer Science, Technical University of Darmstadt, Darmstadt, Germany
[3]Simulation and Data Lab Neuroscience, Jülich Supercomputing Centre (JSC), Jülich, Germany

performance in large-scale simulations:

- *Targeted Spike Delivery* (TSD) a generic scheme, applicable to any spike-based simulator, which reduces transfers by delivering spikes only to the ranks that require them.
- *Hierarchical Neighborhood Communication* (HNC) a domain-specific approach that exploits knowledge of cell positions, enabling a topology-aware strategy that further reduces communication by leveraging locality in neuronal connectivity.

Together, these approaches represent a fundamental shift in the way spike data is exchanged, lowering communication costs and enabling significant improvements in scalability and efficiency on modern HPC systems. In our first experiment, TSD achieves substantial speedups of 2.5 on the CPU (10,000 ranks) and 6 on the GPU (3,500 ranks). In the second combined benchmark, HNC surpasses the Baseline by a factor of 3.7 and TSD by 2 on the CPU (4,096 ranks).

II. Background

Arbor is an open-source library developed in the Human Brain Project [20] for large-scale simulations of biophysical neuron networks. Unlike point neuron models [2], biophysical models explicitly represent dendrites and axons, requiring the solution of the cable equation [3] to capture the spatial and temporal propagation of electrical signals and the location-dependent effects of synaptic inputs.

Implemented in C++17, Arbor supports GPU acceleration, SIMD optimizations, multi-core parallelism, and MPI-based distributed spike communication. Cells are organized into groups and advanced as tasks on CPU or GPU backends, while a work-stealing scheduler balances load dynamically. Because performance depends on both cell complexity and network connectivity, a key development goal is to optimize execution and communication across diverse simulation scenarios.

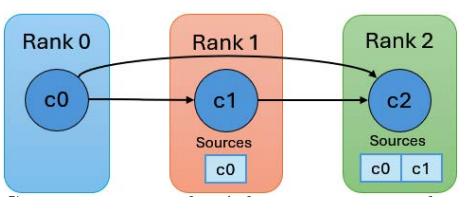


Fig. 2: Connection example. Arbor represents each neuron's synaptic connectivity by storing its incoming synapses within the neuron's internal structure. This design reduces the memory footprint in large simulations and decouples spike communication from the specific network topology.

As a direct consequence of Arbor's connectivity representation (Fig. 2), the communication of spiking activity between processes requires a mechanism that can efficiently collect and distribute spikes. Fig. 3 illustrates the communication pattern, where Arbor collects spikes locally and broadcasts them over the network. However, this flexibility comes at a cost: every process receives the complete set of spikes generated across all ranks, even though only a small fraction is locally relevant. Consequently, each rank must inspect hundreds or thousands of spike events to identify those targeting its own neurons, discarding the rest. In our largest experiments with 10000 CPU ranks, this overhead led to a 2.73× increase in spike postprocessing, consuming resources that could otherwise be dedicated to the simulation.

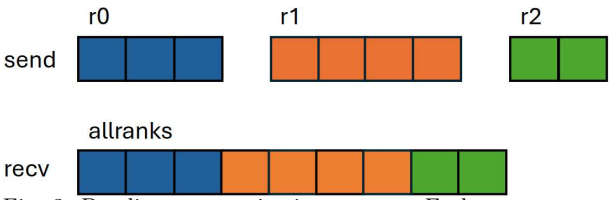


Fig. 3: Baseline communication pattern. Each process contributes a different number of elements, which are gathered and concatenated in rank order. The resulting buffer, identical across all ranks, contains the complete dataset composed of the contributions from all processes. By avoiding auxiliary data structures and relying only on local spike collection, it offers an effective, memory-efficient, yet scalable mechanism for spike exchange in large-scale network simulations.

A second limiting factor is the cost of the interconnection. In Arbor, this is handled using the `MPI_Allgatherv` collective communication, which broadcasts all spikes to every rank. This pattern places a heavy load on the network in a very short time window, saturating the interconnect and blocking all ranks (due to the collective operation) until the communication completes, which results in high latencies and poor bandwidth utilization. At HPC scales, these effects turn network communication into the dominant bottleneck, ultimately restricting the scalability of the simulation.

III. Pattern Communication Optimizations

A. Targeted Spike Delivery

To ensure future scalability of Arbor, we propose a new communication pattern called Targeted Spike Delivery (TSD). In this scheme, each process sends spikes only to the ranks that require them, rather than broadcasting all events to all participants. During an initial preprocessing phase, the set of target cells is determined from the connectivity of the neural network, so that at runtime every rank can directly prepare the spike data needed by its neighbors. This eliminates redundant communication and avoids the costly postprocessing step of filtering irrelevant spikes. By exploiting the network's connectivity, TSD focuses communication strictly on relevant targets, thereby reducing overhead and enabling scalable spike exchange in large-scale network simulations as visualized in Fig. 4.

A requirement of TSD is that the exact destinations of spikes are known in advance. As explained in Sec. II, each cell has explicit knowledge of its incoming connections (Fig. 2), but not of its outgoing ones. This creates a challenge for TSD, since each rank must determine which remote processes the locally generated spikes need to be sent to.

To address the challenges of the new communication pattern, a preprocessing stage is introduced at

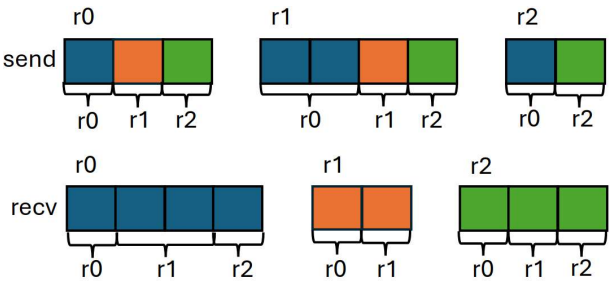


Fig. 4: Targeted Spike Delivery example. Each rank prepares send buffers with variable amounts of data for different peers, and after the collective exchange, every process obtains only the biologically meaningful spikes addressed to it; e.g., rank 1 in this case obtains just two. This contrasts with Fig. 3, the original Arbor communication pattern, where every rank receives all 9 spikes, even if they are not relevant. As a result, both communication volume and computational overhead are significantly reduced, enabling improved scalability, especially as the number of ranks, neurons per node, and connections per cell increase.

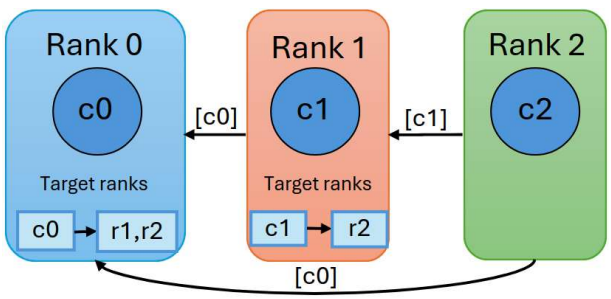


Fig. 5: Initialization of communication based on the links shown in Fig. 2, rank 2 holds the local cell $c2$ and constructs a vector listing all its cell sources ($c0$ and $c1$). This vector is then communicated to the other ranks; rank 0 receives $c0$ and rank 1 receives $c1$. Subsequently, each rank constructs a lookup structure. For instance, rank 0 maps its local cell $c0$ to the destination ranks $r1, r2$, whereas rank 1 maps its local cell $c1$ to the destination rank $r2$.

simulation start. Each MPI rank collects the connectivity information of its local cells by building a vector of the global identifiers of all ranks of its outgoing connections. Given the external ranks, this vector encodes which local cells can send spikes, see Fig. 5.

With this information, each rank can generate a mapping structure that resolves the outgoing connections. For every local cell, the rank determines the set of destination ranks that must receive its spikes during the simulation. If there is a rank without outgoing connections (as rank 2), its structure is not generated for its local cells, as any spike produced or received by them is not forwarded to other ranks. This enables spikes to be routed exclusively to the relevant ranks, avoiding unnecessary communication and improving efficiency during runtime.

Once local spikes are collected at each timestep, each rank uses the mapping to identify the peers that require those events. It then builds the send buffer by replicating spikes according to the destination ranks, effectively filtering communication at the source. The buffer is passed to `MPI_Alltoallv`, which delivers the spikes to their targets.

Importantly, although `MPI_Alltoallv` is invoked collectively by all ranks, it does *not* mean that every rank sends data to every other rank. In our implementation, each rank first determines which peers actually host the target neurons for its local spikes and then packs data only for that subset. Peers that are not involved in the current step simply receive no payload, so the exchange can scale in sparse connectivity scenarios. This is the mechanism by which communication is reduced most ranks are filtered out at the source through zero-count destinations. This ensures that communication scales, avoiding unnecessary transfers and reducing postprocessing overhead.

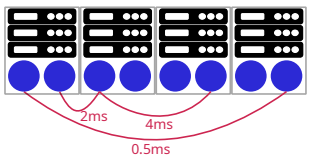
In the BASELINE communication pattern, every rank receives all spikes, so global spike counts are obtained implicitly. With TSD, each rank receives only spikes relevant to its local targets, so global statistics must be computed explicitly, typically with an `MPI_Allreduce`. Although this adds one collective operation per step, its cost is small compared to the savings from avoiding the transmission and processing of irrelevant spikes.

TSD requires additional preprocessing to build mapping structures and communication metadata, but this overhead is incurred mostly during initialization and has little impact on overall runtime or memory usage (see Sec. V). In return, it removes unnecessary communication and processing, improving efficiency and scalability.

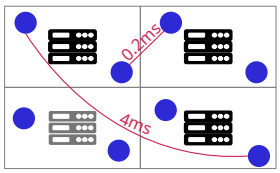
By exploiting network topology, TSD restricts communication to relevant targets and eliminates a major runtime bottleneck. This strategy is not limited to Arbor and can be applied to any spike-exchange–based simulator.

B. Hierarchical neighborhood communication

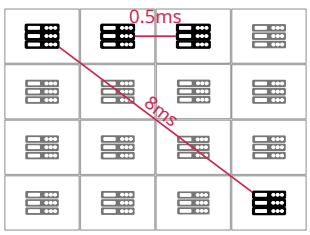
In the brain, neurons are spatially positioned with a high number of local connections and only a few distant ones projecting into other brain areas. Through the spatial distance, spikes travel longer to distant neurons than to close ones. We utilize the spatial connectivity for our next approach. It requires that synaptic delays increase with synaptic length, similar to the brain, and applies to most biological models. As Arbor does not contain information about neurons' positions, we introduce a 3D Euclidean simulation space and assign positions to each neuron. As visualized in Fig. 6, the spatial domain in our simulation is divided into smaller subdomains, each managed by an MPI rank. Since nearby subdomains correspond to smaller synaptic delays, communication with them occurs more frequently than with distant ones. By calculating distances between subdomains and their assigned ranks, we can associate larger synaptic delays with more distant ranks. The synaptic delays determine how much time is available to communicate spikes to their target. Hence, the spatial subdomains enable us to communicate with some subdomains more frequently than with others. MPI neighborhood collectives are, therefore, utilized to communicate with close subdomains locally instead of globally.



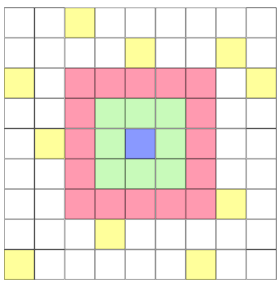
(a) Arbor's default setup. Neurons are not assigned a spatial location and are randomly distributed to MPI ranks. The synaptic delays between the neurons on different ranks are arbitrary.



(b) Euclidean space partitioned into 4 subdomains managed by one MPI rank each (2D example). The synaptic delay between neurons depends on the distance between them.



(c) Minimum synaptic delay between neurons of subdomain pairs increases with their distance.



(d) Communication via MPI neighborhood collective. From most to least frequent: Self (blue), neighbors (green), neighbors of neighbors (red), and other subdomains globally (yellow).

Fig. 6: Network setup in the Euclidean space.

B.1 Network setup

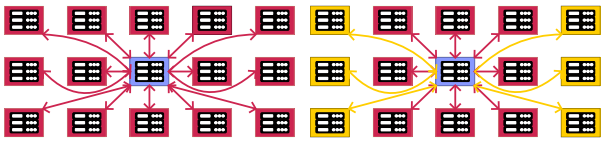
Spatial information are introduced to our network by defining the simulation domain as a 3-dimensional cube with a size of l^3_{domain} in an Euclidean space that is split it into equally sized subdomains. Subdomains are distributed to MPI ranks in round-robin order with a Z-order space-filling curve. Within subdomains, neurons are positioned on a regular grid, ensuring an equal number of neurons per subdomain. To avoid artificial symmetries, random jittering is applied to each neuron position. The maximum displacement is chosen such that a neuron can be shifted at most toward the position of its nearest grid neighbor, ensuring local variation without crossing subdomain boundaries.

Our network is connected based on a distance-dependent probability rule [10], making close connections more likely than distant ones. The relationship is defined in Eq. 1, where d_{ij} denotes the squared Euclidean distance between neurons i and j. The connection probability p_{ij} depends on this distance, with the parameter σ controlling the width of the spatial kernel. Each neuron has k number of incoming and outgoing synapses with a fixed weight of w_{synapse}. As a naïve calculation of the connectivity would have quadratic complexity, we use the software RELeARN that provides an approximation [21, 22] based on Barnes-Hut reducing the complexity to $\mathcal{O}(\backslash \log \backslash)$ with n being the number of neurons. We use the software RELeARN [1] [21, 22] for the subdomain partitioning, neuron position assignment, and connectivity formation. As synaptic delays correlate with the axonal length, we define the delay t_{ij} between neuron i and j, where v_{synapse} is the axonal conduction velocity.

$$\begin{aligned} d_{ij} &= (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 \\ p_{ij} &= \exp\left(-\frac{d_{ij}}{\sigma^2}\right), \quad t_{ij} = 1 + \frac{\sqrt{d_{ij}}}{v_{\text{synapse}}} \end{aligned} \tag{1}$$

[1] https://github.com/tuda-parallel/RELeARN

B.2 Hierarchical communication



(a) Communication over `MPI_Allgather`. The node sends and receives data from all other nodes.

(b) Communication over `MPI_Neighborhood_Allgather` to direct neighbors (level 1, red) and neighbors of neighbors (level 2 yellow).

Fig. 7: Visualization of communication strategies from the view of a single node (blue, center).

With ranks mapped to spatial subdomains, the minimum synaptic delay between two ranks depends on their distance (Fig. 6), motivating a communication scheme that adapts exchange frequency to spatial proximity. We therefore define communication levels: at level l, a subdomain dom_z communicates with subdomains located l hops away, excluding those already covered at lower levels.

Let $\tau_{\text{domains}}(z, z')$ denote the minimum synaptic delay between subdomains dom_z and $\text{dom}_{z'}$. For each level l, we compute the minimum delay τ_{domain} from subdomain z to all domains in that level, and define the global minimum τ_{level} as the smallest such value across all subdomains. The communication frequency at level l is then set to $\tau_{\text{comm},l} = \tau_{\text{level}}/2$. Since delays increase with distance, higher levels communicate less frequently.

Neighborhood communication is limited to a maximum level $l_{\max}$, because connectivity decreases with distance and beyond that point a single global communication becomes more efficient than multiple levels. For this global phase, we evaluate two variants: HNC combined with the Baseline communication pattern, and HNC+TSD.

B.3 Communication implementation

Our hierarchical communication scheme leverages MPI's neighborhood collectives to minimize data transfer and synchronization overhead. Instead of involving all ranks, communication is restricted to a predefined set of neighbors, significantly reducing network traffic. We use `MPI_Neighbor_Allgather` to enable simultaneous data exchange between neighbors at each level of the hierarchy. Graphs representing neighbor relationships are used to determine communication partners. Because MPI neighborhood collectives only supports communication between direct neighbors, we define a separate graph

for each communication level in our hierarchical scheme. At level 1, each rank is connected to its immediate neighboring subdomains.

At level 2, each rank also connects to the neighbors of its neighbors, and so forth. Our communication scheme, therefore, as visualized in Fig. 7, sends the most data and most frequently on level 1, which also has the smallest number of recipients (26 ranks in 3D). Less data is sent less frequently to a higher number of recipients (124 ranks on level 2). This, combined with the efficiency of neighborhood communication, is the foundation of our performance improvements.

IV. Methodology

A. Experimental setup

All experiments were performed on the JUWELS supercomputer and its GPU-accelerated module, JUWELS Booster, at the Jülich Supercomputing Centre (JSC)[23]. Their main characteristics are summarized in Tab. I. To evaluate TSD, we used the *Busyring Benchmark*, a lightweight test composed of interconnected rings of spiking cells that emphasizes communication costs while keeping local dynamics simple. Its parameters are listed in Tab. II.

The benchmark considers two neuron models: *simple* and *complex*. The *complex* model, based on the Allen Cell Types example [24], is more detailed and computationally intensive, making it well suited for GPUs. The *simple* model has lower computational cost and is therefore mapped to CPUs.

We compared TSD and the Baseline communication pattern through weak-scaling experiments, using up to 450 CPU nodes and 850 GPU nodes. This setup reflects the goal of scaling simulations to larger brain regions as more resources become available [19]. On CPUs, we used 24 ranks per node, two threads per rank, and 1024 cells per rank, for up to 11059200 simulated cells. On GPUs, we used one rank per GPU and 12000 cells per rank, reaching 40800000 cells. The workload per rank was fixed at the point giving the best Baseline time-to-solution and then used for all methods.

All experiments used 1000 connections per cell, with additional runs at 500 and 2000 to study the effect of communication load. MPI was used for inter-rank spike exchange in all distributed runs. In the GPU configuration, spike buffers were staged through host memory before and after each MPI communication phase.

The HNC approach is less general than TSD, requiring assumptions such as an underlying Euclidean space and thus preventing a direct comparison. We simulate a network of simple cells as described with 1024 neurons per rank and 1024 incoming synapses per neuron. Each neuron receives input from a separate Poisson spike train with a rate of f_{poisson} and a synaptic weight of w_{poisson}. The simulation runs for 1 s with a time step of 0.25 ms.

For the HNC configuration, all 48 physical cores per node were utilized by assigning one MPI rank per core, leading to a total of up to 4096 ranks across 86 nodes. Weak-scaling benchmarks were performed by increasing the number of ranks, while keeping the number of neurons per rank constant.

The size of the simulation domain also remains constant, so that the maximum possible synaptic delay between neurons remains the same regardless of the number of ranks. For our evaluation, we compare four approaches: Baseline pattern communication, TSD, HNC, and HNC+TSD. As this setup requires more memory than the busyring, we are limited to a maximum of 4096 ranks. A summary of simulation parameters can be found in Tab. III. The key aspects to be evaluated through these experiments are the scalability of the application, energy consumption, memory usage, and communication efficiency. To this end, the following metrics have been selected:

- Execution time: measured in Arbor and split into initialization time, simulation time, and total time.
- Energy consumption: obtained from JUWELS job reports in watt-hours (Wh) and averaged across nodes.
- Memory usage: reported by JUWELS in gigabytes (GB) as the mean CPU and GPU memory per node.
- Bandwidth usage: obtained from JUWELS statistics in megabytes per second (MB/s) and averaged across nodes.
- Transmitted bytes: computed as the number of MPI bytes sent multiplied by the number of targets.

V. Results

In this section, we present the results of the experiments carried out to evaluate the two proposed approaches: the use of TSD to improve scalability in large-scale communication, and the application of neighborhood communication to reduce overheads by exploiting locality in an Euclidean space.

A. Targeted Spike Delivery (TSD) Experiments

Fig. 8 shows the evolution of execution time as the system size (number of processes) and the problem size (amount of cells) increase, splitting in the initialization time and the simulation time for both CPU (a) and GPU (b). Therefore, it represents the weak scalability of the application.

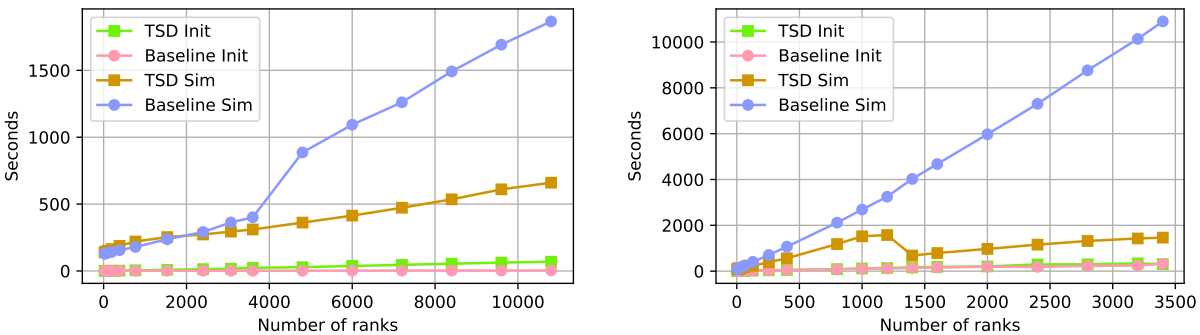
For the CPU experiments (Fig. 8a), the initialization time is very similar in both cases, with TSD showing slightly higher values than Baseline. As explained in Section III-A, this overhead is due to the need for an initial communication step to exchange the connection targets. However, this cost has little impact on the overall performance since the simulation time largely dominates over the initialization time. During the simulation phase, both methods perform similarly at small scale, their behavior diverges significantly as the number of ranks increases. Beyond approximately 3000 ranks, the Baseline approach exhibits a steep, non-linear increase in

Tabla I: JUWELS and JUWELS Booster setup

Feature	Cluster	Booster
Compute nodes	2271	936
CPU	2 × Intel Xeon Platinum 8168	2 × AMD EPYC Rome 7402
CPU cores / node	48	48
GPU	–	4 × NVIDIA A100 (40 GB)
Memory / node	96 GB DDR4	512 GB DDR4
Interconnect	InfiniBand EDR	InfiniBand HDR
Peak performance	∼12 PFLOP/s	∼73 PFLOP/s

Tabla II: Busyring parameters experiments

	Parameter	Value
k_{in}	Number of incoming synapses per neuron	1000
d_{min}	Minimal delay in the global network	5 ms
r_{size}	Ring size	4 cells
w_{syn}	Synaptic weight	0.2
dt	Maximum simulation time step	0.1 ms
dp	Depth of the compartmental hierarchy	2
G_{cpu}	CPU cell group size	1024 cells
G_{gpu}	GPU cell group size	12000 cells



(a) Weak scalability CPU.

(b) Weak scalability GPU.

Fig. 8: Weak scalability results for CPU and GPU.

Tabla III: Simulation parameters for hierarchical neighborhood experiments

	Parameter	Value
k_{in}	Number of incoming synapses per neuron	1024
l_{domain}	Simulation domain length	5 cm
σ	Probability parameter determining synapse length	9000
v_{synapse}	Spike transmission speed	1 m/s
l_{max}	Maximum level for neighborhood communication	2
w_{synapse}	Synaptic weight	0.15
f_{poisson}	Rate of the Poisson spike train	400
w_{poisson}	Weight of the Poisson input	40

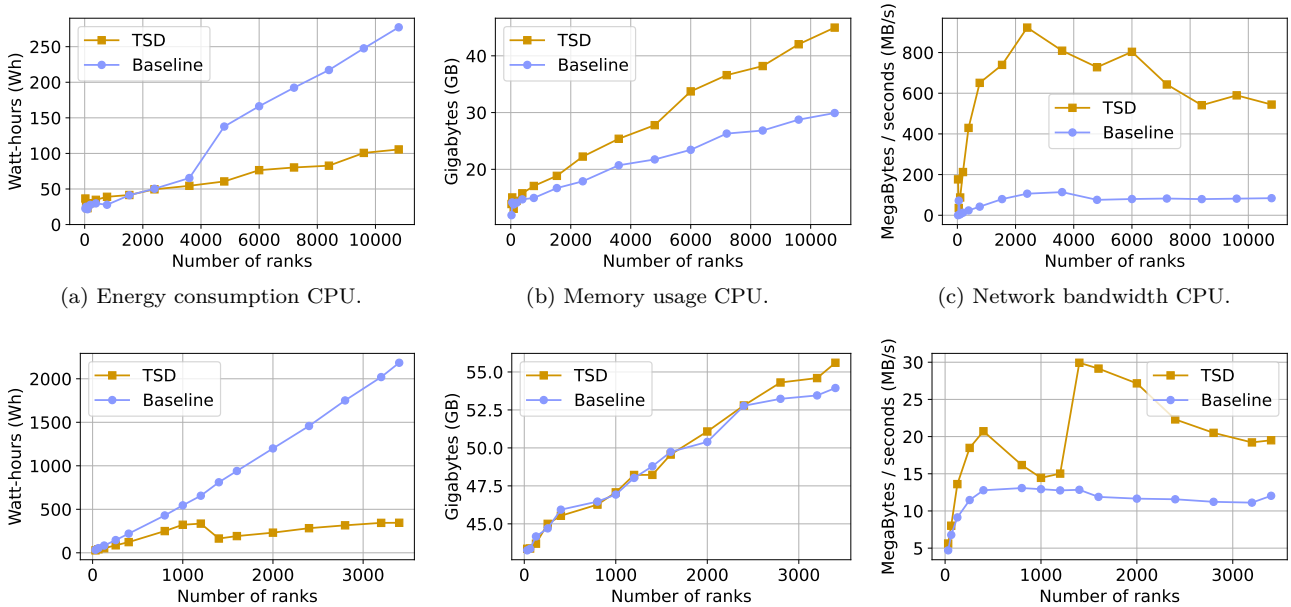
runtime, reaching more than 1700 seconds at 10000 ranks. In contrast, TSD maintains a nearly linear scaling trend, with execution times remaining below 700 seconds at the largest system size.

In the GPU experiments (Fig. 8b), the initialization times are very similar, even though TSD includes an additional communication phase. This overhead is almost negligible because, in this case, each GPU handles 12000 cells compared to 1024 in the CPU experiments, making initialization time more strongly influenced by computational load than by communication cost. The simulation phase of TSD clearly outperforms the BASELINE pattern, which shows a rapid and almost linear increase in time as the number of ranks grows. This leads to a total time that remains much lower for TSD, indicating that it is the more scalable approach in the GPU configuration. An additional effect can be observed in the experiments with 1200 and 1400 ranks, where a reduction in execution time occurs, and then time remains very stable for larger scales. This behavior is explained by the fixed number of 1000 connections per cell: once the number of ranks exceeds the number of connections, the probability of each cell being connected to all ranks decreases, thereby reducing spike communication and leading to savings in simulation time.

Overall, the results highlight that the TSD communication Pattern improves BASELINE, in both cases, achieving up to a 2.5× and 6× speedup, respectively, in the largest configurations, thus demonstrating excellent scalability, enabling the use of larger HPC systems and allowing much larger portions of the brain to be simulated in reasonable time.

After analyzing execution time, we examined energy consumption, which shows a closely correlated behavior. In the CPU experiments (Fig. 9a), BASELINE exhibits a steep, almost linear increase with scale, surpassing 250 Wh, whereas TSD maintains a much lower profile, staying below 150 Wh. In the GPU experiments (Fig. 9d), the contrast is similar: BASELINE grows steadily with the number of ranks, while TSD remains significantly lower across all scales, never exceeding 500Wh even in the largest configuration. The divergence between both strategies becomes more evident at large scale, highlighting the superior efficiency of TSD. Overall, these results confirm that TSD not only reduces runtime but also achieves substantial energy savings, thereby lowering the carbon footprint of the simulations and making it the more effective and sustainable communication strategy for large-scale simulations.

Another fundamental factor to consider is the memory usage per node, which directly limits the scalability of large simulations. Figures 9b and 9e present the evolution of average memory consumption for the CPU and GPU experiments, respectively, highlighting how resource requirements grow with the num-



(a) Energy consumption CPU. (b) Memory usage CPU. (c) Network bandwidth CPU.

(d) Energy consumption GPU. (e) Memory usage GPU. (f) Network bandwidth GPU.

Fig. 9: Energy consumption, memory usage, and bandwidth usage in CPU and GPU.

ber of ranks.

In the CPU experiments, TSD uses more memory than Baseline, and the gap widens with scale: Baseline reaches about 30GB in the largest configuration, whereas TSD exceeds 40GB. However, this remains well below the 96GB available per node. The additional memory comes mainly from the spike vector required by TSD, representing a modest trade-off for its gains in performance and scalability.

In the GPU experiments, both strategies show similar memory usage, remaining between 43GB and 56GB across all scales. Their curves nearly overlap, indicating that the memory overhead of TSD is negligible on GPUs. This is mainly due to the smaller number of ranks per node in the GPU setup. Overall, TSD increases memory usage on CPUs without becoming a practical limitation, while on GPUs it remains comparable to Baseline.

Next, we analyze the communication bandwidth achieved by the two strategies. Figures 9c and 9f report the average transmission rate (in MB/s) for the CPU and GPU experiments, respectively. In CPU experiments (Fig. 9c), TSD achieves significantly higher bandwidth than Baseline across all scales, peaking at nearly 900 MB/s around 2500 ranks. Although the throughput of TSD decreases slightly at larger scales, it remains well above that of Baseline, which stabilizes around 100 MB/s.

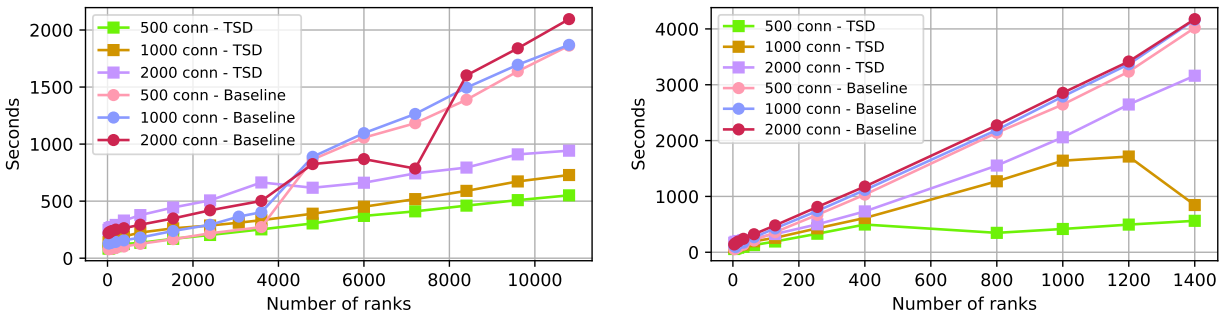
In the GPU experiments (Fig. 9f), the bandwidth of TSD fluctuates with the number of ranks. It first increases up to about 400 ranks as communication grows while exchanges remain efficient. Beyond that point, the fixed limit of 1000 connections per cell reduces the probability that each cell communicates with all ranks, lowering total traffic. However, execution time does not decrease proportionally because some ranks still communicate with nearly all others and dominate synchronization costs. Around 1400 ranks, this effect disappears, reducing execution time and stabilizing bandwidth.

Overall, these results confirm that TSD uses bandwidth more effectively than Baseline and delivers more stable performance across scales. In both CPU and GPU simulations, it reduces execution time and energy consumption with only a small memory overhead, making it the preferable strategy for large-scale simulations.

Network connectivity is a key factor in scalability, since more synapses per cell increase communication costs. Figure 10 compares TSD and Baseline on CPU (10a) and GPU (10b) for 500, 1000, and 2000 connections per cell. In all cases, Baseline grows more rapidly, while TSD scales better.

In the specific case of GPUs, a reduction in communication time is also observed once certain rank thresholds are exceeded. For 1000 connections, this reduction appears around 1000 ranks, while for 500 connections it emerges closer to 500 ranks. This behavior is consistent with the explanation that, as the number of ranks increases, the probability of a cell connecting to all of them decreases, thereby reducing communication cost. However, for 2000 connections, this phenomenon is not yet visible, as the required number of ranks has not been reached.

With the current JUWELS system, simulations exceeding 2000 connections display behavior similar to that shown by the 2000-connection curves. Nevertheless, with the upcoming JUPITER supercomputer, it will be possible to extend these analyses to much larger configurations, reaching the order of 10000 ranks on GPU. This will make it feasible to observe the same reduction effects already identified for 500 and 1000 connections, but now at a scale of several thousand additional connections, thereby validating the scalability trends presented in this study. Importantly, with this improvement, simulations with



(a) Connection scalability CPU.

(b) Connection scalability GPU.

Fig. 10: Connection scalability for CPU and GPU.

10000 connections per cell bring us significantly closer to the long-term goal of modeling complex brain structures. While the number of synapses in the brain can vary from a few to hundreds of thousands of synapses per neuron, the average number of synapses in the cortex is around 10,000 [17].

B. Hierarchical neighborhood communication

After discussing the performance improvements of TSD, we now present results for HNC in Euclidean space in Fig. 11. For this, we compare the baseline with TSD, HNC, and HNC & TSD. In the last case, TSD is used for the global communication step required for long-range connections.

The initialization times are similarly for all approaches (Fig. 11a). For a small number of ranks (≤ 64 ranks) Baseline and TSD outperform HNC regarding simulation time (Fig. 11b) as the additional overhead of the neighborhood exceeds its benefits, especially since smaller simulations can be run on a single node, thereby obviating the need for network communication.

For the largest simulation with 4,096 ranks, TSD achieves a speedup of 2.06 compared to Baseline pattern, HNC & TSD by 3.66, and HNC by 3.74.

The total time (Fig. 11c) shows the same trend as the simulation time but with smaller total speedups. For longer simulations, the speedup of the total time would increase as the initialization time remains unchanged. HNC's advantage lies in its local high-frequency communication and less frequent communication at higher levels. Consequently, both HNC approaches outperform Baseline and TSD. TSD outperforms Baseline as its advantage lies in sending spikes only to destinations rather than sharing them globally. We anticipate TSD's advantage to grow significantly with a higher number of ranks, as observed in Sec. V-A.

The consumed energy (Fig. 11d) follows the trend of the total time as expected. The memory usage (Fig. 11e) is similar for all four approaches. Fig. 11f shows the total number of bytes sent from all nodes.

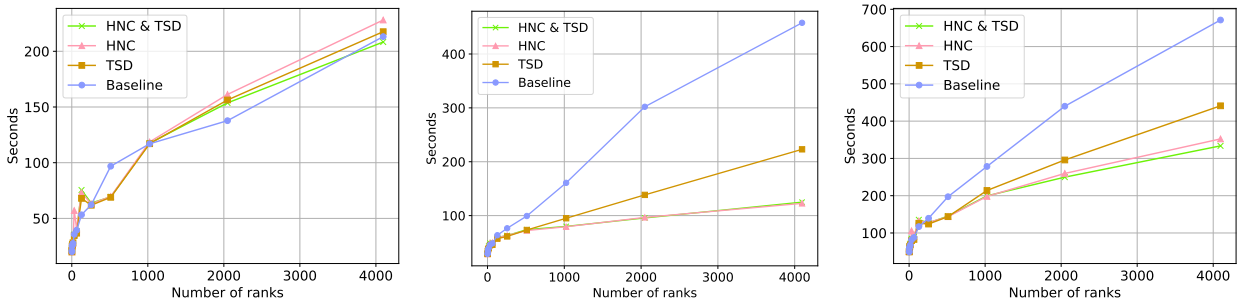
At first glance, it seems counterintuitive that TSD sends fewer data than HNC, as HNC outperforms TSD in terms of simulation time. The reason is that TSD delivers spikes directly to targets, whereas HNC forwards them to all neighbors. This leads to higher data volume, but the advantage of localized communication outweighs the overhead.

The benefits of our communication strategy increase with the size of the communication component in the entire simulation. Higher firing rates increase the benefit of our communication strategy, as it can be influenced by the background activity, synapse weights, and the number of synapses. Another important factor is the distribution of synapses. HNC works best when the network consists of many local connections and only a few locations, as it can also be found in the brain. More distant connections require global communication, which decreases the performance. In the worst case, the network would have all-to-all connectivity, where both HNC and TSD just increase the overhead but still need to broadcast all spikes in the network. This is comparable with the experiments with a small number of ranks, as the number of synapses per neuron remains the same, while the number of neurons is comparably small, resulting in comparably high connectivity. In these cases, Baseline outperforms TSD and HNC.

To sum up, our approach scales very well with increasing the number of ranks for larger simulations. We expect the benefit of our approach to increase with larger networks, as global communication will perform even worse, and local communication should remain almost constant.

VI. Conclusion

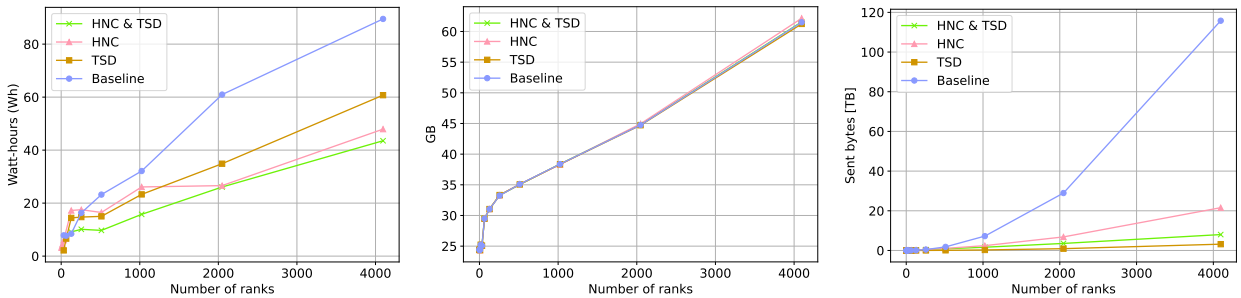
We demonstrated how TSD and HNC communication can significantly reduce communication time in large-scale simulations. Notably, we achieved a speed up of 2.5 on CPU and 6 on GPUs with TSD alone with 10000 and 3500 MPI ranks. When we introduced an Euclidean space and neighborhood communication, we achieved a speed up of 3.7 already with 4096 ranks. This approach is biologically inspired, leveraging spatial locality and plasticity similar to the way spikes in the brain typically travel locally rather than globally. We believe that the performance of both TSD alone and combined with HNC will further increase with larger networks and lay the foundation for large-scale brain simulations. While simulating with large numbers of nodes was mostly unfeasible, we now lay the foundation for these simulations. These advances open the door to mode-



(a) Initialization time. All four approaches perform similarly during initialization. As the number of ranks increases, so does the initialization time. The main cost is network generation, where neuron connectivity requires evaluating many probability pairs, even with the approximation.

(b) Simulation time. For smaller networks up to 64 ranks, Baseline and TSD outperform our HNC approach. At 64 ranks, TSD begins to demonstrate advantages over Baseline. HNC mirrors TSD until 1,024 ranks, where it surpasses TSD.

(c) Total time. Sum of initialization and simulation time. Including initialization, total runtimes yield smaller speedups of 1.52 (TSD), 1.9 (HNC), and 2.01 (HNC & TSD), as initialization dominates and remains constant across methods.



(d) Energy usage. The energy usage generally follows the trend of the total time. In the largest case with 4096 ranks Baseline consumes 89.5Wh, TSD 60.7Wh, HNC 43.5Wh, and HNC & TSD 47.5Wh. Hence, we save 32 % (TSD), 51.3 % (HNC), and 46 % (HNC & TSD) of energy.

(e) Memory usage. In our simulations, the memory usage increases with larger network sizes as expected. All four approaches have a similar memory usage.

(f) Total sent bytes. For all four approaches, the number of bytes increases with the number of ranks because more spikes need to be transmitted in larger networks. Baseline scales worst in the largest simulation (117 TB), followed by HNC (21 TB), and HNC+TSD (8 TB), and TSD (3 TB).

Fig. 11: Timings, memory, energy usage, and transmitted bytes in the Euclidean setup comparing HNC, TSD, and Baseline.

ling scenarios that were previously out of reach and lay the foundation for more biologically realistic and computationally scalable neuroscience.

Acknowledgements

The development of Arbor supported by EBRAINS2.0, which has received funding from the European Union's Research and Innovation Program Horizon Europe under Grant Agreement No. 101147319. Sandra Diaz has also received funding from the European Union's Horizon 2020 Framework Program for Research and Innovation under the Grant Agreement No. 101137289 (Virtual Brain Twin). This research was also supported by the Helmholtz Joint Lab "Supercomputing and Modeling for the Human Brain". The authors would like to thank the NHR-Verein e.V. for supporting this work within the NHR Graduate School of National High Performance Computing (NHR). The authors gratefully acknowledge the Federal Ministry of Research, Technology and Space (BMFTR) and the Hessian Ministry of Science and Research, Art and Culture (HMWK) for supporting this work as part of the NHR funding. The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS Supercomputer JUWELS at Jülich Supercomputing Centre (JSC). This work is also part of Project PID2022-136454NB-C21, funded by MCIN/AEI/10.13039/501100011033 and by FEDER/EU, and supported by the Spanish Ministry of Education through the FPU21/03110 grant.

Referencias

[1] Paula Sanz Leon, Stuart A Knock, M Marmaduke Woodman, Lia Domide, Jochen Mersmann, Anthony R McIntosh, and Viktor Jirsa, "The virtual brain: a simulator of primate brain network dynamics," *Frontiers in Neuroinformatics*, vol. 7, pp. 10, 2013.

[2] Marc-Oliver Gewaltig and Markus Diesmann, "Nest (neural simulation tool)," *Scholarpedia*, vol. 2, no. 4, pp. 1430, 2007.

[3] Gail S. Hines, "Advances in simulation software technology," *SIMULATION*, vol. 42, no. 1, pp. 36–36, 1984.

[4] N. Abi Akar, B. Cumming, V. Karakasis, A. Küsters, W. Klijn, A. Peyser, and S. Yates, "Arbor — A Morphologically-Detailed Neural Network Simulation Library for Contemporary High-Performance Computing Architectures," in *2019 27th Euromicro International Con-*

ference on Parallel, Distributed and Network-Based Processing (PDP), feb 2019, pp. 274–282.

[5] Sarah M Keating, Dagmar Waltemath, Matthias König, Fengkai Zhang, Andreas Dräger, Claudine Chaouiya, Frank T Bergmann, Andrew Finney, Colin S Gillespie, Tomáš Helikar, et al., "Sbml level 3: an extensible format for the exchange and reuse of biological models," *Molecular Systems Biology*, vol. 16, no. 8, pp. e9110, 2020.

[6] Lionel Kusch, Sandra Diaz-Pier, Wouter Klijn, Kim Sontheimer, Christophe Bernard, Abigail Morrison, and Viktor Jirsa, "Multiscale co-simulation design pattern for neuroscience applications," *Frontiers in Neuroinformatics*, vol. 18, pp. 1156683, 2024.

[7] Thorsten Hater, Juliette Courson, Han Lu, Sandra Diaz-Pier, and Thanos Manos, "Arbor-tvb: A novel multi-scale co-simulation framework with a case study on neural-level seizure generation and whole-brain propagation," *arXiv preprint arXiv:2505.16861*, 2025.

[8] Robert Gütig, "Spiking neurons can discover predictive features by aggregate-label learning," *Science*, vol. 351, no. 6277, pp. aab4113, 2016.

[9] Júlia V Gallinaro, Nebojša Gašparović, and Stefan Rotter, "Homeostatic control of synaptic rewiring in recurrent networks induces the formation of stable memory engrams," *PLOS Computational Biology*, vol. 18, no. 2, pp. e1009836, 2022.

[10] Markus Butz and Arjen Van Ooyen, "A simple rule for dendritic spine and axonal bouton formation can account for cortical reorganization after focal retinal lesions," *PLoS computational biology*, vol. 9, no. 10, pp. e1003259, 2013.

[11] Han Lu, Sandra Diaz-Pier, Maximilian Lenz, and Andreas Vlachos, "The interplay between homeostatic synaptic scaling and homeostatic structural plasticity maintains the robust firing rate of neural networks," *eLife*, vol. 12, pp. RP88376, jul 2025.

[12] Viktor K Jirsa, Timothée Proix, Dionysios Perdikis, Michael Marmaduke Woodman, Huifang Wang, Jorge Gonzalez-Martinez, Christophe Bernard, Christian Bénar, Maxime Guye, Patrick Chauvel, et al., "The virtual epileptic patient: individualized whole-brain models of epilepsy spread," *Neuroimage*, vol. 145, pp. 377–388, 2017.

[13] Anna Maxion, Ekaterina Kutafina, Maike F Dohrn, Pierre Sacré, Angelika Lampert, Jenny Tigerholm, and Barbara Namer, "A modelling study to dissect the potential role of voltage-gated ion channels in activity-dependent conduction velocity changes as identified in small fiber neuropathy patients," *Frontiers in Computational Neuroscience*, vol. 17, pp. 1265958, 2023.

[14] Claudia Bachmann, Tom Tetzlaff, Renato Duarte, and Abigail Morrison, "Firing rate homeostasis counteracts changes in stability of recurrent neural networks caused by synapse loss in alzheimer's disease," *PLoS Computational Biology*, vol. 16, no. 8, pp. e1007790, 2020.

[15] Frederico AC Azevedo, Ludmila RB Carvalho, Lea T Grinberg, José Marcelo Farfel, Renata EL Ferretti, Renata EP Leite, Wilson Jacob Filho, Roberto Lent, and Suzana Herculano-Houzel, "Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain," *Journal of Comparative Neurology*, vol. 513, no. 5, pp. 532–541, 2009.

[16] Suzana Herculano-Houzel, "The human brain in numbers: a linearly scaled-up primate brain," *Frontiers in human neuroscience*, vol. 3, pp. 857, 2009.

[17] John C Silbereis, Sirisha Pochareddy, Ying Zhu, Mingfeng Li, and Nenad Sestan, "The cellular and molecular landscapes of the developing human central nervous system," *Neuron*, vol. 89, no. 2, pp. 248–268, 2016.

[18] José Luis Bosque and L. P. Perez, "Theoretical scalability analysis for heterogeneous clusters," in *4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2004), April 19-22, 2004, Chicago, Illinois, USA*. 2004, pp. 285–292, IEEE Computer Society.

[19] José Luis Bosque, Oscar David Robles, Pablo Toharia, and Luis Pastor, "Evaluating scalability in heterogeneous systems," *J. Supercomput.*, vol. 58, no. 3, pp. 367–375, 2011.

[20] "The human brain project: A european future and emerging technologies (fet) flagship project," `https://www.humanbrainproject.eu/`, 2023, 2013–2023.

[21] Sebastian Rinke, Markus Butz-Ostendorf, Marc-André Hermanns, Mikaël Naveau, and Felix Wolf, "A scalable algorithm for simulating the structural plasticity of the brain," *Journal of Parallel and Distributed Computing*, vol. 120, pp. 251–266, 2018.

[22] Fabian Czappa, Alexander Geiß, and Felix Wolf, "Simulating structural plasticity of the brain more scalable than expected," *Journal of Parallel and Distributed Computing*, vol. 171, pp. 24–27, 2023.

[23] Jülich Supercomputing Centre, "Juwels: Modular tier-0/1 supercomputer at the jülich supercomputing centre," *Journal of Large-Scale Research Facilities*, vol. 5, pp. A135, 2019.

[24] Rebecca D Hodge, Trygve E Bakken, Jeremy A Miller, Kimberly A Smith, Eliza R Barkan, Lucas T Graybuck, Jennie L Close, Brian Long, Nelson Johansen, Osnat Penn, et al., "Conserved cell types with divergent features in human versus mouse cortex," *Nature*, vol. 573, no. 7772, pp. 61–68, 2019.

[25] Almut Schüz and Günther Palm, "Density of neurons and synapses in the cerebral cortex of

the mouse," *Journal of comparative neurology*, vol. 286, no. 4, pp. 442–455, 1989.

Interferencia en redes Dragonfly: análisis de patrones de tráfico y políticas de mapeo

Alejandro Baviera[1], María Engracia Gómez[1] y Julio Sahuquillo[1]

Resumen— **La red de interconexión tiene un papel clave en las prestaciones obtenidas en los supercomputadores. La selección de la topología resulta fundamental para satisfacer los requisitos de prestaciones y consumo de estos sistemas. Habitualmente se emplean topologías avanzadas con conmutadores de alto grado, como la Dragonfly. Sin embargo, esta arquitectura no está libre de problemas de contención en la red, tanto entre aplicaciones que comparten el sistema como entre los mensajes de una misma aplicación. Cuando múltiples trabajos se ejecutan de forma concurrente, compiten por los recursos, lo que provoca interferencias en sus comunicaciones a través de la red.**

La interferencia que experimentan las aplicaciones paralelas depende en gran medida del patrón de tráfico que presenta ella y del patrón del resto de aplicaciones ejecutándose concurrente, y de las políticas de asignación de nodos de procesamiento a las aplicaciones que pueden cambiar estos patrones. Por este motivo, en este trabajo analizamos, sobre la topología Dragonfly, la relación existente entre los patrones de tráfico y las políticas de mapeo con las prestaciones obtenidas en la interferencia sufrida por las aplicaciones.

Palabras clave— **Interferencia, Políticas de mapeo, Redes de interconexión, Topología Dragonfly.**

I. Introducción

LOS superordenadores actuales integran un gran número de elementos de procesamiento para llegar a alcanzar altas prestaciones. Estos elementos se comunican por medio de una infraestructura de interconexión, que permite abordar problemas de gran complejidad vinculados a desafíos clave de la sociedad, como la salud, la medicina o la simulación de catástrofes. En este contexto, la red que interconecta los nodos de cómputo se convierte en un componente crítico, ya que contribuye de forma decisiva a las prestaciones de estos sistemas masivamente paralelos [1]. En estos sistemas la red de interconexión debe escalar hasta cientos de miles de nodos.

Las topologías tradicionales no satisfacen los requisitos de coste y prestaciones de los sistemas HPC, por ello se utilizan topologías más recientes con conmutadores de alto grado, como la Dragonfly [2] y otras variantes de la misma [3], [4], [5]. No obstante, estas topologías también introduce nuevos retos que hay que abordar. El uso de conmutadores de alto grado y redes de bajo diámetro dificulta el aislamiento de las aplicaciones que se ejecutan concurrentemente en el sistema, y por lo tanto imponen la compartición de los recursos de red entre ellas. Esta compartición ha sido identificada como una de las principales causas de la variabilidad en las prestaciones de aplicaciones HPC [6], debido a la contención en el uso de los recursos que se generan durante las comunicaciones.[7], [8], [9]. En el caso de las topologías estilo Dragonfly, esta interferencia se magnifica aún más por el hecho de que funcionan mejor bajo tráfico aleatorio uniforme [10], lo que pone de manifiesto la importancia de tener en cuenta la política de mapeo de aplicaciones a nodos de procesamiento y el patrón de tráfico de las mismas para reducir la interferencia. El nivel de degradación de prestaciones debido a la contención entre aplicaciones en la red depende de múltiples factores pero entre ellos la topología, el patrón de tráfico y el mapeo de aplicaciones en la red son dos de las más importantes [9].

Este artículo analiza en la topología Dragonfly la interferencia dependiendo del tráfico que generan las aplicaciones y del mapeo de las aplicaciones a nodos de procesamiento. En concreto hemos analizado cinco políticas de mapeo y seis patrones de tráfico analizando la degradación de prestaciones con distintas intensidades de tráfico background.

II. Topologia Dragonfly

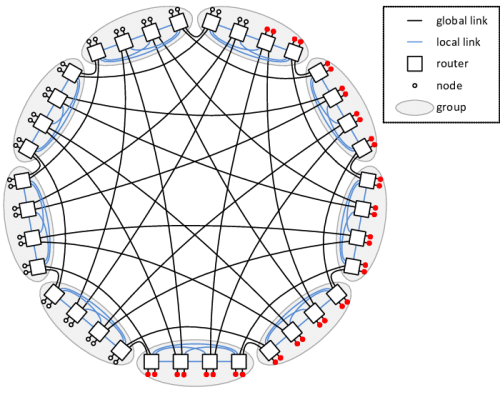
La topología Dragonfly es una red jerárquica definida por 3 parámetros (a, p y h). La red se organiza en grupos, cada uno con el mismo número de switches y cada switch tiene un conjunto de nodos de procesamiento conectados. Los switches pertenecientes al mismo grupo están totalmente interconectados. Cada switch tiene 3 tipos de enlaces; *terminales* que conectan con nodos de procesamiento (parámetro p), *locales* que conectan con el resto de switches del mismo grupo (dado que parámetro a es el número de switches por grupo, cada switch tiene $a-1$ enlaces locales); y *globales* que conectan con h switches cada uno en otro grupo distinto (parámetro h). Por tanto, el grado de cada switch es k=p+a+h-1. Cada grupo tiene conectados $a \times p$ nodos de cómputo y $a \times h$ conexiones globales. El alto grado de los switches permite un diámetro global muy bajo.

Los parámetros a, p y h de la Dragonfly pueden tener cualquier valor. Sin embargo, para equilibrar la carga de los canales en el tráfico con carga uniforme, la red debe tener a=2p=2h. Dado que cada paquete atraviesa dos canales locales a lo largo de su ruta por un canal global y un canal terminal, esta proporción mantiene el equilibrio.

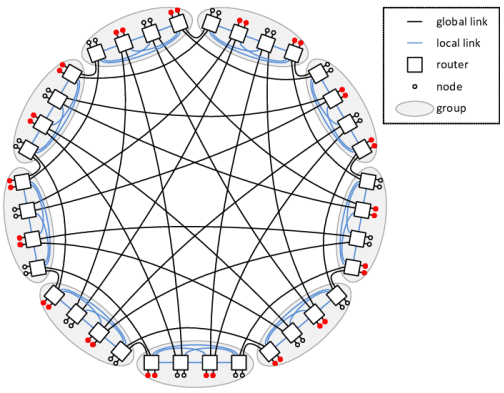
III. Trabajo Relacionado

Las causas de la variabilidad en las prestaciones de aplicaciones paralelas se analizan en [6]. Los autores identifican la contención por recursos entre aplicaciones como el principal contribuyente a esta variabilidad, y al margen de la CPU y la memoria, la

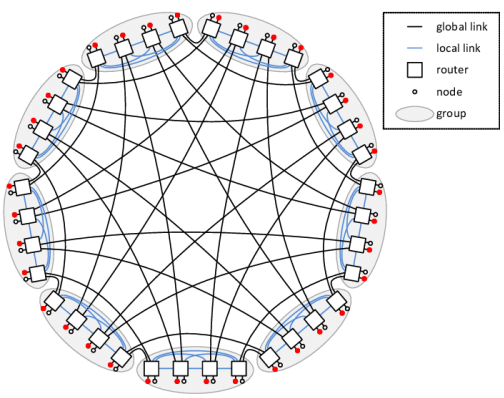
[1]Dpto. de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mail: abavgua@etsinf.upv.es, {megomez,jsahuqui}@disca.upv.es



(a) Mapeo lineal



(b) Mapeo alternativo switch



(c) Mapeo alternativo nodo

Fig. 1: Políticas de mapeo consideradas.

contención en las comunicaciones de las aplicaciones paralelas a través de operaciones MPI es uno de los elementos a considerar. Las causas de variabilidad en las prestaciones son también analizadas en [11]. Los autores observan que en aplicaciones paralelas con mucha comunicación las prestaciones pueden variar entre el 28 % más rápido al 41 % más lento con respecto a las prestaciones medias. Además concluyen que la principal razón de esta alta variabilidad es la contención en los recursos de red compartidos con otras aplicaciones.

El trabajo en [9] evalúa el impacto de la contención entre aplicaciones para cargas de trabajo HPC, pero la topología seleccionada es *fat-tree*. El trabajo estima la cantidad de pérdida de prestaciones del sistema que la contención entre aplicaciones contribuye en los sistemas HPC actuales, estimada en torno al 10 %.

IV. Políticas de mapeo y patrones de tráifco

A. Políticas de mapeo

Esta sección describe las políticas de mapeo que se han utillizado para analizar la interferencia dependiendo de la ubicación de las tareas de las aplicaciones en la red. Hemos considerado cinco políticas diferentes de mapeo de tareas de aplicaciones paralelas a nodos de procesamiento de la red:

- *Lineal.* La primera de ellas consiste en asignar nodos de cómputo sucesivos en la red a tareas consecutivas de la aplicación (ver Figura 1a) hasta completar las tareas de la misma.
- *Switches alternativos.* En la Figura 1b se muestra la segunda política donde se van alternando switches completos, de forma que se asignan todos los nodos de procesamiento de switches alternos.
- *Nodos alternativos.* La tercera política asigna nodos alternos. (ver Figura 1c)
- *Random.* Esta política asigna los nodos de procesamiento de forma aleatoria.
- *Ratio x:y* La última política de mapeo distribuye las aplicaciones en los nodos siguiendo una proporción predefinida dentro de cada uno de los grupos de la topología (x + y = a). En un grupo, se asignan x switches con la aplicación bajo observación e y switches de tráfico sintético de background, mientras que en el siguiente grupo, la asignación se invierte.

Estas políticas pretenden evaluar situaciones dispares de mapeo de aplicaciones para analizar la interferencia producida en cada una de estas situaciones.

B. Patrones de tráfico

Los patrones que hemos evaluado son representativos de comportamientos presentes en aplicaciones paralelas. Son los siguientes:

- Neighbor: Cada nodo envía mensajes a un nodo aleatorio de su mismo grupo.
- Hotspot: Cada nodo, en cada envío, decide entre: enviar al destino hotspot configurado o enviar a un destino aleatorio.
- Stencil2D: Los nodos se organizan como una malla 2D. Cada nodo envía a uno de sus vecinos inmediatos válidos: izquierda, derecha, arriba o abajo.
- Stencil4D: Los nodos se organizan en una malla 4D. Cada nodo envía a uno de sus vecinos inmediatos válidos en cada dimensión.
- Many-to-Many: Cada nodo envía un mensaje a cada uno de los demás nodos de su grupo.
- Adversarial1: Cada nodo envía a un nodo aleatorio dentro del grupo destino calculado como: grupo_origen + 1 (con módulo del número total de grupos).
- AdversarialH: Cada nodo envía a un nodo aleatorio dentro del grupo destino calculado como: grupo_origen + h (con módulo del número total

de grupos).

Para analizar el efecto de la interferencia sobre estos patrones, se han estudiado los siguientes escenarios de interferencia:

1. Escenario sin interferencia: El patrón se ejecuta en la mitad de los nodos de la red y estos nodos se mapean siguiendo alguna de las políticas de mapeo descritas en la sección VI-A para mapear las tareas del patrón.
2. Escenario con interferencia: El patrón seleccionado se ejecuta también en la mitad de los nodos de la red pero en este caso con interferencia proviniente de otro patrón ejecutado en la otra mitad de la red. Esta interferencia es generada por tráfico uniforme o tráfico hot-spot, analizando la degradación de prestaciones sufrida. El tráfico uniforme envía mensajes con igual probabilidad a todos los destinos pertenecientes a este patrón. Estos dos patrones de tráfico pretende reproducir la interferencia producida por diferentes aplicaciones paralelas con distintos niveles de uso de la red. En este escenario podemos graduar el nivel de interferencia para analizar el comportamiento de cada patrón cuando sufre distintos niveles de interferencia.

V. Entorno Experimental: Simulador y Topología

La evaluación experimental se ha realizado por medio de simulación con el simulador y la topología descrita a continuación.

A. Simulador

Las políticas de mapeo y escenarios se han modelado en Sauron [12], un simulador basado en OMNeT++ que modela el comportamiento de toda la red, para su evaluación.

Para un mejor análisis de la utilización de los enlaces se ha dotado al simulador con la capacidad de calcular la utilización de salida de los puertos por aplicación. Esto permite medir de una forma precisa como afecta el tráfico de las aplicaciones en los puertos de la red.

Para reproducir el comportamiento de aplicaciones paralelas utilizamos trazas de aplicaciones HPC descritas en la siguiente sección.

B. Topología analizada

Para evaluar las prestaciones proporcionadas por la red, se ha seleccionado la configuración equilibrada de la topología Dragonfly cuyas característica pueden verse en la Tabla I. Esta configuración con 1056 nodos de cómputo permite analizar cómo la distribución de los recursos influye en la congestión de la red y en el prestaciones generales del sistema.

El primer patrón se ejecuta en 512 nodos de procesamiento y el patrón de background en el resto de nodos de la red.

VI. Resultados experimentales

En esta sección evaluamos la interferencia sufrida por los distintos patrones de tráfico presentados en IV-B dependiendo de las políticas de mapeo presentadas en VI-A. En las distintas subseciones podremos ver que la mejor política depende del patrón de tráfico de la aplicación y del patrón background de las aplicaciones que se están ejecutando de forma concurrente.

A. Tráfico adversarial

Este patrón de tráfico es especialmente dañino para la topología Dragonfly. Este patrón se mapea siguiendo las distintas políticas presentadas en sin interferencia y con interferencia producida por tráfico uniforme (Figura 2)) y tráfico hot-spot (Figura 3).

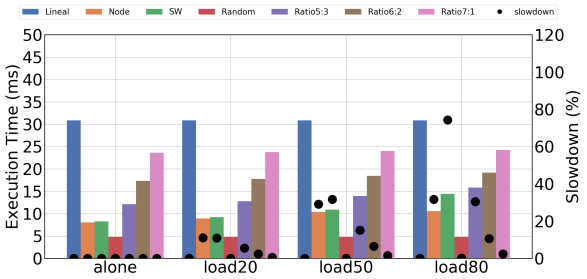
En la Figura 2 podemos ver en el primer grupo de barras el comportamiento de este tráfico con las distintas políticas de mapeo. Como puede verse la política lineal obtiene tiempos de ejecución muy superiores al resto. Esto se debe a que el resto de mapeos modifican el patrón transformándolo en patrones más benignos para la topología. Las políticas ratio son las que más se alejan de la lineal y la que obtiene los mejores tiempos de ejecución es la random porque elimina los cuellos de botella del patrón al distribuir las tareas de la aplicación de manera aleatoria. En la Figura 4 se presenta la utilización para los tres tipos de enlaces presentes en la topología ordenados en orden creciente de utilización. Como puede verse en los mapeos que peor funcionan hay enlaces globales congestionados que provocan árboles de congestión. Mienstras que el mapeo random al distribuir de manera aleatoria los nodos de cómputo permite reducir esta congestión, no llegando ninguno de los enlaces a tener una utilización superior al 60 %, mientras que lineal y ratio7:1 tienen enlaces al máximo de su capacidad.

Al introducir interferencia con tráfico uniforme, ver resto grupos de la Figura 2, todas las políticas excepto la lineal incrementan su tiempo de ejecución, poduciéndose el mayor slowdown en la política switch. Lineal no empeora el tiempo de ejecución con interferencia porque es capaz de aislar ambos patrones de manera que no haya interferencia entre ambos. Independientemente de la intensidad de tráfico uniforme background (eje X de las gráficas) y la intensidad del tráfico adversarial (diferentes subfiguras), la política de mapeo random es la que obtiene los mejores tiempos de ejecución aunque en el caso de la Figura 2c con alto nivel de tráfico adversarial y alto nivel de tráfico background el slowdown que experimenta es el segundo más alto.

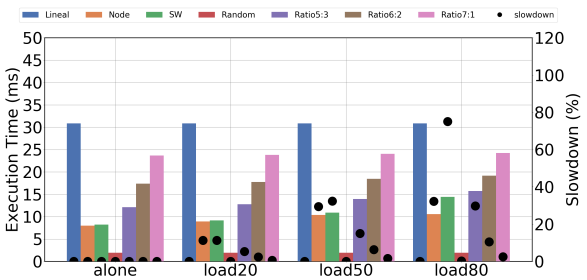
Con tráfico background hotspot, como puede verse en la Figura 3), la política random sufre mayores slowdown, siendo la que tiene más slowdown (400 %) con tráfico background del 80 % en la Figura 3c. Aún así es siempre la política con menor tiempo de ejecución para tráfico adversarial y para ambos tráficos de background. Con tráfico adversarialH las resutlados y conclusiones son similares.

Tabla I: Topologia Dragonfly analizada.

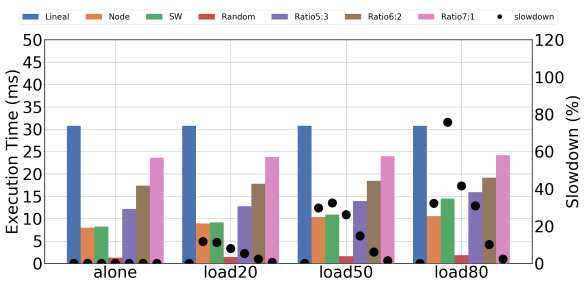
Conf.	# nodos	# nodos por grupo	# switches	# grupos	aridad switch	# enlaces locales	# enlaces globales
a8-h4-p4	1056	32	264	33	15	1848	1056



(a) Adversarial trafico 20 %. Tráfico background uniforme.

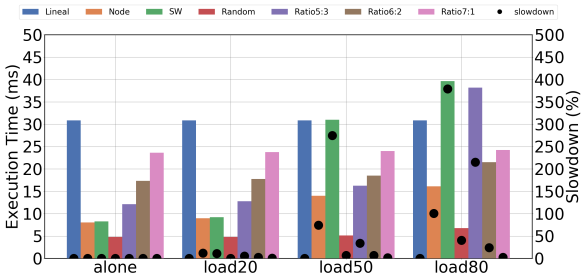


(b) Adversarial trafico 50 %. Tráfico background uniforme.

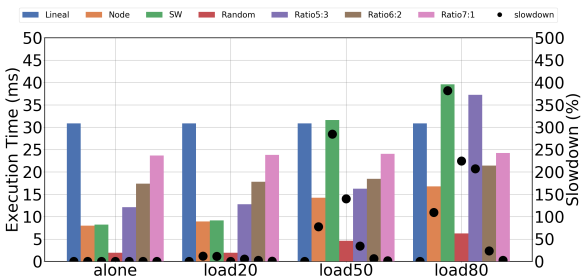


(c) Adversarial trafico 80 %. Tráfico background uniforme.

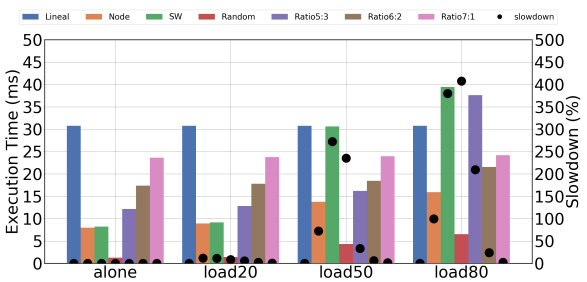
Fig. 2: Tiempos de ejecución (barras) y slowdown (puntos) para tráfico adversarial de diferentes intensidades y con diferentes niveles de tráfico background uniforme (eje X) para las distintas políticas de mapeo.



(a) Adversarial trafico 20 %. Tráfico background hotspot.



(b) Adversarial trafico 50 %. Tráfico background hotspot.



(c) Adversarial trafico 80 %. Tráfico background hotspot.

Fig. 3: Tiempos de ejecución y slowdown para tráfico adversarial de diferentes intensidades y con diferentes niveles de tráfico background hot-spots para las distintas políticas de mapeo.

B. Tráfico neighbor

Este patrón exhibe un comportamiento totalmente local al grupo si no se modifica con la política de mapeo. Observando los tiempos de ejecución en solitario de la Figura 5 se puede ver que los mapeos lineales y random son los que obtienen los menores tiempos de ejecución junto con la política rato7:1. En la Figure 7 puede verse la utilización de los enlaces. Los enlaces globales no se utilizan en el mapeo lineal, mientras que random hace uso de más enlaces globales tensionando menos que el resto de políticas los que están más utilizados.

Al introducir tráfico background uniforme, el mapeo lineal no sufre interferencia y mantiene su tiempo de ejecución, mientras que la política random sufre interferencia cuando el tráfico es elevado. Respecto al resto de políticas, todas ellas sufren más slowdown siendo la política switch la que sufre más degradación

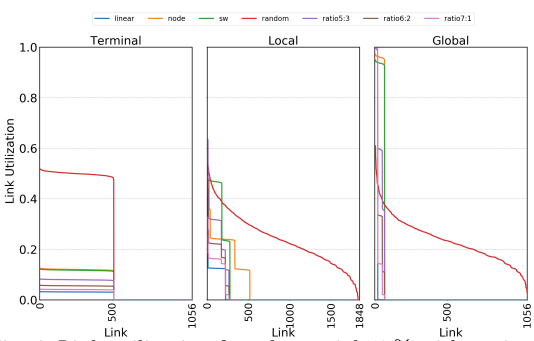
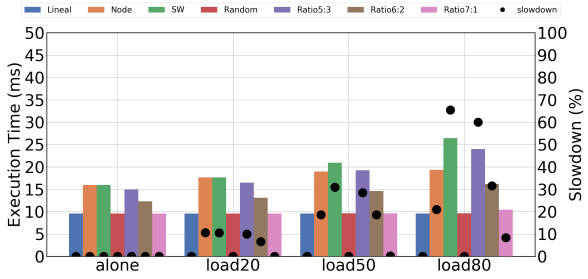


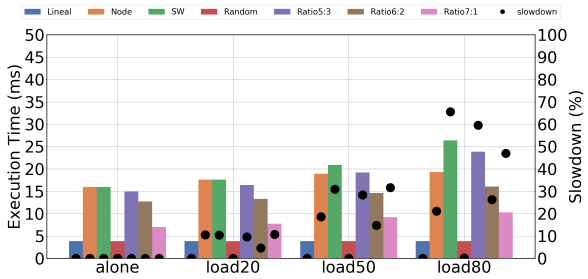
Fig. 4: Link utilization for adversarial 50 % without interference.

de prestaciones.

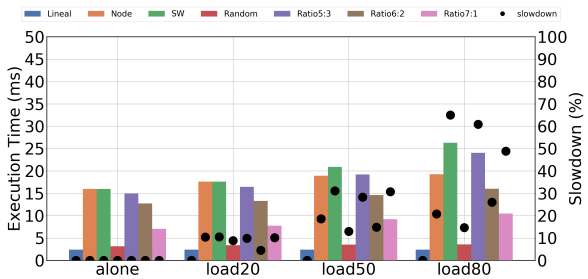
Con tráfico background hotspot (8), la política ramdon sufre más slowdown llegando casi al 400 % y siendo la segunda que más degradación de prestaciones padece con alto tráfico (ver Figura 6c). En este caso las políticas lineal y ratio7:1 son las que consi-



(a) Neighbor tráfico 20 %.

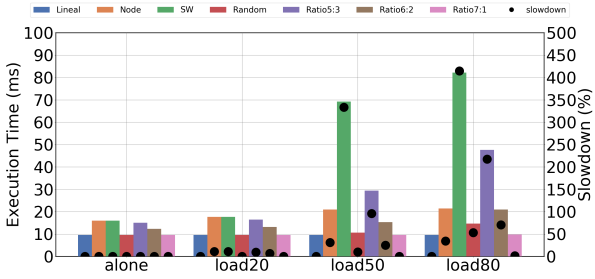


(b) Neighbor tráfico 50 %.

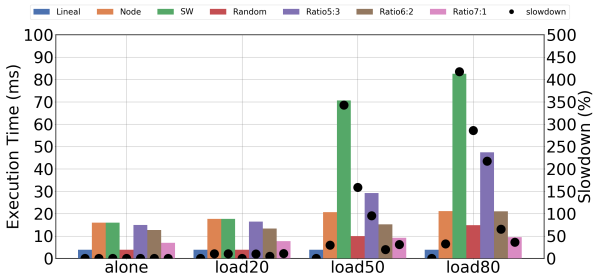


(c) Neighbor tráfico 80 %.

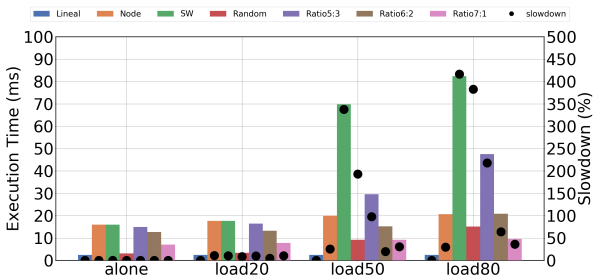
Fig. 5: Tiempos de ejecución y slowdown para tráfico neighbor de diferentes intensidades y con diferentes niveles de tráfico background uniforme para las distintas políticas de mapeo.



(a) Neighbor trafico 20 %.



(b) Neighbor trafico 50 %.



(c) Neighbor trafico 80 %.

Fig. 6: Tiempos de ejecución y slowdown para tráfico neighbor de diferentes intensidades y con diferentes niveles de tráfico background hot-spots para las distintas políticas de mapeo.

guen mantener los tiempos de ejecución más bajos al mantener la mayor parte del tráfico dentro del grupo, siendo el tiempo de ejecución de lineal muy bajo comparado con el del resto.

C. *Tráfico hot-spot*

Para el caso de tráfico hotspot con tráfico background uniforme no se observan diferencias entre los distintos mapeos. Si la interferencia es generada por tráfico hotspot (ver Figura 8), la política que obtiene el peor tiempo de ejecución es la política random, además de ser la que más interferencia sufre, mayor slowdown. Las políticas que sufren menos interferencia son la lineal y las de ratio, al matener el árbol de congestión de cada tráfico local a ese grupo.

D. *Tráfico many-to-many*

Para el caso de tráfico many-to-many podemos ver que lineal es el que obtiene los mejores resultados seguido de random. Con tráfico background uniforme (Figuras 9a y 9b) lineal mantiene su tiempo de ejecución y el resto los incrementa. Si la interferencia es generada por tráfico hotspot (ver Figura 9c y 9d), las conclusiones son similares pero algo maginificadas, siendo random el que obtiene el segundo peor slowdown. Es decir random sufre bastante con este tipo de tráfico background.

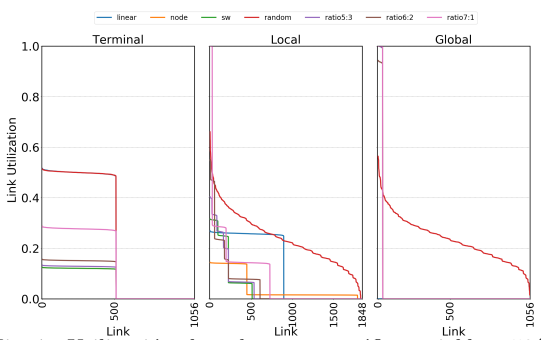
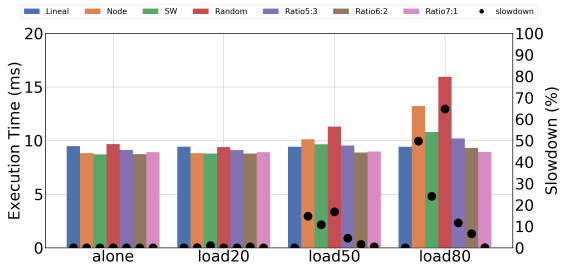


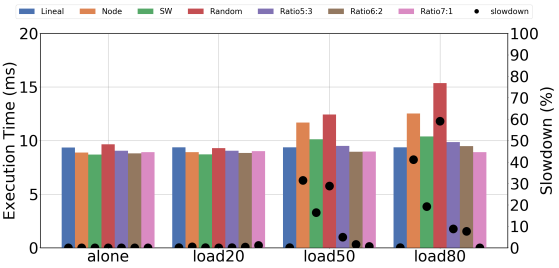
Fig. 7: Utilización de enlace para tráfico neighbor 50 % sin interferencia.

E. *Tráfico stencil2D y stencil4D*

Para ambos tráficos la política random es la que obtiene los tiempos de ejecución menores a solas y con interferencia de tráfico uniforme (ver Figuras 10a y 10c). Sin embargo, cuando el tráfico background es hot-spot y este se incrementa, el mapeo aleatorio, como puede verse en las Figuras 10b y 10d, para am-

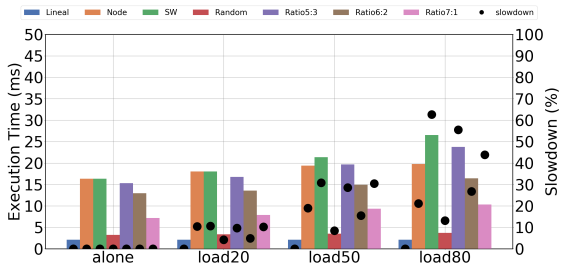


(a) Hotspot tráfico 50 %. Tráfico background hotspot.

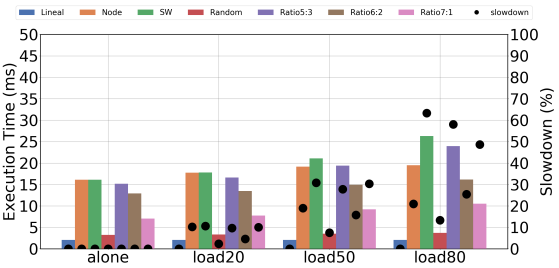


(b) Hotspot tráfico 80 %. Tráfico background hotspot.

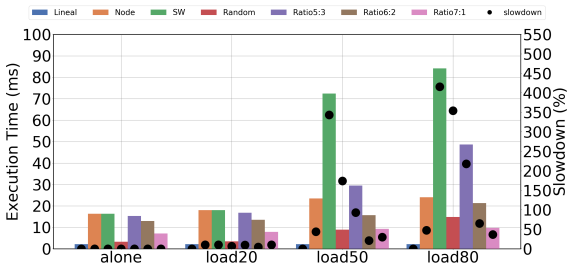
Fig. 8: Tiempos de ejecución y slowdown para tráfico hotspot de diferentes intensidades y con diferentes niveles de tráfico background hot-spots para las distintas políticas de mapeo.



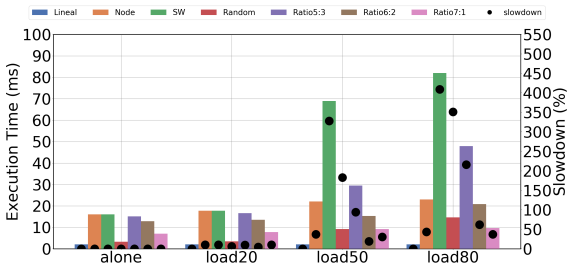
(a) Many-to-many tráfico 50 %. Tráfico background uniforme.



(b) Many-to-many tráfico 80 %. Tráfico background uniforme.



(c) Many-to-many tráfico 50 %. Tráfico background hotspot.



(d) Many-to-many tráfico 80 %. Tráfico background hotspot.

Fig. 9: Tiempos de ejecución y slowdown para tráfico many-to-may 50 % y 80 % intensidades y con distintos niveles de tráfico background uniforme y hot-spot para las distintas políticas de mapeo.

bos, stencil2D y stencil4D, sufre la mayor interferencia y llega a alcanzar los peores tiempos de ejecución, siendo lineal la que obtiene los mejores tiempos de ejecución seguida de la ratio7:1.

VII. Conclusiones

La red de interconexión juega un papel clave en las prestaciones alcanzadas por aplicaciones paralelas en los supercomputadores. La interferencia producida por el uso de los recursos compartidos depende del tráfico que presentan las aplicaciones y del mapeo de las tareas de estas aplicaciones a nodos de cómputo que puede modificar estos patrones. En este artículo hemos comprobado que para tráficos adversos a la topología como es el caso del patrón adversarial analizado, realizar un mapeo que transforme ese patrón beneficia a reducir la interferencia intra-aplicación. Sin embargo, si el patrón es muy local al grupo, hacer mapeos que exploten esa localidad sin modificarlo puede ayudar a reducir los tiempos de comunicación y por tanto a mejorar las prestaciones de la aplicación.

Por otra parte, en cuanto a la interferencia entre aplicaciones se ha podido observar que al tener puntos calientes en la red generados por tráfico hotspot, realizar un mapeo aleatorio puede aumentar la congestión de la red y ser contraproducente.

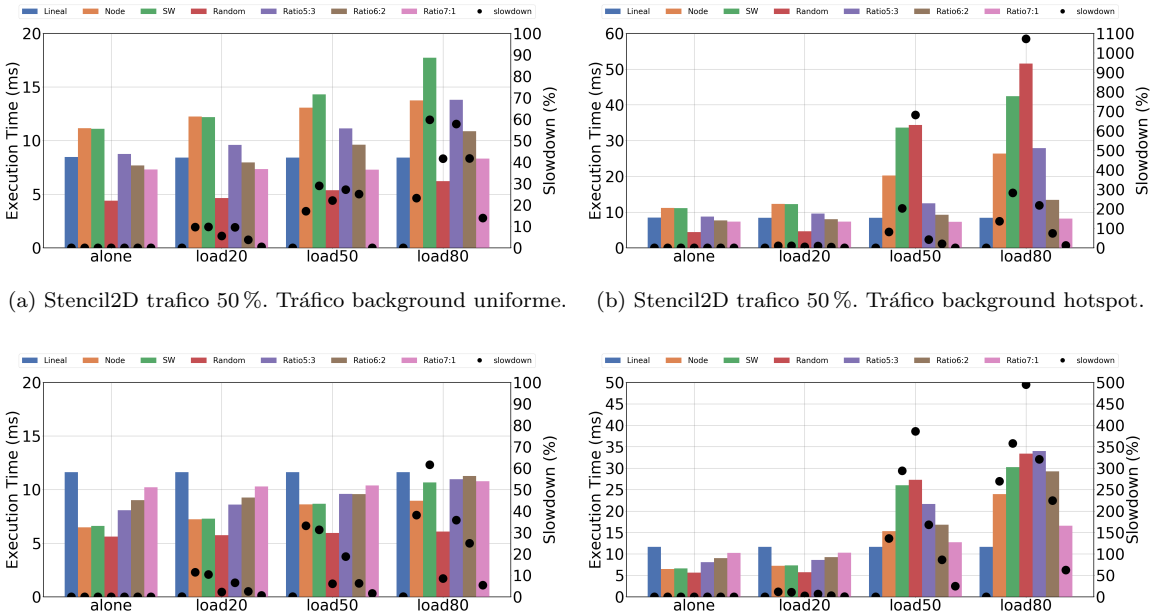
Como conclusión podemos afirmar que la elección de la mejor política de mapeo depende del patrón que exibe la aplicación pero también del patrón de tráfico que presentan las aplicaciones que se están ejecutando de manera concurrente.

Agradecimientos

Este trabajo ha sido financiado por MICIU/AEI/10.13039/501100011033/ FEDER, UE, a través de los proyectos PID2024-158682OB-C31 y PID2021-123627OB-C51, junto con la Generalitat Valenciana a través del proyecto CIPROM/2024/7.

Referencias

[1] Daniele De Sensi, Salvatore Di Girolamo, Kim H. McMahon, Duncan Roweth, and Torsten Hoefler, "An in-depth analysis of the slingshot interconnect," in *SC20:*



(a) Stencil2D trafico 50 %. Tráfico background uniforme.

(b) Stencil2D trafico 50 %. Tráfico background hotspot.

(c) Stencil4D trafico 50 %. Tráfico background uniforme.

(d) Stencil4D trafico 50 %. Tráfico background hotspot.

Fig. 10: Tiempos de ejecución y slowdown para tráfico stencil2D y stencil4D 50 % y con de tráfico background uniforme y hot-spot para las distintas políticas de mapeo.

International Conference for High Performance Computing, Networking, Storage and Analysis, 2020, pp. 1–14.

[2] John Kim, William J. Dally, Steve Scott, and Dennis Abts, "Technology-driven, highly-scalable dragonfly topology," in *35th International Symposium on Computer Architecture (ISCA 2008), June 21-25, 2008, Beijing, China*. 2008, pp. 77–88, IEEE Computer Society.

[3] L. Baba Arimilli, Ravi Arimilli, Vicente Chung, Scott Clark, Wolfgang E. Denzel, Ben C. Drerup, Torsten Hoefler, Jody B. Joyner, Jerry Lewis, Jian Li, Nan Ni, and Ramakrishnan Rajamony, "The PERCS high-performance interconnect," in *IEEE 18th Annual Symposium on High Performance Interconnects, HOTI 2010, Google Campus, Mountain View, California, USA, August 18-20, 2010*, Fabrizio Petrini, Dennis Abts, Ron Brightwell, Pavan Balaji, and Cyriel Minkenberg, Eds. 2010, pp. 75–82, IEEE Computer Society.

[4] Greg Faanes, Abdulla Bataineh, Duncan Roweth, Tom Court, Edwin Froese, Robert Alverson, Tim Johnson, Joe Kopnick, Mike Higgins, and James Reinhard, "Cray cascade: a scalable HPC system based on a dragonfly network," in *SC Conference on High Performance Computing Networking, Storage and Analysis, SC '12, Salt Lake City, UT, USA - November 11 - 15, 2012*, Jeffrey K. Hollingsworth, Ed. 2012, p. 103, IEEE/ACM.

[5] Alexander Shpiner, Zachy Haramaty, Saar Eliad, Vladimir Zdornov, Barak Gafni, and Eitan Zahavi, "Dragonfly+: Low cost topology for scaling datacenters," in *3rd IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era, HiPINEB@HPCA 2017, Austin, TX, USA, February 5, 2017*, Jesús Escudero-Sahuquillo and Pedro Javier García, Eds. 2017, pp. 1–8, IEEE Computer Society.

[6] David Skinner and William Kramer, "Understanding the causes of performance variability in hpc workloads," 11 2005, pp. 137– 149.

[7] Ana Jokanovic, José Carlos Sancho, Germán Rodríguez, Alejandro Lucero, Cyriel Minkenberg, and Jesús Labarta, "Quiet neighborhoods: Key to protect job performance predictability," in *2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2015, Hyderabad, India, May 25-29, 2015*. 2015, pp. 449–459, IEEE Computer Society.

[8] Martin Schulz, James F. Belak, Abhinav Bhatele, Peer-Timo Bremer, Greg Bronevetsky, Marc Casas, Todd Gamblin, Katherine E. Isaacs, Ignacio Laguna, Joshua A. Levine, Valerio Pascucci, David F. Richards, and Barry Rountree, "Performance analysis techniques for the exascale co-design process," in *Proceedings of the International Conference on Parallel Computing, ParCo 2013, 10-13 September 2013, Garching, Germany*, Michael Bader, Arndt Bode, Hans-Joachim Bungartz, Michael Gerndt, Gerhard R. Joubert, and Frans J. Peters, Eds. 2013, vol. 25 of *Advances in Parallel Computing*, pp. 19–32, IOS Press.

[9] Ana Jokanovic, Germán Rodríguez, José Carlos Sancho, and Jesús Labarta, "Impact of inter-application contention in current and future HPC systems," in *IEEE 18th Annual Symposium on High Performance Interconnects, HOTI 2010, Google Campus, Mountain View, California, USA, August 18-20, 2010*, Fabrizio Petrini, Dennis Abts, Ron Brightwell, Pavan Balaji, and Cyriel Minkenberg, Eds. 2010, pp. 15–24, IEEE Computer Society.

[10] Bogdan Prisacari, Germán Rodríguez, Marina García, Enrique Vallejo, Ramón Beivide, and Cyriel Minkenberg, "Performance implications of remote-only load balancing under adversarial traffic in dragonflies," in *Proceedings of the 8th International Workshop on Interconnection Network Architecture - On-Chip, Multi-Chip, INA-OCMC 2014, Vienna, Austria, January 22, 2014*, Giorgos Dimitrakopoulos, Sören Sonntag, José Flich, and Pascal Vivet, Eds. 2014, pp. 5:1–5:4, ACM.

[11] Abhinav Bhatele, Kathryn Mohror, Steven H. Langer, and Katherine E. Isaacs, "There goes the neighborhood: Performance degradation due to nearby jobs," in *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–12.

[12] Pedro Yébenes, Jesús Escudero-Sahuquillo, Pedro Javier García, and Francisco J. Quiles, "Towards modeling interconnection networks of exascale systems with omnet++," in *21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2013, Belfast, United Kingdom, February 27 - March 1, 2013*, pp. 203–207.

Análisis preliminar del rendimiento computacional de PhysiCell en función de los parámetros biológicos y computacionales

Antonio de Jesus Menchaca-Martinez[1,2], Johann Bourcier [1], María Villarroya-Gaudó [2]

Resumen— **PhysiCell es una herramienta de simulación celular cuyo rendimiento depende de numerosos parámetros biológicos y computacionales, lo que dificulta la selección de configuraciones eficientes. En este trabajo, presentamos un análisis preliminar de rendimiento orientado a arquitecturas de CPU con memoria compartida (OpenMP) que combina: (i) sensibilidad un-factor-a-la-vez (OFAT), (ii) impacto de interacciones paramétricas por pares, y (iii) escalabilidad multihilo bajo distintas frecuencias de difusión. Los resultados muestran que el tiempo de ejecución está dominado por la resolución temporal, la discretización espacial y el número de agentes. En particular, el tamaño de vóxel y Δt_{diff} se identifican como factores de mayor impacto en el rendimiento. Además, observamos que algunos parámetros típicamente biológicos también pueden modificar de forma significativa el coste computacional, por ejemplo, al alterar la dinámica poblacional. A partir de estos hallazgos, proponemos una caracterización en dos niveles de los parámetros de PhysiCell: rol funcional e impacto computacional, y un marco teórico para modelar el tiempo de ejecución. Finalmente, presentamos dos modelos preliminares de predicción, cuyos resultados iniciales confirman la viabilidad del enfoque y establecen la base para una guía de configuración orientada al rendimiento.**

Palabras clave— **Simulación multicelular, análisis de rendimiento, análisis de sensibilidad, Physicell, arquitectura de memoria compartida, OpenMP.**

I. Introducción

LA s simulaciones multicelulares multiescala se utilizan cada vez más en biología computacional para estudiar sistemas biológicos complejos *in silico* [1]. Entre estos enfoques, los modelos basados en agentes (ABMs), como PhysiCell, se emplean ampliamente para simular agentes biológicos (por ejemplo, células cancerígenas) como entidades discretas cuyo comportamiento está gobernado por un conjunto predefinido de reglas [2]. Este marco resulta idóneo para modelar sistemas biológicos reales, donde las interacciones célula-célula y las condiciones del microentorno son fundamentales para estudiar cómo progresa el cáncer en el tiempo y cómo responde a tratamientos terapéuticos [3]. Esta herramienta es utizlizada por grupos de investigacón en bioquímica y biomecánica y uno de los principales desafíos para estos investigadores es la necesidad de ejecutar flujos de trabajo iterativos que incluyen exploración de parámetros biológicos, calibración y campañas repetidas de simulación, lo que vuelve cada vez más importante el rendimiento de ejecución y la capacidad de anticiparlo.

PhysiCell sigue un enfoque híbrido: modela los sustratos en un espacio continuo y considera los agentes son discretos. Diferentes sustratos y tipos celulares pueden emularse utilizando propiedades específicas para definir distintos entornos de simulación. Sin embargo, esta flexibilidad conlleva una complejidad computacional significativa. Como resultado, el rendimiento computacional de PhysiCell depende de un amplio conjunto de factores, incluyendo parámetros del modelo, frecuencias de actualización numérica, configuraciones de compilación y la arquitectura hardware elegida. La implementación estándar está diseñada para ejecutarse en CPU con memoria compartida utilizando OpenMP, que es con la que hemos realizado este trabajo, ya que es la más útilizada en la actualidad. Extensiones posteriores han introducido aceleración para GPU [4] y ejecución en memoria distribuida MPI en PhysiCell-X [5].

Seleccionar los valores de configuración para modelar utilizando PhysiCell no es trivial, especialmente para usuarios sin un fuerte conocimiento computacional. Esto se vuelve particularmente limitante cuando las simulaciones deben repetirse múltiples veces para calibración o estudios de exploración a gran escala. Una caracterización computacional estructurada de PhysiCell contribuirá a reducir esta dificultad al identificar qué parámetros tienen mayor impacto en el rendimiento y aclarar cómo cambia el comportamiento de ejecución según la configuración elegida. La predicción del tiempo de ejecución también es valiosa, ya que permite anticipar el coste computacional y guiar decisiones de configuración orientadas al rendimiento antes de lanzar simulaciones a gran escala.

Hasta el momento, no hemos encontrado publicada una caracterización computacional sistemática de PhysiCell. En este trabajo, abordamos este nicho analizando PhysiCell desde una perspectiva orientada al rendimiento. Nuestras principales contribuciones son:

1. Un análisis de sensibilidad del tiempo de ejecución de tipo un-factor-a-la-vez (OFAT) con respecto a los parámetros de PhysiCell (Sección III-B).
2. Un análisis de escalabilidad paralela multihilo, incluyendo el efecto de la variación de la frecuencia temporal de difusión (Sección III-C).
3. Un marco de caracterización estructurado que clasifica los parámetros de PhysiCell según su

[1]LIUPPA, Université de Pau et des Pays de l'Adour, e-mail: adjmmartinez,johann.bourcier@univ-pau.fr.

[2]Instituto de Investigación en Ingeniería de Aragón, Universidad de Zaragoza, e-mail: mvg@unizar.es.

impacto computacional (Sección IV).
4. Un análisis del impacto del tiempo de ejecución debido a las interacciones paramétricas por pares conforme a la caracterización propuesta(Sección V).
5. Un diseño de marco teórico que permita desarrollar un modelo de predicción del tiempo de ejecución de PhysiCell (Sección VII).
6. Implementación de dos modelos preliminares y sus resultados (Sección VIII).

Además, esta caracterización y este análisis sirven como base para un modelo predictivo de tiempo de ejecución en desarrollo, destinado a guiar decisiones de configuración orientadas al rendimiento. En la Sección VII se explica la lógica utilizada para diseñar la primera versión de nuestro modelo predictivo.

II. Contexto Computacional de PhysiCell

A. Modelo computacional de PhysiCell

PhysiCell es un simulador multicelular basado en física diseñado como un marco híbrido basado en agentes acoplado con BioFVM para el transporte bioquímico [1]. El microentorno se representa mediante campos continuos de sustratos gobernados por ecuaciones en derivadas parciales (PDEs) de difusión-decaimiento, resueltas utilizando un método localmente unidimensional y el algoritmo de Thomas para matrices tridiagonales [5].

Los agentes celulares se modelan como agentes discretos fuera de red, centrados en su posición, cuyos estados se actualizan dinámicamente en función de las condiciones locales del entorno, fuerzas mecánicas e interacciones célula-célula [1]. Estos agentes pertenecen a la clase `Cell`, construida como una extensión de la clase `Basic_Agent` de BioFVM [6]. Esta formulación híbrida convierte a PhysiCell en un marco flexible para representar sistemas multicelulares en microentornos dinámicos.

B. Ejecución computacional y cuellos de botella

Para equilibrar la precisión biológica con la eficiencia computacional se utilizan tres pasos temporales para i) desacoplar y actualizar los procesos de difusión y transporte de sustratos (Δt_{diff}; por defecto 0.01 min), ii) la mecánica del movimiento celular (Δt_{mech}; por defecto 0.1 min) y iii) los procesos del estado celular (Δt_{cells}; por defecto 6 min) [1].Estos valores por defecto fueron definidos para problemas típicos en cáncer e ingeniería de tejidos, asegurando que las aproximaciones numéricas mantengan errores relativos iguales o menores al 5 % en comparación con modelos matemáticos exactos o altamente refinados [1]. Dada esta separación de escalas, el *solver* de difusión se ejecuta 10 veces por cada actualización de mecánica y 600 veces por cada actualización de fenotipo. Esta separación temporal influye fuertemente en la distribución del tiempo de ejecución y en la sensibilidad del rendimiento.

Dado que la difusión también se ejecuta con mayor frecuencia que otras funciones, se convierte en el principal cuello de botella del tiempo de ejecución. Implementaciones con OpenACC han mostrado aceleraciones cercanas a 40× en esta etapa [4]. A mayor escala, PhysiCell-X introduce MPI+OpenMP, aunque la difusión sigue siendo el principal reto de escalabilidad.

C. Modelado de rendimiento en HPC

C.1 Paralelismo y Ley de Amdahl

El paralelismo en memoria compartida se implementa por defecto mediante OpenMP, se permite la ejecución en estaciones de trabajo multinúcleo y nodos individuales de HPC [1], [2].

Las actualizaciones a nivel celular pueden distribuirse entre hilos, pero en la etapa de difusión, cada sistema tridiagonal sigue siendo inherentemente secuencial, lo que limita la escalabilidad [5], [7].

En el caso de PhysiCell, la aceleración está fuertemente influenciada por la etapa de actualización de la difusión. Cada sistema tridiagonal que debe resolverse es inherentemente secuencial. De acuerdo con la ley de Amdahl, este factor es un componente clave que limita la escalabilidad al aumentar el número de hilos [7].

C.2 Sincronización, contención y coherencia

La sincronización impacta negativamente la eficiencia durante la ejecución en paralelo que se produce cuando los distintos hilos no pueden trabajar simultáneamente, ya que necesitan esperar entre sí para continuar el algoritmo. También aparecen elevados costes de sincronización en OpenMP cuando la planificación de hilos se distribuye entre múltiples sockets NUMA, lo que ralentiza drásticamente el tiempo entre regiones paralelas [7].

La contención ocurre cuando múltiples hilos en paralelo intentan acceder o modificar un único recurso compartido del sistema exactamente al mismo tiempo, lo que provoca que el hardware limite o encole las solicitudes. Los análisis de perfilado demostraron que existe una caída severa en la frecuencia del procesador durante la fase de actualización mecánica de PhysiCell. Los autores identificaron que millones de operaciones en paralelo solicitaban simultáneamente al sistema operativo asignar y liberar pequeños bloques de memoria, generando una gran contención en el asignador de memoria compartida [2].

A medida que avanza una simulación de PhysiCell, las células se dividen y las nuevas células se añaden al final del vector de memoria. En consecuencia, células físicamente próximas en el espacio visual de la simulación terminan siendo almacenadas de forma dispersa y alejada dentro de la memoria física del sistema. La penalización en el rendimiento se produce debido a la falta de localidad de datos. Esto reduce significativamente las instrucciones por ciclo (IPC) e impacta al rendimiento [2].

III. Metodología Experimental

A. Configuración experimental del análisis

Todos los experimentos en CPU se realizaron en un sistema de memoria compartida utilizando OpenMP para la ejecución paralela tanto en configuraciones monohilo como multihilo. El sistema está compuesto por un procesador Intel Core Ultra 7 155H con 16 núcleos y 22 hilos hardware (6 núcleos de alto rendimiento, 8 núcleos eficientes y 2 núcleos eficientes de bajo consumo), con frecuencias de hasta 4.8 GHz.

Dada la arquitectura heterogénea del procesador, restringimos la ejecución a los seis núcleos de alto rendimiento (12 hilos) mediante afinidad de proceso, enlazando a las CPUs lógicas de los P-cores. Esto evita que al utilizar OpenMP los hilos se distribuyan de forma impredecible entre núcleos P/E/LP-E y mejora el comportamiento de escalabilidad.

B. Análisis de sensibilidad de parámetros

Se realizó un análisis de sensibilidad de tipo unfactor-a-la-vez (OFAT) utilizando la plantilla integrada "heterogeneity" y sus valores iniciales como referencia (baseline). Limitamos el tiempo total de simulación a 1440 minutos para reducir el coste computacional manteniendo datos representativos. Se ejecutaron cinco repeticiones por configuración para minimizar posibles sesgos debidos a valores atípicos y ruido. Nuestro espacio de valores de parámetros estuvo compuesto por 583 valores, definidos considerando aquellos utilizados en las plantillas integradas de PhysiCell que comparten los mismos parámetros. En total, se realizaron 3020 simulaciones, incluyendo 20 ejecuciones baseline, y se calculó la mediana del tiempo de ejecución (wall time) para cada configuración.

El porcentaje de desviación relativa del tiempo de ejecución se calculó como la diferencia entre la mediana del tiempo de ejecución y la mediana del baseline, dividida por la mediana del baseline. Esta métrica expresa la desviación de rendimiento de cada valor individual de parámetro respecto a la referencia fija. Para cada parámetro, se seleccionaron los valores que produjeron la desviación mínima y máxima. A partir de estos valores límite, se calculó el delta relativo total y se ordenaron los parámetros de mayor a menor impacto.

C. Análisis de escalabilidad multihilo

Se realizó un análisis de escalabilidad paralela utilizando el mismo template "heterogeneity" para comparar cómo OpenMP distribuye la carga de trabajo a medida que varían el número de hilos y la frecuencia temporal de difusión. Para este análisis, se consideraron:

- **Frecuencia temporal de difusión:** $\Delta t_{\text{diff}} \in \{0{,}01, 0{,}05, 0{,}1\}$.
- **Barrido de hilos:** $t \in \{1, 2, 4, 6, 8, 10, 12\}$ hilos.
- **Baseline:** $\Delta t_{\text{diff}} = 0{,}01$ & $t = 1$.

IV. Marco de caracterización de parámetros

Basándonos en los análisis presentados en las secciones anteriores, proponemos una caracterización en dos niveles de los parámetros de PhysiCell. En primer lugar, los parámetros se agruparon según su rol funcional: *biológicos* y *computacionales*. Los parámetros biológicos describen principalmente el sistema modelado, mientras que los parámetros computacionales controlan el comportamiento de ejecución. Al interpretar nuestros resultados de sensibilidad desde una perspectiva computacional, observamos que varios parámetros que inicialmente podrían considerarse puramente biológicos también tienen un impacto directo en el rendimiento computacional (por ejemplo, al incrementar el número de agentes o de ecuaciones en derivadas parciales a resolver), y viceversa. Además, los parámetros que funcionan como metadatos y que no afectan los resultados de la simulación, se excluyeron de la caracterización de segundo nivel orientada al rendimiento.

Los parámetros restantes se reclasificaron posteriormente según el mecanismo principal a través del cual afectan al tiempo de ejecución. Un parámetro se asignó a una categoría no solo en función de su significado semántico, sino también de cómo una variación en su valor impacta la carga computacional subyacente o el coste de ejecución. Este segundo nivel de caracterización define cinco subcategorías orientadas al rendimiento:

1. **Resolución temporal**. Los parámetros de esta categoría modifican la frecuencia de actualización de las funciones de difusión, mecánica o fenotipo. Su impacto en el rendimiento proviene del aumento o la reducción de la carga algorítmica total ejecutada durante la simulación.
2. **Escala espacial**. Esta categoría incluye parámetros que modifican el tamaño o la discretización del dominio computacional, la dimensión de la simulación (2D/3D) o el número de sustratos. Su efecto en el rendimiento se refleja principalmente en la carga de trabajo del *solver* de difusión de BioFVM, donde el número de ecuaciones en derivadas parciales a resolver escala linealmente con el número de vóxeles y sustratos.
3. **Número de agentes**. Los parámetros se asignaron a esta categoría cuando influyen en el número de células modeladas en la simulación, ya sea de forma inicial o dinámica mediante procesos de proliferación y muerte. Su impacto en el tiempo de ejecución está asociado con la cantidad de trabajo a nivel de agente realizado por las actualizaciones de difusión, mecánica y fenotipo, que escala linealmente con el número de células.
4. **Arquitectura hardware**. Esta categoría considera parámetros relacionados con la configuración de ejecución, como ajustes de compilación, paralelismo a nivel de hilos o arquitectura

hardware objetivo. Estos parámetros no necesariamente modifican la cantidad de trabajo biocomputacional simulado, pero afectan la eficiencia con la que se ejecuta.

5. **Entrada/Salida (I/O)**. Los parámetros de esta categoría controlan la frecuencia y el tipo de datos almacenados en disco, como capturas instantáneas SVG, salidas completas de simulación y archivos de respaldo. Su impacto en el rendimiento proviene del sobrecoste asociado a las operaciones de almacenamiento.

Además de esta categorización orientada al rendimiento, a cada parámetro se le asignó un nivel cualitativo de impacto (*bajo*, *medio* o *alto*) en función de los resultados de sensibilidad obtenidos en los análisis previos. Esta caracterización estructura el espacio de parámetros de PhysiCell desde una perspectiva de rendimiento, ayuda a explicar por qué algunos parámetros tienen efectos más significativos que otros y proporciona una base para futuros modelos predictivos del tiempo de ejecución. El Apéndice X presenta la categorización completa de parámetros.

Esta caracterización no solo permite interpretar los resultados del análisis OFAT, sino que también proporciona la base estructural para definir las categorías consideradas en el análisis de interacciones y, posteriormente, en la construcción del modelo predictivo.

V. Análisis de interacciones por pares entre categorías

Para analizar el rendimiento de las interacciones de los distintos parámetros, se profundizó en un análisis por pares entre las categorías previamente definidas. Se utilizaron los parámetros con mayor impacto computacional según nuestro análisis de sensibilidad. En la Tabla I, se presenta el espacio de valores definido. En lugar de explorar el impacto parámetro por parámetro de manera aislada, se analizó cómo interactúan las categorías que afectan al coste computacional.

Tabla I: Categorías de parámetros y valores evaluados en el análisis de pares.

Categoría	Parámetro	Valores
temporal_resolution	dt_diffusion	0.0001, 0.001, 0.01, 0.1
temporal_resolution	dt_mechanics	0.0001, 0.001, 0.01, 0.1, 1
temporal_resolution	dt_phenotype	0.06, 0.6, 6, 60
spatial_scale	dx/dy/dz	5, 10, 20, 30
spatial_scale	substrates	1, 10, 50, 100
spatial_scale	use_2D	true, false
agent_count	initial_cells	0, 100, 1000, 10000, 100000
hardware	threads	1, 2, 4, 6, 8, 10, 12
io	full_data_interval & svg_interval	1, 5, 15, 60

A. Muestreo bajo el método de Latin Hypercube

Para cada par de categorías, existen distintos parámetros y valores posibles. La dimensionalidad del espacio de búsqueda depende del número de parámetros involucrados en ambas categorías a analizar. Un análisis cartesiano completo de este espacio resulta muy costoso. Por lo tanto, para realizar un muestreo eficiente del espacio, se utilizó el método de muestreo de Latin Hypercube Sampling (LHS) [8]. Este método garantiza que haya cobertura uniforme de cada dimensión sin necesidad de explorar exhaustivamente todas las combinaciones posibles. Dado que el número de muestras depende de la dimensión del espacio considerado, y que además cada categoría puede tener una dimensionalidad diferente, se implementó un muestreo dinámico que varía en función de dicha dimensión. Este número dinámico de muestras busca cubrir de forma representativa el espacio de valores. Cada muestra generada corresponde a una configuración completa del simulador, la cual se ejecuta de forma independiente para medir su rendimiento.

B. Análisis del coste computacional

Además del tiempo transcurrido en cada simulación, se obtuvieron datos complementarios (p. ej., número de células iniciales y finales) y se calcularon ciertas métricas (p. ej., número de vóxeles y densidad temporal), las cuales ayudan a analizar la correlación entre los distintos factores con el tiempo de ejecución observado.

C. Infraestructura experimental

Dentro de nuestro análisis paramétrico se implementó una infraestructura experimental automatizada para garantizar la generación sistemática, la ejecución controlada y la reproducibilidad de las simulaciones. En base al muestreo del espacio obtenido por el método de LHS, se generan de manera automática los archivos XML con las configuraciones específicas de cada simulación. Durante la ejecución de cada simulación se extraen métricas como el tiempo de ejecución, además de la recolección de telemetría del sistema como frecuencia y rendimiento del procesador. Para contrarrestar comportamientos atípicos, se procedió con un esquema de 5 repeticiones por configuración. Además, se implementan fases de reposo entre ejecuciones, con el fin de mitigar efectos de variabilidad. Finalmente, cada ejecución es completamente reproducible gracias al almacenamiento de las configuraciones generadas, parámetros de ejecución y resultados asociados.

VI. Resultados

A. Clasificación de parámetros

Nuestros resultados mostrados en la Figura 1 indican que el tiempo de ejecución está fuertemente influenciado por parámetros relacionados con los pasos temporales, la discretización espacial y el número de agentes. En particular, el paso temporal de difusión y el tamaño del voxel se identificaron como algunos de los parámetros más influyentes, alcanzando un porcentaje de sensibilidad absoluto de 175 %.

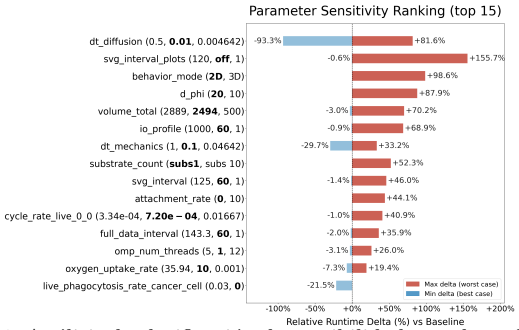


Fig. 1: Análisis de clasificación de sensibilidad para los valores del espacio de templates integrados de PhysiCell.

Además, entre los parámetros que más impactan el rendimiento en la Figura 1, `d_phi` proporciona un ejemplo representativo de un parámetro con impacto tanto biológico como computacional. En nuestro análisis, `d_phi` controla el tamaño del voxel al variar la discretización espacial (dx, dy, dz) del dominio de BioFVM. Cambiar este parámetro no sólo modifica la carga computacional del *solver* de difusión, sino que también afecta la evolución biológica de la población celular simulada.

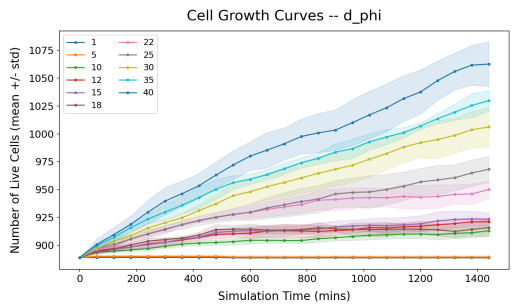


Fig. 2: Dinámica de proliferación celular para diferentes tamaños de voxel.

La Figura 2 muestra las curvas de proliferación celular obtenidas para diferentes valores de `d_phi`. Los tamaños de voxel mayores a 20 micras incrementan la tasa en la que las células se reproducen. Por lo que demuestra que este parámetro tiene también un impacto directo en el resultado biológico del modelo.

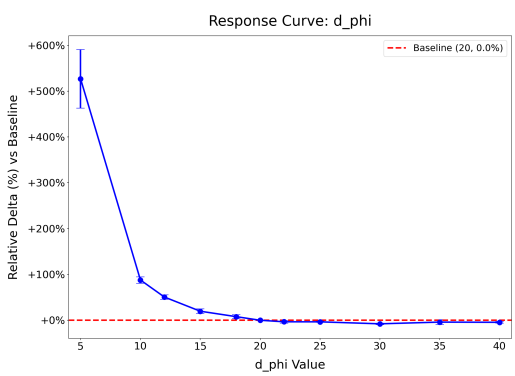


Fig. 3: Desviación relativa del tiempo de ejecución por tamaño de voxel respecto al baseline.

La Figura 3 muestra que la desviación relativa del tiempo de ejecución respecto a la configuración de referencia de `d_phi`, aumenta de forma exponencial por debajo del umbral sugerido de 10 micras. De manera más general, nuestros resultados muestran que incluso parámetros habitualmente considerados biológicos afectan significativamente al rendimiento, especialmente cuando modifican la evolución de la población celular a lo largo del tiempo. A partir de estas observaciones, proponemos un marco de caracterización en dos niveles para organizar los parámetros de PhysiCell según su rol funcional y su impacto computacional.

B. Escalabilidad

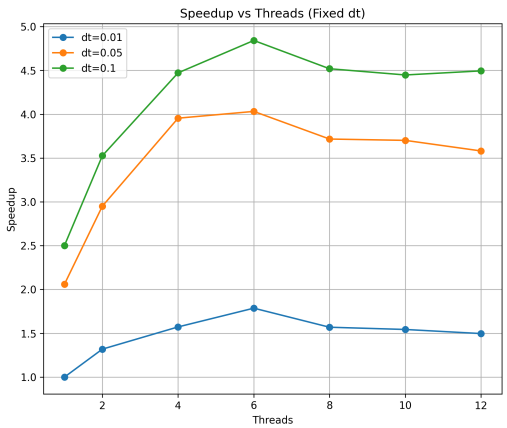


Fig. 4: Análisis de aceleración: número de hilos vs frecuencia temporal de difusión.

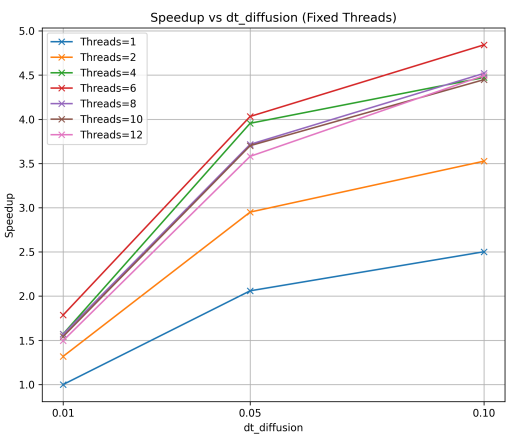


Fig. 5: Interpolación transversal para los valores de la frecuencia temporal de difusión.

Los resultados obtenidos al analizar la aceleración obtenida al ejecutar el modelo en paralelo con diferentes hilos, se calculó y representó gráficamente en las Figuras 4 y 5. Se observan mayores aceleraciones para valores más elevados de la frecuencia temporal de difusión, lo cual es consistente con la reducción de la carga algorítmica durante la ejecución. Una vez alcanzada la aceleración óptima para cada configuración, el comportamiento de degradación se atribuye a la contención de memoria, la coherencia de caché y los costes de sincronización durante la ejecución paralela.

Se observó que la escalabilidad paralela permanece limitada en gran medida por la etapa de difusión, la cual, debido a su naturaleza algorítmica, solo permite que una parte de los cálculos se ejecuten de manera

secuencial. Esto limita la escalabilidad al distribuir la carga de trabajo en múltiples hilos.

C. Primeras observaciones de las interacciones paramétricas

Uno de los objetivos del análisis de interacciones paramétricas es observar específicamente cómo los parámetros pertenecientes a las diferentes categorías moldean de forma no homogénea la carga computacional en función de los valores establecidos. Nuestros resultados permitieron identificar qué conjunto de parámetros tiene mayor y menor impacto sobre los tiempos de simulación. Este análisis sirvió de base para comenzar el modelado del predictor y para evaluar qué componentes del coste computacional requieren una formulación más estructurada.

En conjunto, estos resultados permiten identificar qué mecanismos dominan el coste computacional de PhysiCell y proporcionan una base cuantitativa para formular un modelo de predicción del tiempo de ejecución.

VII. Hacia un modelo predictivo

En esta sección definimos un marco teórico que establece la formulación conceptual de las funciones consideradas para construir una primera versión de un modelo predictivo de rendimiento. Además, se implementaron dos modelos preliminares, basados en Ridge y XGBoost, para poner en práctica el marco teórico propuesto y obtener los primeros resultados que orienten su optimización.

A. Formulación conceptual

PhysiCell combina múltiples operaciones numéricas que se distribuyen en diferentes escalas espaciales y temporales, incluyendo interacciones mecánicas entre células, difusión, y procesos celulares. Posterior a la categorización y análisis que realizamos en secciones anteriores, observamos que gran parte del tiempo de ejecución se produce por la interacción entre factores como carga algorítmica, la eficiencia de ejecución en hardware y el coste de entrada/salida (I/O). Esta descomposición refleja cómo la carga algorítmica interactúa con la paralelización, mientras que el coste de I/O es principalmente aditivo. En base a esto, definimos el tiempo de ejecución predicho, $\hat{T}$, donde x representa la configuración completa de la simulación y t es el número de hilos utilizados.

Proponemos una formulación multiplicativa, ya que refleja cómo la carga de trabajo, W_{core}, define la cantidad total de operaciones a ejecutar, mientras que el término de eficiencia paralela, G_{parallel}, modula el tiempo efectivo de ejecución en función del hardware. Finalmente, $T_{\text{I/O}}$ hace referencia al coste de entrada/salida (I/O).

$$\hat{T}(x,t) = W_{\text{core}}(x) \cdot G_{\text{parallel}}(x,t) + T_{\text{I/O}}(x) \quad (1)$$

B. Eficiencia paralela

Se ha definido una función para capturar el escalado ideal tipo Amdahl y la degradación del rendimiento debida a la contención, la sincronización y la presión de memoria que observamos al realizar nuestro análisis.

$$G_{\text{parallel}}(x,t) = \frac{1 + p(x)(t-1)}{t} \cdot e^{\,c(x)\cdot \max(0,t-t_0(x))} \quad (2)$$

donde $p(x)$ describe la fracción de trabajo que no escala idealmente, $c(x)$ captura la degradación por contención, sincronización y presión de memoria, y $t_0(x)$ indica el número de hilos a partir del cual se alcanza el mejor rendimiento. Su dependencia de x refleja que estos efectos varían con las características del *workload*.

C. Descomposición de la carga de trabajo total

Para definir $W_{\text{core}}(x)$ se utiliza la categorización de parámetros introducida en la Sección IV. Esto permite mapear de forma estructurada los parámetros con su impacto computacional. Por lo tanto, la carga de trabajo total se compone de la suma de la carga temporal (W_{temp}), espacial (W_{spatial}), y de agentes (W_{agent}):

$$W_{\text{core}}(x) = W_{\text{temp}}(x) + W_{\text{spatial}}(x) + W_{\text{agent}}(x) \quad (3)$$

C.1 Resolución temporal

Los parámetros temporales determinan la frecuencia de ejecución de cada fase (difusión, mecánica y procesos celulares). Por lo tanto, para representar el número de iteraciones por fase definimos:

$$\begin{aligned} N_{\text{diff}} &= \frac{T_{\max}}{\Delta t_{\text{diff}}}, \\ N_{\text{mech}} &= \frac{T_{\max}}{\Delta t_{\text{mech}}}, \\ N_{\text{phen}} &= \frac{T_{\max}}{\Delta t_{\text{phen}}} \end{aligned} \quad (4)$$

donde $T_{\max}$ es el tiempo total de la simulación, mientras que Δt_{diff}, Δt_{mech} y Δt_{phen} denotan, respectivamente, los pasos temporales asociados a difusión, mecánica y procesos celulares definidos. Finalmente, N_{diff}, N_{mech} y N_{phen} expresan el número de veces que cada fase es ejecutada durante una simulación completa.

C.2 Escala espacial

Los parámetros espaciales afectan principalmente el coste de resolver las ecuaciones diferenciales parciales involucradas durante el cálculo de la difusión. El dominio espacial se discretiza en vóxeles, por lo que se calcula el número de vóxeles por cada eje simulado:

$$N_i = \frac{x_i^{\text{máx}} - x_i^{\text{mín}}}{d_i}, \quad i \in \{x,y,z\} \quad (5)$$

donde $x_i^{\text{máx}}$ y $x_i^{\text{mín}}$ son los límites del dominio en la dirección i, d_i es la dimensión del voxel en cada eje i y N_i el número de divisiones cada eje.

$$N_{\text{vox}} = \prod_{i \in \{x,y,z\}} N_i \tag{6}$$

donde N_{vox} representa el número total de vóxeles de la malla espacial.

La carga computacional escala dependiendo de la resolución y dimensión espacial, además del número de sustratos definidos. Sea N_{sub} el número de sustratos.

$$W_{\text{spatial}} \propto N_{\text{diff}} \cdot N_{\text{vox}} \cdot N_{\text{sub}} \tag{7}$$

donde W_{spatial} representa al componente espacial de la carga de trabajo, N_{diff} el número total de actualizaciones de difusión en la simulación, N_{vox} el número total de vóxeles del dominio y N_{sub} el número de sustratos.

Esta relación expresa que el coste computacional del *solver* de difusión aumenta con el número de veces que se ejecuta la difusión que a su vez aumenta el número de ecuaciones a resolver.

C.3 Coste por agentes

El comportamiento celular, incluyendo las funciones de mecánica y fenotipo, escala de forma lineal con el número de agentes y la frecuencia de actualización. Sea N_{cells} el número de células:

$$W_{\text{agent}} \propto N_{\text{cells}} \cdot (N_{\text{mech}} + N_{\text{phen}}) \tag{8}$$

donde W_{agent} representa la componente de la carga de trabajo asociada a las operaciones a nivel celular, N_{cells} el número de células presentes en la simulación, N_{mech} el número total de actualizaciones mecánicas y N_{phen} el número total de actualizaciones de fenotipo. Esta relación expresa que el coste computacional crece con el número de agentes que deben actualizarse y con la frecuencia con la que dichas actualizaciones se ejecutan.

C.4 Modelado del coste de I/O

Las operaciones de salida introducen un coste extra significativo, especialmente en simulaciones de gran escala, donde tanto la frecuencia elegida de escritura como el volumen de datos generados influyen directamente en el tiempo total de ejecución. En este trabajo, se ha considerado únicamente el coste asociado al almacenamiento de los estados de la simulación en ficheros `.mat`, ya que estos forman parte del proceso estándar de simulación. Por el contrario, la generación de visualizaciones se excluye del análisis ya que forma parte de la etapa de postprocesado. A diferencia del cómputo, este coste se modela como un término aditivo independiente.

$$T_{\text{I/O}}(x) = \boldsymbol{\theta}_{\text{io}} \cdot \boldsymbol{\phi}_{\text{io}}(x) \tag{9}$$

donde el vector de características del coste por I/O se define como:

$$\boldsymbol{\phi}_{\text{io}} = \begin{bmatrix} N_{\text{save,full}} \\ N_{\text{save,full}} \cdot N_{\text{cells}} \end{bmatrix} \tag{10}$$

donde $N_{\text{save,full}}$ representa el número total de veces que se almacenan los datos completos de la simulación, y N_{cells} representa una aproximación del número medio de células presentes durante la simulación.

$$N_{\text{save,full}} = \frac{T_{\max}}{\Delta t_{\text{full}}} \tag{11}$$

donde $T_{\max}$ es el tiempo total de simulación y Δt_{full} es la frecuencia de almacenado de datos.

Debido a que el volumen de datos almacenado en cada snapshot depende del número de células presentes en la simulación, se utiliza N_{cells} como una aproximación del tamaño de los datos, permitiendo capturar indirectamente el coste temporal de las operaciones de I/O.

D. Parámetros paralelos estructurados

Los parámetros de escalado se modelan como funciones de la carga de trabajo:

$$p(x) = \boldsymbol{\beta}_p \cdot \boldsymbol{\psi}_p(x), \tag{12}$$

$$c(x) = \boldsymbol{\beta}_c \cdot \boldsymbol{\psi}_c(x), \tag{13}$$

$$t_0(x) = \boldsymbol{\beta}_t \cdot \boldsymbol{\psi}_t(x) \tag{14}$$

donde $p(x)$ describe la fracción efectiva de trabajo que no escala idealmente con el número de hilos, $c(x)$ mide la intensidad de la degradación del rendimiento debida a contención, sincronización y presión de memoria, y $t_0(x)$ representa el umbral de hilos a partir del cual comienza el retroceso de la escalabilidad. Los vectores $\boldsymbol{\beta}$ contienen los coeficientes del modelo, mientras que $\boldsymbol{\psi}(x)$ agrupa características derivadas de la configuración, como el tamaño espacial, el número de agentes, el número de sustratos y la dimensionalidad. Esta formulación permite expresar la eficiencia paralela como una función de la carga de trabajo sin ajustar estos parámetros de forma independiente para cada configuración.

D.1 Predictor final

Nuestro modelo implementa una descomposición basada en las diferentes categorías definidas en secciones anteriores, separando sus diversos efectos sobre la carga algorítmica, la eficiencia de paralelización y las cargas adicionales de I/O. Este desglose nos proporciona una interpretación directa del impacto que cada grupo de parámetros genera en el tiempo de ejecución. La estructura final se define de la siguiente manera:

$$\hat{T}(x,t) = \left[\boldsymbol{\theta} \cdot \boldsymbol{\phi}_{\text{compute}}(x)\right] \cdot G_{\text{parallel}}(x,t) + \boldsymbol{\theta}_{\text{io}} \cdot \boldsymbol{\phi}_{\text{io}}(x) \tag{15}$$

donde $\boldsymbol{\phi}_{\text{compute}}(x)$ representa un vector de características que capturan el número de operaciones de cada fase (difusión, mecánica y procesos celulares), y $\boldsymbol{\theta}$ sus costes computacionales asociados.

La carga de trabajo determina la cantidad total de operaciones a ejecutar, mientras que la función de

eficiencia paralela ajusta el tiempo efectivo de ejecución en función del hardware elegido. Finalmente, esta formulación permite predecir el tiempo de ejecución a partir de configuraciones definidas, facilitando la exploración eficiente del espacio de parámetros y proporcionando una base para la optimización del rendimiento en simulaciones multiescala.

VIII. Resultados preliminares del predictor

Con el fin de validar de manera preliminar la formulación presentada en los apartados C y D de esta sección, se implementaron dos modelos preliminares para predecir el tiempo de ejecución a partir del subconjunto actualmente disponible del análisis de interacciones paramétricas: un modelo lineal regularizado de tipo Ridge, basado en variables derivadas de la carga computacional, y un modelo no lineal basado en XGBoost. Debido a que cada configuración se ejecutó 5 veces, para reducir el ruido experimental, se obtuvo la mediana del tiempo de ejecución, obteniéndose 42 configuraciones estructurales únicas. Dado el tamaño aún reducido del conjunto disponible, la validación se realizó mediante *Repeated K-Fold* con 5 particiones y 3 repeticiones.

La Tabla II muestra los resultados obtenidos. XGBoost superó claramente al modelo Ridge en todas las métricas evaluadas, lo que sugiere la existencia de interacciones no lineales y efectos combinados que no quedan todavía recogidos por la formulación lineal inicial.

Tabla II: Resultados preliminares del predictor de tiempo de ejecución.

Métrica	**Ridge**	**XGBoost**
Log-MAE	1.8693 ± 1.4703	0.8162 ± 0.2629
Log-RMSE	3.7027 ± 3.8945	1.0849 ± 0.3602
sMAPE	78.25 % ± 13.60 %	67.61 % ± 18.03 %

La Figura 6 confirma esta observación al comparar los tiempos predichos frente a los tiempos reales. Se observa que las predicciones de XGBoost se concentran más cerca de la diagonal ideal, mientras que el modelo lineal de tipo Ridge presenta una dispersión considerablemente mayor, especialmente en configuraciones que presentan un alto coste computacional. Este comportamiento indica que la estructura del modelo propuesto aún no modela adecuadamente las no linealidades asociadas a la dimensionalidad, al número de sustratos y a la interacción entre carga de trabajo y eficiencia de ejecución.

IX. Conclusiones y trabajo futuro

En este artículo, presentamos los primeros resultados de un análisis del rendimiento computacional de PhysiCell basado en sensibilidad sobre una arquitectura de CPU de memoria compartida. En primer lugar, identificamos los parámetros que, de forma individual, tienen mayor impacto en el rendimiento computacional. Además, propusimos una clasificación desde una perspectiva orientada al rendimiento. Posteriormente, a partir de la caracterización realizada, extendimos el análisis de sensibilidad para evaluar la interacción paramétrica y su impacto en el rendimiento. Además de identificar los parámetros más influyentes, capturamos el conjunto de configuraciones que más afectan al rendimiento. El uso del método LHS nos permitió explorar de forma eficiente el espacio paramétrico y generar métricas distribuidas. Se implementaron los dos primeros modelos preliminares para predecir el tiempo de ejecución. Una vez que concluya toda la campaña de experimentación y se cuente con los resultados completos, continuaremos optimizando el modelo para obtener mejores predicciones.

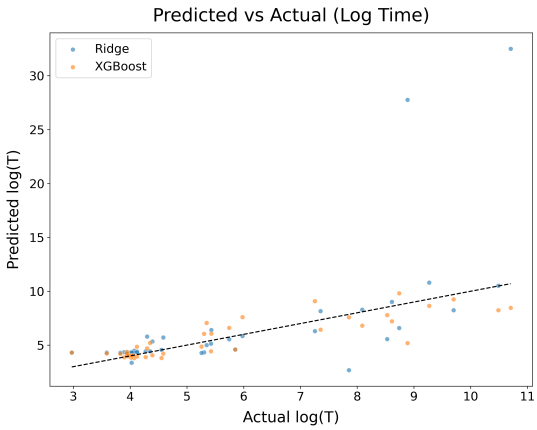


Fig. 6: Comparación entre tiempo real y tiempo predicho en escala logarítmica para los modelos Ridge y XGBoost. La línea discontinua representa la predicción ideal.

El análisis de errores mostró que el modelo preliminar lineal presenta peor rendimiento en configuraciones 3D y en escenarios con un alto número de sustratos, donde la componente espacial del coste computacional deja de comportarse de forma lineal. Debido a que el análisis de interacciones paramétricas aún no ha sido completado, la componente de paralelización no ha podido modelarse todavía de forma explícita en interacción con el resto de variables. En esta primera versión, el número de hilos se consideró como una variable de entrada del modelo, en lugar de representarse mediante un bloque específico de eficiencia paralela.

Estos resultados no invalidan el enfoque físico-informado propuesto en la sección anterior, sino que indican que los modelos propuestos todavían no alcanzan a representar correctamente el comportamiento del simulador. Como siguiente ajuste, se propone refinar la componente espacial del modelo, incorporar de manera más explícita el efecto de la dimensionalidad y avanzar hacia una formulación estructurada del bloque de eficiencia paralela. El modelo XGBoost se mantendrá como referencia para cuantificar el margen de mejora del predictor interpretable.

La formulación y modelado de este predictor de rendimiento se alinea con la estructura computacional de PhysiCell. La separación entre la carga computacional, la eficiencia de la escalabilidad por paralelización y los costes provenientes de las entradas y salidas, además de hacer una distinción de los diferentes componentes temporales, espaciales y de agen-

tes, fortalece al modelo ya que lo hace replicable y portable para otros simuladores biológicos.

Como trabajo futuro, planeamos el análisis de rendimiento modificando los parámetros de compilación. Asimismo, escalaremos el estudio utilizando el Clúster de Computación de Alto Rendimiento (HPC) del I3A, además de ejecutar simulaciones con memoria distribuida basada en MPI y aceleración con GPU. Finalmente, continuaremos con la implementación de nuestro modelo predictivo de tiempo de ejecución. Una vez obtenidos los resultados del análisis paramétrico, completaremos la primera versión del modelo para arquitecturas de CPU con memoria compartida.

Referencias

[1] Ahmadreza Ghaffarizadeh, Randy Heiland, Samuel H. Friedman, Shannon M. Mumenthaler, and Paul Macklin, "PhysiCell: An open source physics-based cell simulator for 3-D multicellular systems," *PLOS Computational Biology*, vol. 14, no. 2, pp. e1005991, Feb. 2018.

[2] Marc Clascà, Marta Garcia-Gasulla, Arnau Montagud, Jose Carbonell Caballero, and Alfonso Valencia, "Lessons learned from a performance analysis and optimization of a multiscale cellular simulation," in *Proceedings of the Platform for Advanced Scientific Computing Conference*, June 2023, pp. 1–10, arXiv:2306.11544 [cs].

[3] John Metzcar, Yafei Wang, Randy Heiland, and Paul Macklin, "A Review of Cell-Based Computational Modeling in Cancer Biology," *JCO Clinical Cancer Informatics*, , no. 3, pp. 1–13, Dec. 2019.

[4] Matt Stack, Paul Macklin, Robert Searles, and Sunita Chandrasekaran, "OpenACC Acceleration of an Agent-Based Biological Simulation Framework," Oct. 2021, arXiv:2110.13368 [cs].

[5] Gaurav Saxena, Miguel Ponce-de Leon, Arnau Montagud, David Vicente Dorca, and Alfonso Valencia, "BioFVM-X: An MPI+OpenMP 3-D Simulator for Biological Systems," in *Computational Methods in Systems Biology*, Eugenio Cinquemani and Loïc Paulevé, Eds., vol. 12881, pp. 266–279. Springer International Publishing, Cham, 2021, Series Title: Lecture Notes in Computer Science.

[6] Ahmadreza Ghaffarizadeh, Samuel H. Friedman, and Paul Macklin, "BioFVM: an efficient, parallelized diffusive transport solver for 3-D biological simulations," *Bioinformatics*, vol. 32, no. 8, pp. 1256–1258, Apr. 2016.

[7] Jose Luís Estragués Muñoz, "HIGH PERFORMANCE FINITE VOLUME METHODS SOLVER FOR MULTISCALE CELL SIMULATIONS," .

[8] A. Olsson, G. Sandberg, and O. Dahlblom, "On latin hypercube sampling for structural reliability analysis," *Structural Safety*, vol. 25, no. 1, pp. 47–68, 2003.

X. Apéndice A

Tabla III: Parámetros de PhysiCell caracterizados por su función y por su impacto en el rendimiento computacional. Los niveles de impacto se asignan a partir del ranking de sensibilidad y del análisis de proliferación de agentes.

Clase	**Variable**	**Categoría funcional**	**Categoría por rendimiento**	**Impacto**
Dominio	x_min/y_min/z_min, x_max/y_max/z_max	Computacional	Escala espacial	Alto
	dx/dy/dz	Computacional	Escala espacial	Alto
	use_2D	Computacional	Escala espacial	Alto
Condiciones iniciales / parámetros de usuario	number_of_cells	Biológico	Número de agentes	Alto
	tumor_radius	Biológico	Número de agentes	Alto
Estructura del microentorno	substrate_count	Computacional	Escala espacial	Alto
Global	dt_diffusion	Computacional	Resolución temporal	Alto
	dt_mechanics	Computacional	Resolución temporal	Alto
	dt_phenotype	Computacional	Resolución temporal	Medio
	max_time	Computacional	Resolución temporal	Alto
	space_units, time_units	Metadatos	–	–
Paralelo	omp_num_threads	Computacional	Arquitectura hardware	Medio
Guardado	folder	Metadatos	–	–
	full_data.interval	Computacional	I/O	Alto
	full_data.enable	Computacional	I/O	Medio
	SVG.interval	Computacional	I/O	Alto
	SVG.enable	Computacional	I/O	Medio
	SVG.plot_substrate.enabled	Computacional	I/O	Medio
	SVG.plot_substrate.limits	Metadatos	–	–
	legacy_data.enable	Computacional	I/O	Medio
Opciones	legacy_random_points_on_sphere_in_divide	Computacional	Número de agentes	Bajo
	virtual_wall_at_domain_edge	Computacional	Escala espacial	Bajo
	disable_automated_spring_adhesions	Computacional	Número de agentes	Bajo
	random_seed	Biológico	Número de agentes	Bajo
Sustrato	name	Metadatos	–	–
	diffusion_coefficient	Biológico	Número de agentes	Medio
	decay_rate	Biológico	Número de agentes	Bajo
	initial_condition	Biológico	Número de agentes	Bajo
	dirichlet_boundary_condition	Biológico	Número de agentes	Medio
Secreción / captación celular	oxygen_uptake_rate	Biológico	Número de agentes	Alto
Ciclo celular	phase_durations	Biológico	Número de agentes	Alto
	phase_transition_rates	Biológico	Número de agentes	Alto
Muerte celular	phase_durations	Biológico	Número de agentes	Medio
	phase_transition_rates	Biológico	Número de agentes	Medio
Volumen celular	total	Biológico	Número de agentes	Alto
	fluid_fraction	Biológico	Número de agentes	Bajo
	nuclear	Biológico	Número de agentes	Medio
	fluid_change_rate	Biológico	Número de agentes	Bajo
	cytoplasmic_biomass_change_rate	Biológico	Número de agentes	Bajo
	nuclear_biomass_change_rate	Biológico	Número de agentes	Bajo
	calcified_fraction	Biológico	Número de agentes	Bajo
	calcification_rate	Biológico	Número de agentes	Bajo
	relative_rupture_volume	Biológico	Número de agentes	Bajo
Mecánica celular	cell_cell_adhesion_strength	Biológico	Número de agentes	Medio
	cell_cell_repulsion_strength	Biológico	Número de agentes	Bajo
	relative_maximum_adhesion_distance	Biológico	Número de agentes	Medio
Motilidad celular	speed	Biológico	Número de agentes	Bajo
	persistence_time	Biológico	Número de agentes	Bajo
	migration_bias	Biológico	Número de agentes	Bajo

Profiling de programas XDP en ARM mediante contadores hardware: diseño e implementación de ARMpect

Diego Valenzuela[1], Álex Gracia[1], José Luis Briz[1], Juan Segarra[1]

Resumen—
El profiling de programas XDP es una tarea crítica para identificar cuellos de botella en aplicaciones de red de alto rendimiento. Las herramientas convencionales como `perf` o `bpftool` introducen una sobrecarga excesiva que compromete la validez de las mediciones, y las soluciones ligeras existentes están limitadas a arquitecturas concretas y a la medición de un único evento hardware por ejecución. En este trabajo presentamos *ARMpect*, una herramienta de profiling ligero para programas XDP sobre arquitectura ARMv8/AArch64 que accede directamente a los contadores hardware de la PMU mediante un módulo de kernel propio compilado como código nativo ARM64, y que permite medir simultáneamente hasta seis eventos hardware en una única ejecución del programa. Evaluamos la herramienta sobre una Raspberry Pi 5 instrumentando un programa XDP que implementa un Count-Min Sketch para la monitorización de flujos de red, obteniendo un análisis de pipeline completo que revela indicios de cuello de botella y demuestra el valor práctico de la herramienta desarrollada.

Palabras clave— **eBPF, XDP, profiling, ARM, PMU, contadores hardware, procesamiento de paquetes, redes de alto rendimiento**

I. Introducción

El *extended Berkeley Packet Filter* (eBPF) es una tecnología del kernel Linux que permite ejecutar programas de usuario directamente en el espacio del kernel de forma segura, sin necesidad de modificar su código fuente ni cargar módulos [1], [2]. Su amplio abanico de aplicaciones abarca desde la observabilidad y la seguridad hasta, de forma destacada, el procesamiento de paquetes de red de alto rendimiento.

En este ámbito destaca *eXpress Data Path* (XDP) [3], el punto de procesamiento más cercano al hardware dentro de la pila de red del kernel. Al interceptar los paquetes directamente en el driver de la tarjeta de red, antes de cualquier procesamiento por parte del sistema operativo, XDP alcanza tasas competitivas con respecto a soluciones de *kernel bypass* como DPDK [4], manteniendo la integración con el resto del sistema [5]. Esta combinación ha propiciado su adopción en funciones de red críticas: balanceo de carga [6], cortafuegos [7], mitigación de ataques de denegación de servicios [8], aceleración de protocolos distribuidos [9] y construcción de servicios de red complejos [10], [11].

Alcanzar un alto rendimiento con XDP exige un diseño cuidadoso: pequeñas decisiones de implementación pueden traducirse en diferencias significativas de rendimiento [12], [13]. La identificación de cuellos de botella requiere herramientas de *profiling* de bajo coste computacional basadas en contadores hardware (PMC), cuyo propio uso debe minimizarse para no perturbar el sistema bajo análisis [14], [15]. Las herramientas habituales como `perf` [16] o `bpftool` [17] resultan excesivamente intrusivas en este contexto, reduciendo el rendimiento hasta un 75 % en aplicaciones sencillas [18].

Para abordar esta limitación, Paschali et al. propusieron InXpect [18], un sistema de *profiling* ligero que accede directamente a los PMC mediante funciones de kernel (*kfuncs*), reduciendo la sobrecarga hasta ocho veces respecto a `perf`. Sin embargo, InXpect presenta dos limitaciones: 1) fue concebido exclusivamente para procesadores de una arquitectura concreta, lo que impide su uso en otros sistemas como ARM, cuya presencia en servidores crece de forma sostenida gracias a plataformas como AWS Graviton o Ampere Altra; y 2) solo permite medir un evento hardware por ejecución del programa XDP, dificultando la correlación entre métricas como instrucciones, ciclos y fallos de cache [19], crítico a la hora de hacer profiling para casos atípicos.

En este trabajo presentamos una extensión de InXpect que supera ambas limitaciones: la portamos a ARM adaptando el acceso a la PMU de ARMv8, y extendemos su mecanismo de recogida de datos para leer múltiples eventos hardware en una única ejecución del programa XDP. Además, evaluamos la implementación sobre hardware ARM con un conjunto de programas XDP representativos.

El resto del artículo se organiza como sigue. La Sección II introduce los conceptos necesarios para entender la herramienta. La Sección III detalla el diseño de nuestra extensión. La Sección IV presenta la evaluación experimental y la Sección V recoge las conclusiones.

II. Contexto

A. eBPF y XDP

eBPF permite extender el comportamiento del kernel Linux cargando pequeños programas verificados en tiempo de ejecución. Antes de ejecutarse, cada programa se somete a un verificador estático que garantiza la seguridad de los accesos a memoria y la terminación del programa, descartando cualquier código que pudiera comprometer la estabilidad del sistema [1]. Tradicionalmente los programas se escri-

[1]Dept. of Computing and Systems Engineering - I3A, Universidad de Zaragoza, e-mail: {874085, alex.gracia, briz, jsegarra}@unizar.es

ben en C restringido, aunque en la actualidad existen entornos que permiten desarrollarlos también en Rust, Go o Python, entre otros [20]. En cualquier caso, el código fuente se compila a un *bytecode* independiente de la arquitectura; el proceso completo, ilustrado en la figura 1, consta de dos fases: una fase de compilación en la que `clang` genera el bytecode y lo empaqueta en un objeto ELF, y una fase de carga en la que el kernel verifica el programa y lo traduce a código nativo mediante un compilador *just-in-time*, con soporte tanto para x86-64 como para ARM64 y otras arquitecturas [2]. Los programas pueden acceder a datos persistentes entre ejecuciones a través de estructuras de datos compartidas denominadas *maps* [21], e invocar funciones del kernel a través de *helper functions* o *kfuncs*, estas últimas sin la restricción de una interfaz estable [18].

XDP (*eXpress Data Path*) es el punto de enganche eBPF situado en el driver de la tarjeta de red [3], antes de que el kernel asigne ninguna estructura de datos para el paquete ethernet recibido. El programa parte de un descriptor con punteros al buffer del paquete, en la tarjeta de red, y debe devolver un veredicto que determina su destino: `XDP_DROP` descarta el paquete de forma inmediata, `XDP_PASS` lo envía a la pila de red convencional para su procesamiento normal, `XDP_TX` lo retransmite por la misma interfaz de red por la que lo ha recibido, y `XDP_REDIRECT` lo redirige a otra interfaz o a un socket en espacio de usuario mediante AF_XDP. Al actuar en el nivel más bajo de la pila de red, XDP permite procesar decenas de millones de paquetes por segundo en hardware convencional [12].

En cuanto a los modos de operación, XDP ofrece tres variantes según el grado de integración con el hardware. El modo *native* es el más eficiente: el programa se ejecuta directamente en el driver de la NIC, sin que el kernel haya construido aún ninguna estructura de datos para el paquete. El modo *generic* permite usar XDP en cualquier tarjeta de red sin necesidad de modificar su driver, a costa de un rendimiento inferior al integrarse más tarde en la pila de red. Por último, el modo *offload* ejecuta el programa directamente en el procesador empotrado de la NIC, descargando completamente la CPU del host, aunque está disponible únicamente en determinadas *SmartNIC*s.

B. Contadores hardware de rendimiento en ARM

Los procesadores modernos incluyen una unidad de monitorización del rendimiento (PMU) con contadores hardware (PMC) capaces de registrar eventos de bajo nivel durante la ejecución: instrucciones completadas, ciclos de reloj, fallos de caché, fallos de predicción de saltos, entre otros [14], [15]. En la arquitectura ARMv8/AArch64, la PMU está estandarizada mediante la especificación PMUv3 [22]: cada contador se configura con el código del evento deseado en `PMEVTYPER`*n*`_EL0`, se habilita a través de `PMCNTENSET_EL0` y su valor se lee desde `PMEVCNTR`*n*`_EL0`. PMUv3 garantiza un mínimo de 3 contadores simultáneos, siendo habitual entre 6 y 8 en procesadores de alto rendimiento. La disponibilidad de varios contadores activos al mismo tiempo es precisamente la característica que explotamos en este trabajo para medir múltiples eventos en una única ejecución del programa XDP.

III. Diseño e Implementación

La herramienta sigue la misma filosofía que InXpect [18]: el programador delimita las regiones de código a analizar mediante las macros `BPF_KPERF_START_BURST` y `BPF_KPERF_END_BURST` directamente en el fuente del programa XDP, y una herramienta de usuario recoge y presenta las métricas en tiempo real. La figura 2 muestra la arquitectura completa del sistema, donde se aprecia la separación entre el bytecode eBPF portable, que contiene las macros y el *map* de resultados y el módulo de kernel compilado como código nativo ARM64, responsable del acceso directo a los registros PMU. La capa de acceso al hardware ha sido desarrollada desde cero para la arquitectura ARMv8/AArch64 mediante un módulo de kernel propio, y se ha añadido la capacidad de leer múltiples contadores PMU de forma simultánea en cada ejecución del programa, funcionalidad no presente en trabajos previos.

A. Acceso a los contadores PMU desde el kernel

Dado que los programas eBPF/XDP no pueden ejecutar instrucciones privilegiadas directamente, el acceso a la PMU se implementa mediante *kfuncs*: funciones del kernel compiladas como código nativo ARM64 y expuestas como llamables desde eBPF. Se han desarrollado dos *kfuncs*, `bpf_kperf__pmc_init` y `bpf_kperf__stop_and_read`, que delimitan cada ventana de medición: la primera detiene los contadores, los inicializa a cero y los reactiva; la segunda los detiene, lee los valores acumulados y los devuelve al programa eBPF.

La selección y configuración de los eventos hardware se realiza desde la herramienta de usuario mediante una llamada `ioctl` al módulo de kernel, que escribe los códigos de evento seleccionados directamente en los registros `PMEVTYPER` de la PMU. Esta operación ocurre una única vez antes de comenzar la medición y no introduce ninguna sobrecarga en la ejecución del programa XDP. El acceso a los contadores dentro de las *kfuncs* invocadas por cada paquete se realiza mediante lectura y escritura directa de los registros PMU a través de las instrucciones `mrs`/`msr` en ensamblador ARM64, sin ninguna capa de abstracción intermedia.

Esta decisión de diseño está motivada por dos propiedades fundamentales de los programas XDP. En primer lugar, los programas XDP son no expulsables (*non-preemptible*): una vez que comienza su ejecución, el programa se ejecuta hasta completarse sin que el planificador del kernel pueda interrumpirlo ni migrarlo a otro núcleo [18]. Esta garantía elimina la necesidad de la infraestructura de gestión de contexto que requiere `perf` para filtrar eventos de otras

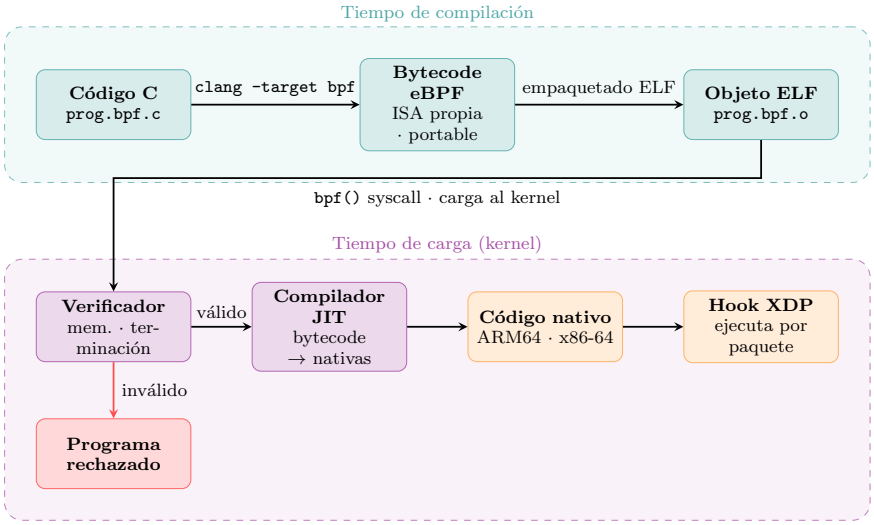


Fig. 1: Flujo de compilación eBPF/XDP: fase offline mediante Clang y fase de carga en el kernel con verificación y compilación *just-in-time*.

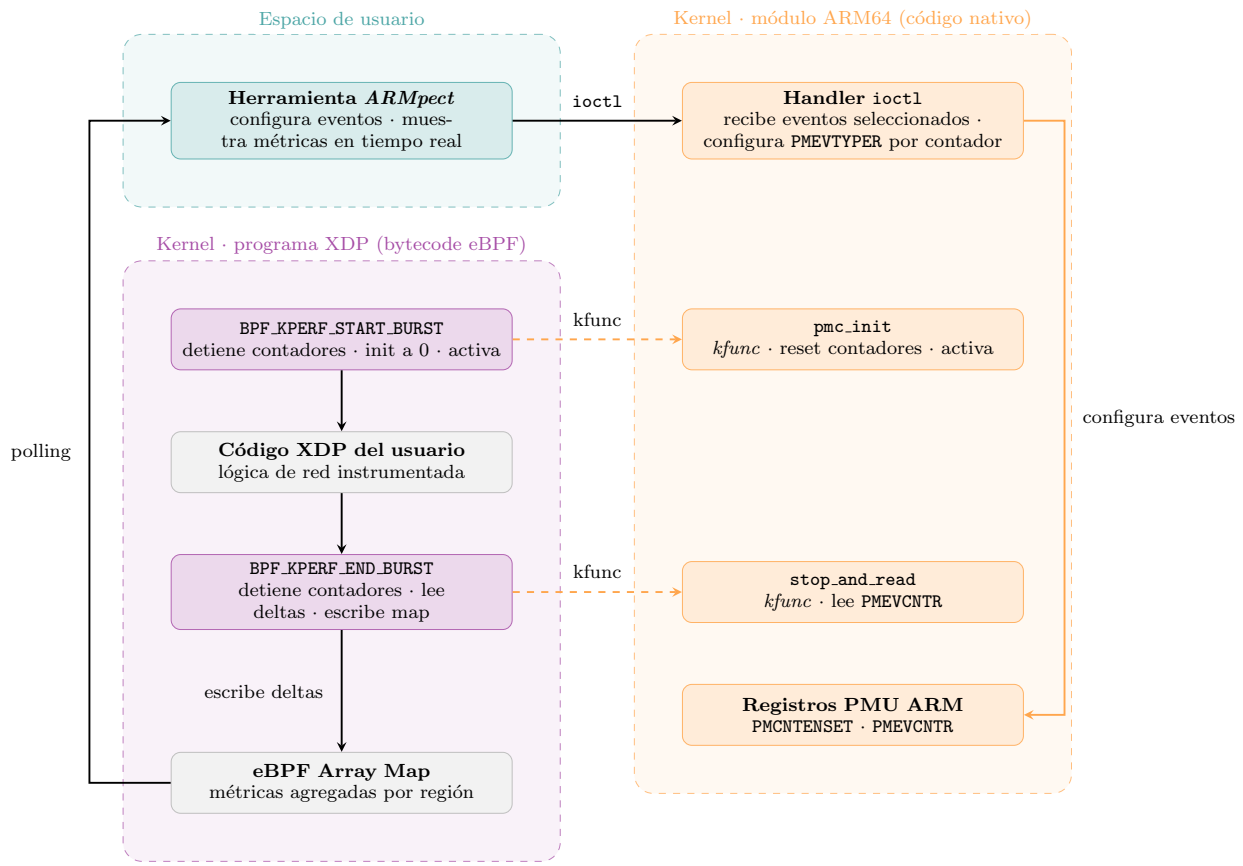


Fig. 2: Arquitectura de *ARMpect*: la configuración de los registros PMU se realiza desde la herramienta de usuario mediante `ioctl`; las *kfuncs* únicamente resetean y leen los contadores en cada ventana de medición.

tareas, permitiendo acceder a los registros PMU sin riesgo de contaminación entre contextos. En segundo lugar, al ejecutar un número reducido de instrucciones por paquete ethernet, del orden de pocas decenas a pocos cientos, cualquier sobrecarga de medición representa una fracción significativa del trabajo útil. Al usar un acceso directo mediante ensamblador se reduce el coste de cada ventana de medición a las instrucciones estrictamente necesarias para leer y escribir los registros `PMCNTENSET_EL0` y `PMXEVCNTR_EL0`, sin llamadas a funciones del kernel ni gestión de estructuras de datos intermedias.

IV. Evaluación

A. Plataforma experimental

Los experimentos se han realizado sobre una Raspberry Pi 5 equipada con un procesador ARM Cortex-A76 de cuatro núcleos. El Cortex-A76 implementa la especificación PMUv3 con 6 contadores de eventos programables simultáneos, según indica el campo N del registro `PMCR_EL0`. El tráfico de red se genera con `iperf3`, produciendo flujos TCP.

Tabla I: Métricas de pipeline por paquete para `exp_cms` (Cortex-A76, Raspberry Pi 5).

Evento	Media/paquete	% ciclos
Instrucciones retiradas	198.00	—
Ciclos totales	890.50	100 %
Stall backend	414.67	46.6 %
Stall frontend	403.67	45.3 %

B. Caso de uso: Count-Min Sketch sobre XDP

Para demostrar la utilidad de la herramienta en un escenario real se ha instrumentado `exp_cms`, un programa XDP que implementa un Count-Min Sketch (CMS) para la monitorización de flujos de red. El CMS es una estructura de datos probabilística para la estimación de frecuencias [23]. Almacena una matriz de d filas y w columnas de contadores, donde las filas se asocian a funciones hash y las columas a resultados de dichas funciones. Para cada paquete entrante, calcula d funciones hash independientes sobre el identificador de flujo, incrementando el contador correspondiente en cada fila. La estimación de la frecuencia de un flujo se obtiene como el mínimo de los d contadores asociados, lo que proporciona una cota superior con error acotado probabilísticamente. Su reducido coste computacional y su ausencia de falsos negativos lo hacen especialmente adecuado para el procesamiento en la ruta de datos del kernel.

En la implementación evaluada, el CMS cuenta con cuatro filas ($d = 4$) de contadores indexadas mediante el hash de cada flujo, con offsets de 1 MB entre filas, lo que resulta en un *working set* total del orden de los 3 MB. El tráfico de entrada consiste en flujos UDP generados con `iperf3` desde una máquina externa conectada directamente a la Raspberry Pi 5. Para cada ejecución del programa XDP se recogen los valores brutos de los contadores PMU; los resultados que se presentan corresponden a la media aritmética calculada sobre el conjunto de muestras válidas, descartando aquellas en las que el contador `EXC_TAKEN` registra alguna excepción, lo que indica una interrupción durante la ventana de medición. Se han medido simultáneamente seis eventos hardware en una única ejecución del programa, sin necesidad de rondas de medición independientes. La tabla I recoge los resultados.

A partir de estos datos se derivan las siguientes métricas de análisis de pipeline siguiendo la metodología Top-Down [22], [24]:

$$\text{IPC} = \frac{198{,}00}{890{,}50} = 0{,}22 \tag{1}$$

$$\text{Backend Bound} = \frac{414{,}67}{890{,}50} = 46{,}6\,\% \tag{2}$$

$$\text{Frontend Bound} = \frac{403{,}67}{890{,}50} = 45{,}3\,\% \tag{3}$$

La categoría *Backend Bound* se asocia a ciclos en los que no se emite ninguna operación debido a un bloqueo (*stall*) en el *pipeline*, mientras que *Frontend Bound* corresponde a los ciclos en los que no se emiten operaciones debido a que la cola de instrucciones se encuentra vacía.

El IPC de 0.22 es notablemente inferior a la capacidad teórica del Cortex-A76 de 4 instrucciones por ciclo [22], lo que sugiere que el programa no está limitado por la capacidad de cómputo del procesador sino por algún otro recurso. El Backend Bound del 46.6 % indica que el backend pasa una fracción significativa de los ciclos esperando algún recurso, apuntando a un posible cuello de botella fuera de la unidad de ejecución [24].

El Frontend Bound del 45.3 % merece una interpretación cuidadosa. Según la metodología Top-Down, cuando frontend y backend están parados simultáneamente el ciclo se clasifica como Backend Bound, ya que resolver el stall del frontend no mejoraría el rendimiento mientras el backend permanezca bloqueado [24]. Los valores similares en ambas métricas son consistentes con un escenario en el que el backend lleva la iniciativa del bloqueo, propagando el efecto hacia el frontend, aunque confirmar esta hipótesis requeriría un análisis más detallado a nivel de microarquitectura [15].

En conjunto, el programa pasa más del 90 % de los ciclos con el pipeline parado, sumando Backend Bound y Frontend Bound, con apenas un 10 % de los ciclos dedicados a retirar instrucciones de forma efectiva, sin tener en cuenta los ciclos perdidos por mala especulación. Esto es consistente con la presencia de un cuello de botella, que podría ser detectado realizando un análisis en mayor profundidad con esta herramienta.

V. Conclusiones y Trabajo Futuro

En este trabajo hemos presentado *ARMpect*, una herramienta de *profiling* ligero para programas XDP sobre arquitectura ARM inspirada en el diseño de InXpect [18]. Nuestra herramienta supera sus dos principales limitaciones: la dependencia de una arquitectura concreta y la restricción de medir un único evento hardware por ejecución del programa.

La evaluación realizada demuestra que los programas XDP, pese a su aparente simplicidad, presentan comportamientos microarquitecturales complejos que requieren herramientas de análisis precisas para ser caracterizados correctamente. El diagnóstico obtenido en el análisis del programa *Count-Min Sketch* pone de manifiesto la necesidad de optimización de este tipo de programas, algo que con herramientas convencionales como `perf` o `bpftool` resultaría inviable debido a la sobrecarga que introducen [18].

A medida que XDP se consolida como plataforma para funciones de red de alto rendimiento [3], [2], la disponibilidad de herramientas de *profiling* precisas y de bajo coste computacional se vuelve imprescindible para guiar las decisiones de diseño. La arquitectura ARM, con su creciente presencia en servidores y plataformas de computación en el borde, representa un escenario donde esta necesidad es especialmente acuciante, dado que las herramientas existentes están orientadas mayoritariamente a x86.

Como trabajo futuro, la principal línea de interés es la integración de *ARMpect* con la plataforma InXpect [18] para construir un framework de *profiling* unificado que opere de forma transparente sobre distintas arquitecturas, abstrayendo las diferencias entre los mecanismos de acceso a la PMU de cada plataforma y permitiendo así que los desarrolladores de programas XDP dispongan de una herramienta independiente del hardware subyacente.

Agradecimientos

Este trabajo ha sido financiado por MICIU/AEI/10.13039/501100011033 (proyecto PID2022-136454NB-C22), y por el Departamento de Ciencia, Universidad y Sociedad del Conocimiento del Gobierno de Aragon (ayuda a grupo de investigación de referencia T58_23R).

Referencias

[1] Farbod Shahinfar, Sebastiano Miano, Aurojit Panda, and Gianni Antichi, "Demystifying performance of eBPF network applications," *Proceedings of the ACM on Networking*, vol. 3, no. CoNEXT3, pp. 1–21, Sept. 2025.

[2] Marcos A. M. Vieira, Matheus S. Castanho, Racyus D. G. Pacífico, Elerson R. S. Santos, Eduardo P. M. Câmara Júnior, and Luiz F. M. Vieira, "Fast packet processing with eBPF and XDP: Concepts, code, challenges, and applications," *ACM Computing Surveys*, vol. 53, no. 1, pp. 1–36, 2020.

[3] Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller, "The eXpress data path: Fast programmable packet processing in the operating system kernel," in *Proceedings of the 14th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '18)*, Heraklion, Greece, 2018, pp. 54–66, ACM.

[4] DPDK Project, "DPDK: Data plane development kit," `https://www.dpdk.org/about/`, 2025.

[5] Federico Parola, Roberto Procopio, Roberto Querio, and Fulvio Risso, "Comparing user space and in-kernel packet processing for edge data centers," 2023, vol. 53, pp. 14–29, ACM.

[6] Nikita Shirokov and Ranjeeth Dasineni, "Katran, a scalable network load balancer," `https://engineering.fb.com/2018/05/22/open-source/open-sourcing-katran-a-scalable-network-load-balancer/`, 2018.

[7] Sebastiano Miano, Matteo Bertrone, Fulvio Risso, Mauricio Vásquez Bernal, Yunsong Lu, and Jianwen Pi, "Securing linux with a faster and scalable iptables," *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 3, 2019.

[8] Gilberto Bertin, "XDP in practice: Integrating XDP into our DDoS mitigation pipeline," in *Technical Conference on Linux Networking (Netdev)*, 2017.

[9] Yang Zhou, Zezhou Wang, Sowmya Dharanipragada, and Minlan Yu, "Electrode: Accelerating distributed protocols with eBPF," in *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI '23)*. 2023, pp. 1391–1407, USENIX Association.

[10] Sebastiano Miano, Matteo Bertrone, Fulvio Risso, Massimo Tumolo, and Mauricio Vásquez Bernal, "Creating complex network services with eBPF: Experience and lessons learned," in *Proceedings of the IEEE International Conference on High Performance Switching and Routing (HPSR)*. 2018, pp. 1–8, IEEE.

[11] Cilium Community, "Cilium: eBPF-based networking, security, and observability," `https://cilium.io`, 2025.

[12] Oliver Hohlfeld, Johannes Krude, Jens Helge Reelfs, Jan Rüth, and Klaus Wehrle, "Demystifying the performance of XDP BPF," in *Proceedings of the IEEE Conference on Network Softwarization (NetSoft)*. 2019, pp. 1–8, IEEE.

[13] Marcelo Abranches, Oliver Michel, Eric Keller, and Stefan Schmid, "Efficient network monitoring applications in the kernel with eBPF and XDP," in *Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 2021, pp. 28–34, IEEE.

[14] Georgios Bitzes and Andrzej Nowak, "The overhead of profiling using PMU hardware counters," Tech. Rep., CERN openlab, 2014.

[15] Tanvir Ahmed Khan, Ian Neal, Gilles Pokam, Barzan Mozafari, and Baris Kasikci, "DMon: Efficient detection and correction of data locality problems using selective profiling," in *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI '21)*. 2021, pp. 163–181, USENIX Association.

[16] Arnaldo Carvalho De Melo, "The new linux `perf` tools," `https://lwn.net/Articles/337493/`, 2010.

[17] "`bpftool`: Tool for inspection and simple manipulation of eBPF programs and maps," `https://bpftool.dev/`, 2023.

[18] Vladimiro Paschali, Andrea Monterubbiano, Francesco Fazzari, Michael Swift, and Salvatore Pontarelli, "InXpect: Lightweight XDP profiling," in *Proceedings of the 3rd Workshop on eBPF and Kernel Extensions (eBPF '25)*, Coimbra, Portugal, Sept. 2025, pp. 48–54, ACM.

[19] Sebastiano Miano, Alireza Sanaee, Fulvio Risso, Gábor Rétvári, and Gianni Antichi, "Domain specific run time optimization for software data planes," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22)*. 2022, pp. 1148–1164, ACM.

[20] Carlos Machado, Bruno Gião, Sebastião Amaro, Miguel Matos, João Paulo, and Tânia Esteves, "No two snowflakes are alike: Studying eBPF libraries' performance, fidelity and resource usage," in *Proceedings of the 3rd Workshop on eBPF and Kernel Extensions (eBPF '25)*, Coimbra, Portugal, Sept. 2025, pp. 31–37, ACM.

[21] Chang Liu, Byungchul Tak, and Long Wang, "Understanding performance of eBPF maps," in *Proceedings of the ACM SIGCOMM 2024 Workshop on eBPF and Kernel Extensions (eBPF '24)*, Sydney, Australia, Aug. 2024, pp. 9–15, ACM.

[22] ARM Limited, *ARM Architecture Reference Manual: ARMv8, for ARMv8-A Architecture Profile*, 2024, DDI0487.

[23] Graham Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[24] Intel Corporation, "Top-down microarchitecture analysis method," `https://www.intel.com/content/www/us/en/docs/vtune-profiler/cookbook/2023-0/top-down-microarchitecture-analysis-method.html`, 2023.

[25] Yoann Ghigoff, Julien Sopena, Kahina Lazri, Antoine Blin, and Gilles Muller, "BMC: Accelerating memcached using safe in-kernel caching and pre-stack processing," in *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI '21)*. 2021, pp. 487–501, USENIX Association.

[26] Farbod Shahinfar, Aurojit Panda, and Gianni Antichi, "Software prefetching for eBPF programs," in *Proceedings of the ACM SIGCOMM 2025 Posters and Demos*, Coimbra, Portugal, Sept. 2025, pp. 61–63, ACM.

[27] Milo Craun and Dan Williams, "Pairwise BPF programs should be optimized together," in *Proceedings of the 3rd Workshop on eBPF and Kernel Extensions (eBPF '25)*, Coimbra, Portugal, Sept. 2025, pp. 22–27, ACM.

[28] Yusheng Zheng, Panayiotis Gavriil, and Marios Kogias, "uXDP: Frictionless XDP deployments in userspace," in *Proceedings of the 3rd Workshop on eBPF and Kernel Extensions (eBPF '25)*, Coimbra, Portugal, Sept. 2025, pp. 8–14, ACM.

[29] Sebastiano Miano, Xiaoqi Chen, Ran Ben Basat, and Gianni Antichi, "Fast in-kernel traffic sketching in eBPF," 2023, vol. 53, pp. 3–13, ACM.

Computación cuántica (2/2)

Simulación de computación cuántica en RISC-V: vectorización y paralelismo multihilo

Rebeca Rasco, Sonia González-Navarro, Eladio Gutiérrez, Fco. Javier Hormigo y Óscar Plata[1]

Resumen— **La simulación de algoritmos cuánticos presenta desafíos de escalabilidad cuyo impacto se analiza en este trabajo, sobre la arquitectura RISC-V SpacemiT K1, mediante el uso de la extensión vectorial RVV y OpenMP. Al evaluar variantes secuenciales, vectorizadas ($SIMD$), multinúcleo ($MIMD$) e híbridas, se determinó que la configuración híbrida es la más eficiente, alcanzando un *speedup* de hasta $6{,}1\times$. Los resultados demuestran que la vectorización beneficia a sistemas pequeños, mientras que el paralelismo de hilos resulta más eficaz en sistemas intermedios. Sin embargo, en sistemas grandes (hasta 30 qubits, 2^{30} amplitudes), el rendimiento queda limitado por el ancho de banda de memoria.**

Este trabajo también analiza cómo la posición del qubit objetivo y el valor de *stride* afectan la distribución de carga, obteniendose que un *stride* elevado puede reducir el paralelismo disponible al disminuir el número de bloques independientes, limitando el trabajo que puede repartirse entre los hilos. Para mitigar esto, se ha empleado la cláusula `collapse` de OpenMP, la cual estabiliza el rendimiento optimizando el reparto del trabajo. En conclusión, el éxito en arquitecturas RISC-V no depende solo del tamaño del sistema, sino de una organización del trabajo que combine eficientemente la vectorización SIMD y el paralelismo MIMD.

Palabras clave— **Computación cuántica, multiplicación *qubit-wise*, RISC-V, RVV vector extension, OpenMP, paralelismo**

I. Introducción

La simulación de algoritmos cuánticos en hardware clásico presenta un desafío significativo debido al crecimiento exponencial del vector de estado con el número de qubits (n). Para un sistema de n qubits, el estado cuántico se describe mediante un vector de estado en la base computacional [1], que puede expresarse como

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \tag{1}$$

donde cada $\alpha_i \in \mathbb{C}$ es la amplitud compleja que describe la superposición cuántica del sistema y $|i\rangle$ representa un estado base del sistema. El vector de estado se compone de 2^n amplitudes complejas.

Los algoritmos cuánticos consisten principalmente en la aplicación sucesiva de puertas cuánticas, que se modelan como operaciones matriciales sobre el vector de estado y representan la evolución del sistema dentro del espacio de Hilbert. En un sistema de n qubits, el espacio de Hilbert tiene dimensión 2^n, por lo que la transformación global del sistema tendría dimensión $2^n \times 2^n$. Este crecimiento exponencial hace que la construcción explícita de dicha matriz resulte computacionalmente inviable incluso para sistemas de tamaño moderado.

Para evitar este coste computacional, los simuladores cuánticos emplean habitualmente una estrategia que, en este trabajo, denominamos *qubit-wise*. En este enfoque, las puertas cuánticas de un solo qubit, representadas por matrices unitarias de dimensión 2×2, se aplican únicamente sobre los pares de amplitudes afectados por el qubit objetivo, evitando la construcción explícita de la matriz global (cuya complejidad computacional sería $O(2^{2n})$) [2], [3].

Para una puerta unitaria $U \in \mathbb{C}^{2\times 2}$, la transformación aplicada sobre cada par de amplitudes puede expresarse como

$$\begin{pmatrix} \alpha'_{i_0} \\ \alpha'_{i_1} \end{pmatrix} = U \begin{pmatrix} \alpha_{i_0} \\ \alpha_{i_1} \end{pmatrix}, \tag{2}$$

donde i_0 e i_1 representan los índices de dos estados base que difieren únicamente en el valor del qubit objetivo. Este enfoque permite que la aplicación de una puerta unitaria de un solo qubit sobre el vector de estado tenga complejidad lineal $O(2^n)$ respecto al tamaño del vector de estado.

A pesar de esta mejora algorítmica, la simulación cuántica continúa siendo altamente exigente desde el punto de vista computacional. El tamaño del vector de estado crece exponencialmente con el número de qubits, lo que implica que incluso simulaciones de tamaño moderado requieren grandes cantidades de memoria y un elevado ancho de banda para el acceso a los datos. En consecuencia, el rendimiento de los simuladores basados en vector de estado está fuertemente condicionado por las características del hardware, especialmente por la jerarquía de memoria y las capacidades de paralelismo de la arquitectura [4].

En este contexto, aprovechar técnicas de paralelización y vectorización resulta fundamental para mejorar el rendimiento de este tipo de aplicaciones. Las arquitecturas modernas ofrecen diferentes niveles de paralelismo [5], incluyendo paralelismo a nivel de hilo y paralelismo a nivel de instrucción, que pueden explotarse para acelerar el procesamiento del vector de estado y reducir el tiempo de ejecución de la simulación [6], [7], [8].

En este trabajo se desarrolla un simulador cuántico completamente funcional en lenguaje C con el objetivo de explorar estrategias de paralelización para la operación de multiplicación *qubit-wise*, ya que esta constituye el núcleo computacional dominante en los simuladores cuánticos. Para ello, se empleó la extensión vectorial RISC-V Vector (RVV) [9] junto con

[1]Dpto. de Arquitectura de Computadores, Universidad de Málaga, e-mail: {rebecarasco, sgn, eladio, fjhormigo, oplata}@uma.es.

paralelismo multihilo. Aunque la implementación se inspira en conceptos presentes en simuladores existentes [10], el simulador fue desarrollado desde cero para permitir la experimentación con intrínsecos vectoriales a nivel de instrucción (SIMD) y paralelismo a nivel de hilo mediante OpenMP (MIMD).

II. Estrategia *qubit-wise* sobre un qubit

La aplicación de la estrategia *qubit-wise* a un solo qubit consiste en recorrer el vector de estado siguiendo un patrón regular que permite identificar los pares de amplitudes afectados por la puerta cuántica. Estos pares corresponden a estados base cuyos índices difieren únicamente en el valor del qubit objetivo.

El patrón de acceso al vector de estado depende de la posición del qubit objetivo k y determina la forma en que se agrupan y procesan las amplitudes del vector. A partir de k, se definen dos parámetros que estructuran dicho acceso: el *stride* y el tamaño de bloque.

El *stride* define la separación entre las amplitudes de cada par afectado por la puerta y viene dado por

$$\text{stride} = 2^k \tag{3}$$

El tamaño de bloque (blockSize) define la periodicidad con la que este patrón de acceso se repite en el vector de estado:

$$\text{blockSize} = 2 \cdot \text{stride} = 2^{k+1} \tag{4}$$

A partir de estos parámetros, el número de bloques (*numBlocks*) a procesar en un vector de estado de tamaño 2^n viene dado por:

$$\text{numBlocks} = \frac{2^n}{2^{k+1}} \tag{5}$$

Por tanto, la aplicación de una puerta cuántica sobre el qubit k consiste en recorrer todos los bloques del vector de estado y, dentro de cada uno, aplicar la transformación unitaria sobre las parejas de amplitudes separadas por el valor de *stride* [10]. Este patrón de acceso corresponde a la implementación habitual de la aplicación de puertas de un solo qubit en simuladores cuánticos basados en vector de estado.

Para ilustrar este patrón de acceso, la Figura 1 muestra un ejemplo para $n = 3$ qubits. El vector de estado contiene $2^3 = 8$ amplitudes $a_0, \ldots, a_7$. Si la puerta se aplica sobre el qubit objetivo $k = 1$, entonces $stride = 2$, $blockSize = 4$ y $numBlocks = 2$, por lo que las parejas procesadas son (a_0, a_2), (a_1, a_3), (a_4, a_6) y (a_5, a_7). En la implementación, cada amplitud compleja se representa mediante dos arrays separados para las partes real e imaginaria.

El Algoritmo 1 resume el esquema general de esta operación.

En cada bloque se procesan *stride* parejas de amplitudes independientes. Dado que $stride = 2^k$, el número de parejas por bloque aumenta con k, mientras que el número total de bloques disminuye proporcionalmente. Esta relación define la organización del cómputo en la estrategia *qubit-wise*.

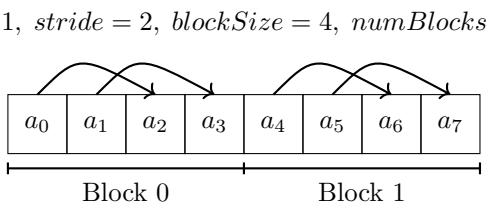


Fig. 1: Ejemplo del patrón de acceso para $n = 3$ y $k = 1$. Las amplitudes se procesan en parejas separadas por $stride = 2$ dentro de bloques de tamaño $blockSize = 4$.

Algorithm 1 Estrategia *qubit-wise* de una puerta de un qubit

Require: Vector de estado ψ de tamaño 2^n, matriz unitaria $U \in \mathbb{C}^{2\times 2}$, qubit objetivo k

$stride \leftarrow 2^k$
$blockSize \leftarrow 2^{k+1}$
$numBlocks \leftarrow 2^n / blockSize$
for $b = 0$ to $numBlocks - 1$ **do**
  $base \leftarrow b \cdot blockSize$
  for $offset = 0$ to $stride - 1$ **do**
    $i_0 \leftarrow base + offset$
    $i_1 \leftarrow i_0 + stride$
    $a_0 \leftarrow \psi[i_0]$
    $a_1 \leftarrow \psi[i_1]$
    $\psi[i_0] \leftarrow U_{00}a_0 + U_{01}a_1$
    $\psi[i_1] \leftarrow U_{10}a_0 + U_{11}a_1$
  end for
end for

III. Metodología

Para evaluar el impacto de las optimizaciones arquitectónicas, se implementaron cuatro variantes de la operación *qubit-wise*, cada una con distintos niveles de paralelización:

- **Secuencial (baseline):** versión inicial que realiza la operación de forma secuencial. Sirve como referencia para comparar las mejoras de rendimiento.
- **Paralela (OpenMP):** introducción de paralelismo multihilo mediante directivas #pragma omp parallel for, distribuyendo los bloques del vector de estado entre los hilos para repartir la carga de trabajo.
- **Vectorial RVV (RISC-V Vector):** optimización a nivel de instrucción que utiliza los registros vectoriales del procesador. Esta versión procesa múltiples amplitudes simultáneamente, aprovechando la extensión vectorial RVV mediante intrínsecos SIMD.
- **Versión híbrida (OpenMP + RVV):** combinación de paralelismo a nivel de hilo (MIMD) y a nivel de instrucción (SIMD). Cada hilo ejecuta operaciones vectoriales sobre su bloque de datos, buscando maximizar el rendimiento global del hardware.

A. Pruebas experimentales

Para evaluar el impacto de las optimizaciones, se diseñaron dos experimentos basados en la aplica-

ción de puertas de Hadamard mediante la operación *qubit-wise*, con el objetivo de aislar el coste computacional dominante de esta operación.

En el primer experimento, la puerta se aplica secuencialmente a cada qubit incrementando el tamaño del sistema hasta $n = 30$ (2^{30} amplitudes), lo que permite analizar la escalabilidad de las distintas implementaciones. En el segundo experimento, se mantiene fijo el tamaño del sistema y varía el qubit objetivo con el fin de estudiar cómo el valor de *stride* influye en el paralelismo disponible y en el *speedup*.

Cada prueba mide exclusivamente el tiempo de cómputo de la operación, excluyendo la inicialización de memoria y otras operaciones auxiliares, y los resultados corresponden al tiempo promedio obtenido tras varias iteraciones. Este estudio se limita a puertas unitarias de un solo qubit; la optimización de puertas controladas mediante este enfoque se plantea como trabajo futuro.

B. Plataforma experimental

Los experimentos se ejecutaron sobre la plataforma *Banana Pi BPI-F3 SBC* [11], equipada con el procesador *SpacemiT K1*, una arquitectura RISC-V de 64 bits con una longitud de vector $VLEN = 256$ bits y soporte para la extensión vectorial RVV v1.0, donde LMUL (Vector Length Multiplier) puede tomar valores $LMUL = 1 - 8$ [9]. Esta arquitectura permite una ejecución agnóstica a la longitud del vector (*Vector Length Agnostic*). El entorno de software incluyó el sistema operativo Linux y el compilador GCC con soporte nativo para intrínsecos vectoriales.

IV. Resultados experimentales

Los resultados se presentan siguiendo los dos experimentos introducidos en la sección de metodología: analisis de la escalabilidad con el número de qubits e influencia del qubit objetivo.

A. Escalabilidad con el número de qubits

La Figura 2 presenta el *speedup* relativo al *baseline* secuencial, seleccionando los mejores tiempos obtenidos para cada versión optimizada. Para la versión paralela con OpenMP, se realizaron pruebas con 1, 2, 4 y 8 hilos, siendo esta última la implementación más rápida. Para la versión vectorial RVV, se evaluaron valores de $LMUL = 1 - 8$, resultando $LMUL = 1$ como la opción óptima, ya que libera una mayor cantidad de registros para las operaciones intermedias y reduce el *overhead* de gestión de instrucciones. La versión híbrida OMP + RVV combina ambos parámetros (8 hilos y $LMUL = 1$), operando así bajo su configuración más eficiente.

A.1 Rendimiento según el número de qubits

El rendimiento del simulador varía drásticamente según el tamaño del sistema, identificándose 3 régimenes claros:

1. **Sistemas pequeños ($n \leq 8$ qubits):** La versión RVV es la más eficiente. En este rango, el *overhead* derivado de la creación y sincronización de hilos en OpenMP supera el tiempo de cómputo de la operación cuántica, haciendo que las versiones basadas en hilos (OpenMP o híbrida) presenten tiempos de ejecución superiores al *baseline* secuencial. La vectorización a nivel de instrucción es la única optimización que aporta mejoras sin penalización de latencia.
2. **Sistemas medios ($8 < n \leq 16$ qubits):** A partir de los 9 qubits, el paralelismo por hilos comienza a compensar su coste inicial. El *speedup* máximo se alcanza con 16 qubits, punto en el que los datos aún residen eficientemente en las cachés y la combinación OMP + RVV demuestra una sinergia ideal entre el paralelismo MIMD y SIMD.
3. **Sistemas grandes ($n \geq 20$ qubits):** La simulación pasa a estar limitada por la capacidad de la jerarquía de memoria, ya que el vector de estado deja de caber en caché, lo que provoca que el rendimiento quede dominado por el ancho de banda de la memoria RAM y el *speedup* se estabilice.

A.2 Análisis del comportamiento en Memoria

En una simulación de 30 qubits —máximo que soporta el simulador— el vector de estado requiere el manejo de 2^{30} amplitudes complejas. En la implementación utilizada, cada amplitud se representa en precisión simple, con 4 bytes para la parte real y 4 para la imaginaria, por lo que el requisito de memoria alcanza aproximadamente 8 GB. Este volumen excede la capacidad de las caches del SpacemiT K1 (*L2 compartida* de 512 KB por *cluster*), obligando a un flujo masivo de datos desde la RAM. Para mitigar este cuello de botella, se emplea una estructura de arreglos (*Structure of Arrays* o SoA) que separa las partes reales e imaginarias, permitiendo accesos contiguos y optimizando la localidad de memoria.

Para 30 qubits, los tiempos de ejecución son: secuencial $\sim$ 400 s, OpenMP $\sim$ 130 s, RVV $\sim$ 200 s e híbrida $\sim$ 119 s. Comparando la versión secuencial con la híbrida, se obtiene un *speedup* de aproximadamente 3,38×, lo que confirma que, incluso en condiciones limitadas por la capacidad de la caché, la combinación de paralelismo por hilos y vectorización mantiene una ventaja sustancial. La Tabla I resume el consumo de memoria para sistemas grandes y su relación con la jerarquía de memoria.

Tabla I: Relación entre el número de qubits y la jerarquía de memoria

Q (n)	**Amp.** (2^n)	**Mem.**	**Cache/RAM**
16	65,536	512 KB	Alojable en L2
20	1,048,576	8 MB	Acceso mixto L2/RAM
24	16,777,216	128 MB	Excede capacidad L2
30	1,073,741,824	8 GB	Excede capacidad L2

B. Impacto del qubit objetivo en el paralelismo

La Figura 3 muestra el modelo teórico del paralelismo en función del qubit objetivo para un sistema

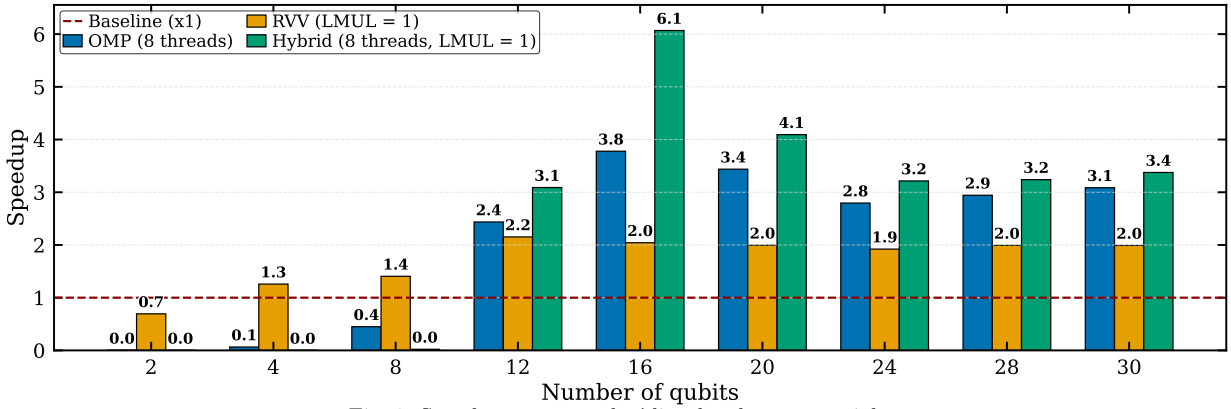


Fig. 2: Speedup respecto al código *baseline* secuencial.

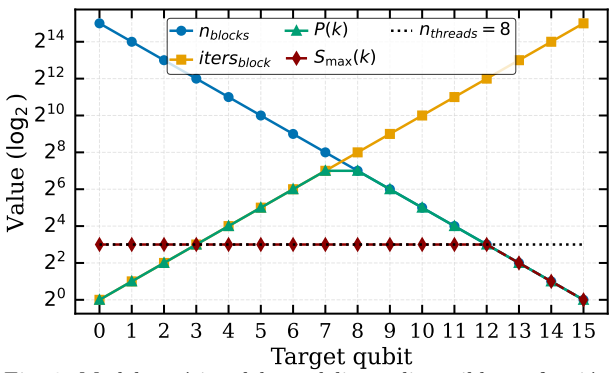


Fig. 3: Modelo teórico del paralelismo disponible en función del qubit objetivo y del speedup máximo.

de $n = 16$ qubits. Se selecciona este tamaño debido a que, como se observó en el experimento anterior, es donde se alcanza el máximo *speedup* y gran parte del conjunto de trabajo cabe en caché, permitiendo evaluar el comportamiento del algoritmo con menor influencia de efectos asociados a la jerarquía de memoria. En la figura se considera un ejemplo con $n_{threads} = 8$ hilos.

La figura muestra la evolución del número de bloques y del número de iteraciones por bloque en función del qubit objetivo k. Ambos parámetros presentan un comportamiento inverso: al aumentar k, el número de bloques decrece exponencialmente, mientras que las iteraciones por bloque crecen en la misma proporción. En esta implementación, el número de iteraciones del bucle interno por bloque coincide con el valor del *stride*.

Como consecuencia, el trabajo total del algoritmo puede expresarse como el producto entre el número de bloques y el número de iteraciones por bloque, es decir, $n_{blocks} \cdot iters_{block}$, que se mantiene constante e igual a 2^{n-1}, correspondiente al número total de parejas de amplitudes procesadas. Así, al variar k, no cambia la cantidad total de trabajo, sino únicamente su distribución.

Este efecto se ilustra mediante la curva $P(k) = \text{mín}(n_{blocks}, iters_{block})$, utilizada aquí como una medida heurística del equilibrio entre el número de tareas disponibles y la carga de trabajo por tarea. Esta magnitud no representa una cota exacta del paralelismo explotable, pero permite visualizar la región en la que ambos factores resultan más equilibrados.

Por otro lado, bajo la estrategia de paralelización empleada, en la que el reparto del trabajo se realiza a nivel de bloques, el paralelismo máximo explotable queda limitado por el número de hilos disponibles, tal como se muestra mediante la curva $S_{\text{máx}}(k) \approx \text{mín}(n_{threads}, n_{blocks})$. Para valores bajos de k, el número de bloques es suficientemente elevado como para saturar los hilos, mientras que para valores altos, la reducción del número de bloques limita directamente el paralelismo.

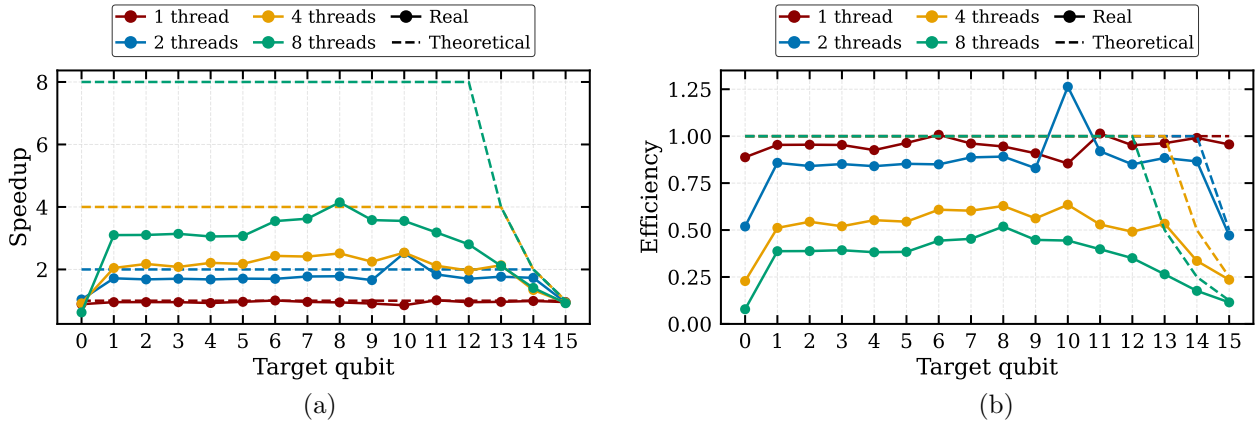
Este modelo teórico asume un reparto ideal del trabajo y no considera efectos de memoria ni sincronización. Por ello, aunque el *speedup* real no alcanza el teórico, sí mantiene su misma tendencia.

Los resultados experimentales mostrados en la Figura 4 presentan tres comportamientos diferenciados en función del qubit objetivo (Figura 4(a) muestra el speedup real, mientras que la Figura 4(b) muestra la eficiencia del paralelismo, calculada como $\frac{speedup}{n_{\text{threads}}}$). Para $k = 0$, aunque el número de bloques es elevado, el *speedup* es reducido debido a que el número de iteraciones por bloque es mínimo ($iters_{block} = 1$), lo que impide aprovechar eficazmente el paralelismo disponible, ya que la cantidad de trabajo por tarea es insuficiente para compensar los costes de paralelización.

El máximo rendimiento se alcanza en la región intermedia ($k \approx 6$–10), coincidiendo con un equilibrio más favorable entre número de bloques y carga por bloque. En esta zona, además, el tiempo de ejecución secuencial también aumenta. Los contadores hardware muestran un incremento de fallos de caché, lo que sugiere una pérdida de localidad de memoria asociada a estos valores del qubit objetivo.

Para valores altos de k, el número de bloques decrece exponencialmente, limitando directamente el paralelismo disponible. En particular, se tiene que $numBlocks = 4, 2$ y 1 para $k = 13, 14$ y 15, respectivamente. Dado que estos valores son inferiores al número de hilos considerados ($n_{threads} = 8$), el número máximo de tareas que pueden ejecutarse en paralelo queda restringido por $numBlocks$.

Como consecuencia, el *speedup* se ve limitado por el número de bloques y no por el número de hilos, lo que provoca que las curvas correspondientes a dis-



(a) (b)

Fig. 4: Comparación del rendimiento en función del qubit objetivo para distintos números de hilos. (a) Speedup real frente al teórico. (b) Eficiencia del paralelismo ($\frac{speedup}{n_{\text{threads}}}$).

tintos números de hilos tiendan a igualarse en esta región, evidenciando la pérdida de paralelismo efectivo.

Para mitigar estos efectos, se implementa una versión alternativa en la que la estructura de bucles se paraleliza mediante la cláusula `collapse` de OpenMP, que combina varios niveles de iteración en un único espacio de iteración, incrementando el número de tareas disponibles.

Los resultados de esta versión se muestran en la Figura 5 para $n_{threads} = 8$. La Figura 5(a) muestra los tiempos de ejecución, donde, a diferencia del comportamiento observado previamente, estos se mantienen aproximadamente constantes para valores altos del qubit objetivo cuando se utiliza la cláusula `collapse`. En cambio, en la versión original sin `collapse` se observa un incremento progresivo del tiempo de ejecución a medida que disminuye el número de bloques disponibles con el aumento de k.

La Figura 5(b) muestra la eficiencia del paralelismo, calculada como el cociente $\frac{speedup}{n_{\text{threads}}}$. En este caso, se observa que la eficiencia se mantiene aproximadamente constante para valores altos del qubit objetivo al emplear la cláusula `collapse`, evitando la caída observada en la versión original.

Estos resultados confirman que la reorganización del espacio de iteración mediante la cláusula `collapse` permite mitigar la limitación de paralelismo identificada en el modelo inicial.

V. Conclusiones

Los resultados obtenidos validan la plataforma SpacemiT K1 como una opción competitiva para la simulación cuántica basada en vector de estado. La configuración híbrida OpenMP + RVV presenta el mejor rendimiento, lo que confirma que la combinación de paralelismo a nivel de hilo (MIMD) y paralelismo a nivel de instrucción (SIMD) constituye una estrategia efectiva en esta arquitectura.

El rendimiento del simulador está condicionado por el tamaño del sistema. Para tamaños pequeños, la vectorización mediante RVV resulta la opción más eficiente, al evitar el *overhead* asociado a la gestión de hilos. En tamaños intermedios, el paralelismo por hilos comienza a ser beneficioso, y la versión híbrida permite explotar de forma conjunta ambos niveles de paralelismo, alcanzando el mayor *speedup*. Sin embargo, en sistemas grandes el rendimiento queda limitado por la capacidad máxima de la memoria como consecuencia del crecimiento exponencial del vector de estado.

La posición del qubit objetivo constituye otro factor determinante en el rendimiento. Aunque la cantidad total de operaciones permanece constante, su distribución varía con el valor de *stride*, lo que afecta al equilibrio entre el número de tareas disponibles y la carga de trabajo por tarea. El máximo rendimiento se alcanza cuando ambos factores están equilibrados. Para valores altos de *stride*, el número de bloques puede ser inferior al número de hilos, restringiendo el paralelismo efectivo y provocando una infrautilización de los recursos computacionales.

Para abordar esta limitación, la cláusula `collapse` de OpenMP permite reorganizar el espacio de iteración combinando varios niveles de bucle, lo que facilita un reparto más equilibrado del trabajo entre los hilos. De este modo, se mejora la eficiencia en el uso de los recursos computacionales y se evita la degradación del rendimiento en estos casos.

El rendimiento de la simulación cuántica no depende únicamente del tamaño del sistema, sino también de la estructura del acceso a memoria y de la distribución del trabajo determinada por el qubit objetivo. En este sentido, la combinación de vectorización SIMD y paralelismo MIMD, junto con estrategias de paralelización adaptativas, resulta clave para explotar eficientemente las capacidades del hardware.

Como trabajo futuro, se plantea extender estas optimizaciones a puertas cuánticas de múltiples qubits y puertas controladas, así como desarrollar esquemas de ejecución adaptativos capaces de seleccionar dinámicamente la configuración más eficiente en función del tamaño del sistema y de la posición del qubit objetivo. Asimismo, será necesario profundizar en el análisis del comportamiento de la jerarquía de memoria y de los patrones de acceso para mitigar los cuellos de botella en simulaciones de gran escala. En particular, resulta relevante analizar cómo el valor de

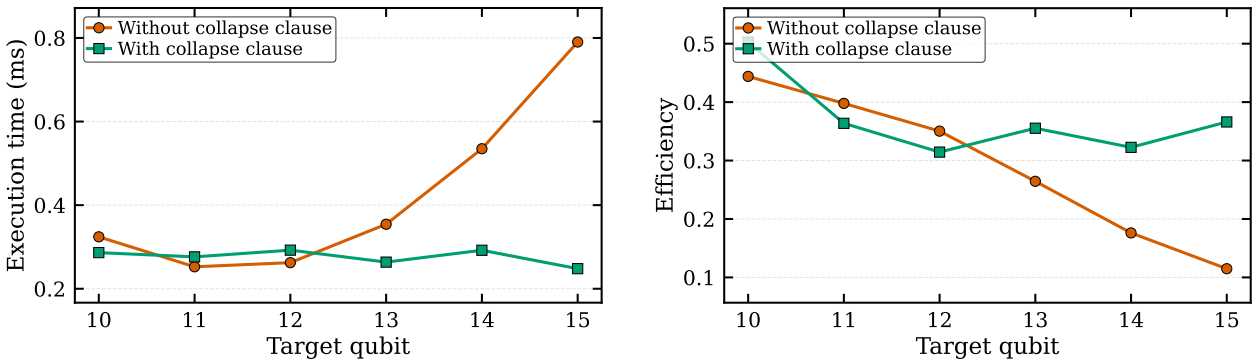


Fig. 5: Impacto de la cláusula `collapse` para $n_{threads} = 8$: (a) tiempo de ejecución; (b) eficiencia paralela en función del qubit objetivo.

stride condiciona los patrones de acceso no contiguos y cómo estos impactan en la eficiencia de la vectorización, pudiendo reducir el aprovechamiento efectivo de las unidades SIMD.

Agradecimientos

El presente trabajo ha sido financiado mediante el proyecto TSI-069100-2023-0013.

Referencias

[1] Michael A. Nielsen and Isaac L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2010.

[2] Thomas Häner, Damian S. Steiger, Mikhail Smelyanskiy, and Matthias Troyer, "Highly efficient simulation of quantum circuits," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2016.

[3] Qiskit Contributors, "Qiskit: An open-source framework for quantum computing," `https://qiskit.org`, 2023.

[4] Edwin Pednault, John A. Gunnels, Dmitri Maslov, and Jay M. Gambetta, "Breaking the 49-qubit barrier in the simulation of quantum circuits," *arXiv:1710.05867*, 2017.

[5] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 2017.

[6] Eladio Gutierrez, Sergio Romero, Maria A Trenas, and Emilio L Zapata, "Parallel quantum computer simulation on the cuda architecture," in *International Conference on Computational Science*. Springer, 2008, pp. 700–709.

[7] Eladio Gutierrez, Sergio Romero, Maria A Trenas, and Oscar Plata, "Experiences with mapping non-linear memory access patterns into gpus," in *Computational Science–ICCS 2009: 9th International Conference Baton Rouge, LA, USA, May 25-27, 2009 Proceedings, Part I*. Springer, 2009, pp. 924–933.

[8] Eladio Gutiérrez, Sergio Romero, María A Trenas, and Emilio L Zapata, "Quantum computer simulation using the cuda programming model," *Computer Physics Communications*, vol. 181, no. 2, pp. 283–300, 2010.

[9] RISC-V International, "The risc-v vector extension, version 1.0," `https://github.com/riscv/riscv-v-spec`, 2021.

[10] Michael J. McGuffin, Jean-Marc Robert, and Kazuki Ikeda, "How to write a simulator for quantum circuits from scratch: A tutorial," *arXiv:2105.01533*, 2021.

[11] Banana Pi, "Banana pi bpi-f3 official documentation," `https://wiki.banana-pi.org/Banana_Pi_BPI-F3`, 2024.

Agregación de Quantum Walks de Szegedy con Quantum Markov Chain Monte Carlo

Aingeru Ramos[1], Jose A. Pascual [2], Javier Navaridas[2] e Ivan Coluzza[3]

Resumen— **El muestreo de distribuciones de probabilidad en espacios de gran tamaño constituye un problema fundamental en numerosas áreas científicas. En computación clásica, este problema se aborda habitualmente mediante métodos Markov Chain Monte Carlo, aunque estos presentan limitaciones tanto en la aproximación de distribuciones complejas como en la escalabilidad del muestreo. En este contexto, se han propuesto diversas aproximaciones cuánticas que ofrecen o una aceleración cuadrática o una mayor fidelidad mediante convergencia asintótica. En este trabajo proponemos un esquema híbrido con el objetivo de combinar ambas ventajas. En concreto, utilizamos múltiples instancias del circuito de Szegedy y agregamos las distribuciones obtenidas mediante aplicaciones sucesivas del circuito QMCMC. Los resultados experimentales muestran que el método propuesto permite aprovechar la aceleración cuadrática junto con la convergencia asintótica de ambos métodos en la mayoría de los casos, mejorando la calidad de la distribución obtenida respecto al circuito de Szegedy estándar. Además, se identifica un sesgo que puede influir en la calidad de la aproximación, cuya mitigación se plantea como una línea prometedora de trabajo futuro.**

Palabras Clave— **Metropolis–Hastings, Quantum Walk Szegedy, Quantum Markov Chain Monte Carlo, Distribución Estacionaria**

I. Introducción

NUMEROSOS problemas de relevancia científica pueden formularse en términos de distribuciones de probabilidad. En física estadística, un problema central consiste en caracterizar la distribución de Boltzmann asociada a un sistema, que concentra la probabilidad en las configuraciones de baja energía y cuyo conocimiento permite calcular observables macroscópicos como la energía libre, la capacidad calorífica o la magnetización. Esta misma estructura aparece en biofísica, donde caracterizar la dinámica de sistemas moleculares complejos como las proteínas requiere trabajar con distribuciones de energía libre definidas sobre espacios de dimensión enormemente alta. En todos estos casos, el espacio de estados es tan vasto y la función de partición tan intratable que no existe solución cerrada para la distribución objetivo. El muestreo se convierte así en la única vía práctica para estudiar estas distribuciones, lo que requiere el uso de métodos numéricos [1].

En este contexto, los métodos Markov Chain Monte Carlo (MCMC) constituyen la herramienta estándar para la generación de muestras en computación clásica. Su principio fundamental consiste en construir una cadena de Markov ergódica cuya distribución estacionaria coincide con la distribución objetivo a través de un mecanismo de aceptación y rechazo de propuestas, de forma que las muestras generadas a lo largo de la trayectoria convergen asintóticamente hacia dicha distribución. El algoritmo de Metropolis–Hastings (MH) [2], [3], originalmente llamado Hybrid Monte Carlo, fue el primero en formalizar el esquema básico de este tipo de algoritmos. Sin embargo, el método MH presenta limitaciones cuando la distribución objetivo es multimodal o tiene barreras de probabilidad pronunciadas, pudiendo quedar la cadena atrapada en una región local de la distribución, lo que hace que el tiempo de mezcla, es decir, el tiempo necesario para alcanzar la distribución estacionaria con una precisión dada, resulte prohibitivo.

Para paliar estas limitaciones, se han propuesto extensiones y refinamientos del algoritmo MH original. Los métodos MALA [4], Hamiltonian Monte Carlo (HMC) [5] y similares, usan el gradiente de la distribución objetivo para guiar el proceso hacia las regiones de alta probabilidad con mayor eficiencia, reduciendo, en consecuencia, el tiempo de mezcla. Por su parte, los métodos Parallel Tempering [6], [7] abordan el problema de la multimodalidad ejecutando simultáneamente varias cadenas a distintas temperaturas: a mayor temperatura, la cadena se mueve más fácilmente por el espacio de búsqueda, y los intercambios periódicos entre cadenas favorecen la exploración global del espacio de estados. Los métodos adaptativos [8] constituyen otra línea de investigación activa: en lugar de fijar la distribución de propuesta a priori, esta se actualiza a lo largo de la ejecución en función de la información acumulada por la cadena, ajustándose progresivamente a la distribución objetivo. A pesar de estos avances, el muestreo eficiente de distribuciones complejas en espacios discretos de gran tamaño sigue siendo un problema abierto en el ámbito de la computación clásica.

La computación cuántica ofrece nuevas formas de abordar este problema, con la expectativa de obtener ventajas algorítmicas respecto a los métodos clásicos [9]. Subyacente al problema de muestreo, se encuentra un desafío fundamental: la carga o preparación de distribuciones de probabilidad arbitrarias en registros cuánticos. Poder codificar una distribución objetivo f como un estado cuántico es un requisito previo para cualquier esquema de muestreo cuántico, y representa por sí mismo un problema de considerable dificultad. Los enfoques variacionales, como los circuitos cuánticos paramétricos [10], han sido am-

[1]Basque Center for Materials, Applications and Nanostructures (BCMaterials), e-mail: aingeru.ramos@bcmaterials.net.

[2]Dept. of Computer Architecture and Technology, University of the Basque Country (EHU), e-mail: {joseantonio.pascual,javier.navaridas}@ehu.eus.

[3]Department of Chemistry, Ken Kennedy Institute, Center for Theoretical Biological Physics and Smalley-Curl Institute, e-mail: ic33@rice.edu.

pliamente explorados, pero su convergencia depende fuertemente de la calidad de la inicialización y del espacio de parámetros, que pueden dificultar la optimización del circuito.

El Quantum Walk de Szegedy [11] ofrece una alternativa distinta al codificar la estructura de transición de una cadena de Markov en un operador unitario, permitiendo aproximar la distribución estacionaria mediante interferencia cuántica con una aceleración cuadrática teórica respecto al tiempo de mezcla clásico. Sin embargo, su aplicación práctica presenta limitaciones relevantes: la evolución es periódica y no converge estrictamente, el tiempo óptimo depende de la brecha espectral, generalmente desconocida, y el ruido espectral puede degradar la aproximación a la distribución objetivo. Estas limitaciones motivan el desarrollo de esquemas híbridos que combinen su aceleración con mecanismos de convergencia más robustos. En este contexto, la existencia de una ventaja cuántica genuina en el muestreo sigue siendo objeto de debate, como reflejan propuestas como el Quantum-enhanced Monte Carlo [12] (QEMC) y sus análisis críticos [13].

En un trabajo previo [14] propusimos el circuito QMCMC, un circuito cuántico basado en el modelo de Discrete Quantum Walk [15] que implementa una versión cuántica del algoritmo de Metropolis–Hastings y garantiza la convergencia asintótica de la distribución marginal del registro de posición hacia la distribución objetivo. En este contexto, proponemos un esquema que combina múltiples instancias del Quantum Walk de Szegedy con el circuito QMCMC, con el objetivo de integrar la rápida aproximación inicial del primero con la convergencia asintótica del segundo. La propuesta consiste en ejecutar varias instancias de Szegedy en paralelo y agregar sus distribuciones mediante una estructura jerárquica basada en QMCMC, reinterpretando el registro de propuestas como una distribución adaptativa. Este enfoque permite mejorar empíricamente la calidad de la aproximación a la distribución estacionaria, aunque introduce un sesgo cuya magnitud depende de la geometría de la distribución objetivo.

II. Marco Teórico

A. Metropolis–Hastings

El método Metropolis–Hastings (MH) es un algoritmo iterativo que construye una cadena de Markov cuya distribución estacionaria π coincide con una distribución objetivo f de interés, permitiendo así obtener muestras de dicha distribución. En cada iteración t, se genera una propuesta x' a partir de una distribución $g(x' \mid x_t)$, denominada distribución de propuestas, donde x_t es el estado actual de la cadena. La propuesta es aceptada o rechazada según la probabilidad de aceptación

$$\alpha(x_t, x') = \min\left(1, \frac{f(x')\, g(x_t \mid x')}{f(x_t)\, g(x' \mid x_t)}\right). \qquad (1)$$

Si x' es aceptada, la cadena evoluciona al estado $x_{t+1} = x'$. En caso contrario, la cadena permanece en el estado actual, $x_{t+1} = x_t$.

De este modo, toda la dinámica de la cadena queda descrita por el kernel de transición K definido por

$$K(x, x') = g(x' \mid x)\, \alpha(x, x') + r(x)\, \delta(x' - x)$$

donde $r(x)$ es la probabilidad total de rechazo,

$$r(x) = 1 - \int g(x' \mid x)\, \alpha(x, x')\, dx'$$

y $\delta(x' - x)$ es la delta de Dirac, definida como $\delta(x' - x) = 0$ para todo $x' \neq x$ y $\delta(x' - x) = 1$ para $x' = x$, recogiendo así la probabilidad de que la cadena permanezca en el estado actual.

En este trabajo se consideran distribuciones de probabilidad definidas sobre espacios discretos finitos Σ. En este caso, la dinámica de la cadena puede describirse de forma equivalente mediante una matriz de transición $P \in \mathbb{R}^{|\Sigma|\times|\Sigma|}$, cuyos elementos $P_{x,x'} = K(x, x')$ recogen la probabilidad de transición entre cada par de estados $x, x' \in \Sigma$. Dada una distribución inicial $\hat{d}^{(0)}$, la distribución tras t pasos de la cadena viene dada por $\hat{d}^{(t)} = \hat{d}^{(0)} P^t$.

Para garantizar que el proceso converja hacia la distribución π es necesario que se cumpla la condición ergódica, que exige: (i) **irreducibilidad**, haciendo posible que cualquier estado $x' \in \Sigma$ sea alcanzable desde cualquier estado $x \in \Sigma$ en un número finito de pasos, asegurando que la cadena pueda explorar todo el espacio; y (ii) **aperiodicidad**, esto es, que la cadena no regrese a ningún estado con un periodo fijo. Ambas propiedades dependen de la estructura del espacio Σ y de la distribución de propuestas g, ya que son estos los que determinan qué transiciones son posibles. Bajo estas condiciones, el teorema ergódico garantiza la existencia de una única distribución estacionaria $\hat{\pi}$ a la que la cadena converge desde cualquier distribución inicial $\hat{d}^{(0)}$; es decir,

$$\lim_{t\to\infty} \hat{d}^{(0)} P^t = \hat{\pi}$$

Sin embargo, la condición ergódica por sí sola no determina cuál es la distribución estacionaria π, sino únicamente que existe y es única. Para asegurar que π coincide con la distribución objetivo f, se impone adicionalmente el balance detallado:

$$f(x)\, K(x, x') = f(x')\, K(x', x) \quad \forall\, x, x' \in \Sigma$$

El algoritmo Metropolis–Hastings satisface el balance detallado por construcción, gracias a la forma de la probabilidad de aceptación $\alpha(x, x')$, garantizando así que π se aproxime asintóticamente a f.

La descomposición espectral de la matriz P de un proceso que cumple las condiciones anteriores permite analizar la velocidad de convergencia. Denotando los pares autovalor–autovector de P como $\{(\lambda_i, \hat{v}_i)\}_{i=1}^n$ y ordenándolos por módulo decreciente, se tiene

$$1 = \lambda_1 > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|,$$

donde λ_1 es el autovalor asociado a la distribución estacionaria, con autovector $\hat{v}_1 = \hat{\pi} = \hat{f}$. Esta propiedad de los autovalores donde $|\lambda_i| < 1$ para todo

$i \geq 2$, consecuencia directa de la condición ergódica, es lo que permite al proceso converger hacia la distribución objetivo. Se ve más claro descomponiendo la distribución inicial $d^{(0)}$ en la base de autovectores de P,

$$d^{(0)} = \sum_{i=1}^{n} c_i \hat{v}_i,$$

donde se obtiene que

$$\lim_{t\to\infty} d^{(0)} P^t = \lim_{t\to\infty} \sum_{i=1}^{n} c_i \lambda_i^t \hat{v}_i = c_1 \hat{v}_1 = \hat{f}.$$

La igualdad $c_1 = 1$ se obtiene cuando $d^{(0)}$ está normalizado, es decir, cuando representa una distribución de probabilidad.

El autovalor λ_2 juega un papel fundamental en la estimación de la velocidad de convergencia del proceso. Dado que $|\lambda_2| \geq |\lambda_i|$ para todo $i > 2$, se tiene que, si $\lambda_2^t \to 0$, entonces $\lambda_i^t \to 0$ para todo $i > 2$, por lo que todas las contribuciones no estacionarias desaparecen. En consecuencia, la velocidad de convergencia viene determinada por el autovalor λ_2, y la distancia a la distribución estacionaria decrece a una tasa del orden de $O\left(|\lambda_2|^t\right)$. Para expresar esta velocidad de manera más directa, es habitual en la literatura usar el concepto de brecha espectral, que se define como

$$\delta = 1 - |\lambda_2|,$$

de modo que un mayor δ corresponde a una convergencia más rápida del proceso. En particular, el tiempo de mezcla, el tiempo necesario para que el proceso se aproxime a la distribución estacionaria hasta una precisión ε, está controlado por δ, y se estima como

$$t_{\text{mix}} = \mathcal{O}\left(\frac{1}{\delta}\log\frac{1}{\varepsilon}\right) \simeq \mathcal{O}\left(\frac{1}{\delta}\right). \tag{2}$$

B. Szegedy Quantum Walk

El Quantum Walk de Szegedy es una extensión cuántica de una cadena de Markov clásica, diseñada para aprovechar la interferencia cuántica y así acelerar la convergencia hacia la distribución estacionaria. La dinámica del proceso se codifica en un operador unitario W, que generaliza la matriz de transición clásica P de una cadena de Markov reversible, de manera que las propiedades espectrales de P queden reflejadas en el operador cuántico.

El circuito se define sobre dos registros que componen el espacio de Hilbert $\mathcal{H} = \mathcal{H}_i \otimes \mathcal{H}_j$, cuya base computacional viene dada por $\{|i\rangle \otimes |j\rangle\}_{i,j\in\Sigma}$. A partir de la matriz de transición P se definen dos subespacios de transición $\mathcal{A}$ y $\mathcal{B}$:

$$\mathcal{A} = \text{span}\{|\psi_\text{i}\rangle : \text{i} \in \Sigma\}, \quad \mathcal{B} = \text{span}\{|\phi_\text{j}\rangle : \text{j} \in \Sigma\},$$

donde los estados coherentes $|\psi_i\rangle$ y $|\phi_j\rangle$ vienen dados por

$$|\psi_i\rangle = |i\rangle \otimes \sum_{j\in\Sigma} \sqrt{P_{ij}}\,|j\rangle, \quad |\phi_j\rangle = \sum_{i\in\Sigma} \sqrt{P_{ij}}\,|i\rangle \otimes |j\rangle\,.$$

De forma análoga al circuito de Grover, el Quantum Walk de Szegedy explota la rotación en subespacios para amplificar las componentes del estado objetivo. Estas rotaciones se implementan mediante reflexiones respecto a los subespacios de transición $\mathcal{A}$ y $\mathcal{B}$, que codifican la información de transición de la cadena de Markov.

Se definen los proyectores de cada subespacio $\Pi_\mathcal{A} = \sum_{i\in\Sigma} |\psi_i\rangle\langle\psi_i|$ y $\Pi_\mathcal{B} = \sum_{j\in\Sigma} |\phi_j\rangle\langle\phi_j|$ respectivamente, y a partir de ellos se construyen los operadores de reflexión

$$R_\mathcal{A} = 2\Pi_\mathcal{A} - I, \quad R_\mathcal{B} = 2\Pi_\mathcal{B} - I,$$

El operador $W = R_\mathcal{B} R_\mathcal{A}$ define el operador unitario del Quantum Walk de Szegedy con el estado $|\pi\rangle$ (Ecuación 3) como estado estacionario.

$$|\pi\rangle = \sum_{i\in\Sigma} \sqrt{\pi_i}\,|\psi_i\rangle \tag{3}$$

El operador W está diseñado para rotar en cada subespacio con un ángulo de rotación dependiente de los autovalores de P, de manera que cada autovalor λ_i se traduce en un ángulo de rotación $\theta_i = \arccos(\lambda_i)$ dentro de cada subespacio específico. Estas rotaciones permiten realizar una amplificación de amplitud de las componentes asociadas a los autovalores, acercando progresivamente el estado cuántico a la distribución estacionaria.

Esta codificación espectral es la clave de la mejora frente al método clásico. Consideremos λ_2, el autovalor no estacionario más grande de P, y su ángulo de rotación asociado $\theta_2 = \arccos(\lambda_2)$. Para valores $\lambda_2 \lesssim 1$, el ángulo θ_2 es pequeño, lo que permite aproximarlo como $\theta_2 = \sqrt{2(1-\lambda_2)} \approx \mathcal{O}\left(\sqrt{\delta}\right)$[16], siendo δ la brecha espectral. Dado que cada aplicación de W produce una rotación de ángulo θ_2, se requieren aproximadamente $\mathcal{O}\left(1/\sqrt{\delta}\right)$ iteraciones para acercarse a la distribución estacionaria, lo que representa una aceleración cuadrática respecto al tiempo de mezcla clásico de la Ecuación 2.

Sin embargo, beneficiarse de la aceleración teórica del Quantum Walk de Szegedy en la práctica presenta desafíos importantes. En primer lugar, la estimación del tiempo de mezcla requiere conocer la brecha espectral δ de P, dependiente de λ_2. Al igual que en el método clásico, esta información es generalmente difícil o incluso imposible de obtener en casos reales, por lo que en la práctica es necesario recurrir a estimaciones, lo que introduce incertidumbre en el tiempo de ejecución. Además, debido a que W es un operador de rotación unitario, la evolución del estado es periódica: el estado final se aproxima y se aleja de la distribución estacionaria sin llegar a converger exactamente. Excederse en el número de iteraciones puede provocar que el estado se aleje de la distribución deseada, aumentando la sensibilidad a la estimación de la brecha espectral. Por último, el hecho de que cada autovalor de P genere una rotación diferente puede dar lugar a ruido espectral, ya

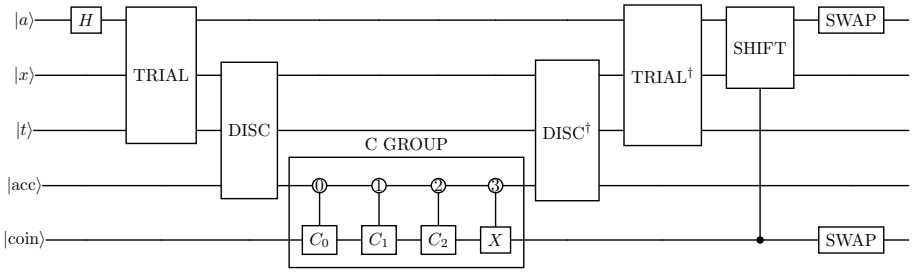


Fig. 1: Circuito cuántico para el algoritmo QMCMC. El circuito se compone de cuatro operaciones principales: TRIAL, DISC, C GROUP (un conjunto de puertas C controladas) y SHIFT. Las puertas marcadas con el símbolo † representan las inversas de sus respectivas operaciones y se incluyen para permitir la reversibilidad y la iterabilidad del proceso.

que las distintas rotaciones se acoplan y pueden interferir entre sí, aumentando el tiempo de mezcla a la distribución estacionaria.

Incluso en condiciones ideales, un factor crítico es el solapamiento del estado inicial $|\Psi_0\rangle$ con el estado estacionario cuántico $|\pi\rangle$ que es el único autovector de W con autovalor 1, es decir,

$$W\,|\pi\rangle = |\pi\rangle\,,$$

Cualquier otra componente ortogonal a $|\pi\rangle$ experimenta rotaciones periódicas dentro de los subespacios asociados a los autovalores distintos de 1, generando un comportamiento oscilatorio que nunca converge exactamente. Por ello, la aproximación del estado cuántico a $|\pi\rangle$ depende directamente del solapamiento inicial $\langle\pi|\Psi_0\rangle$: si es pequeño, la amplitud en $|\pi\rangle$ crece lentamente y la distancia al estado estacionario puede permanecer significativa, independientemente del número de iteraciones realizadas.

III. Quantum Markov Chain Monte Carlo

El circuito Quantum Markov Chain Monte Carlo (QMCMC) que propusimos en un trabajo anterior [14] implementa una versión cuántica del algoritmo de Metropolis–Hastings basada en el modelo de Quantum Walk Discreto (DQW), siguiendo paso a paso el algoritmo clásico e incorporando progresivamente, en el registro que codifica la distribución final, información en superposición sobre la distribución objetivo.

El circuito opera sobre cinco registros cuánticos: el registro de acción $|a\rangle$, el registro de posición $|x\rangle$, el registro de prueba $|t\rangle$, el registro de aceptación $|\text{acc}\rangle$ y el registro de moneda $|\text{coin}\rangle$. La Figura 1 muestra los registros y operadores que componen una iteración en el circuito QMCMC. Cada iteración del circuito se estructura en cuatro bloques principales: TRIAL, DISC, C GROUP y SHIFT, que reproducen las etapas del algoritmo de Metropolis-Hastings. En todas las expresiones que describen operadores cuánticos, se asume implícitamente que las operaciones aritméticas son módulo $|\mathcal{H}|$, donde $|\mathcal{H}|$ denota la dimensión del espacio de Hilbert del registro sobre el que se opera. El primer bloque TRIAL, definido como

$$\text{TRIAL} = \sum_{a,x,t} \left| a, x, x + a + \mathbf{1}_{\{\mathbf{a} \geq \frac{|\mathcal{H}_\mathbf{a}|}{\mathbf{2}}\}} - \frac{|\mathcal{H}_\mathbf{a}|}{\mathbf{2}} + \mathbf{t} \right\rangle \langle a, x, t|\,, \tag{4}$$

escribe en el registro $|t\rangle$ el estado propuesta en función de los estados $|a\rangle$ y $|x\rangle$. Los estados en $|a\rangle$ son usados para representar desplazamientos (a partir de ahora acciones) alrededor del estado actual $|x\rangle$. La expresión

$$x + a + \mathbf{1}_{\{\mathbf{a} \geq \frac{|\mathcal{H}_\mathbf{a}|}{\mathbf{2}}\}} - \frac{|\mathcal{H}_\mathbf{a}|}{\mathbf{2}}$$

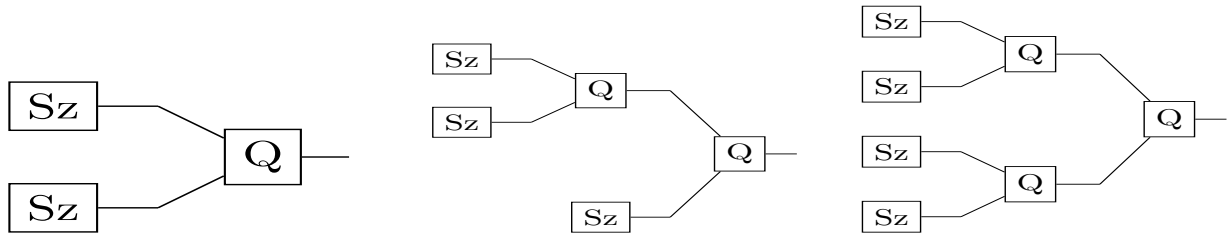
describe la aplicación de una acción a sobre el estado x, evitando que el desplazamiento total sea 0 y la operación en su conjunto sea unitaria. La probabilidad de las acciones afecta a la probabilidad de generar una propuesta u otra en t. Por ello, se puede interpretar al registro $|a\rangle$ como la distribución de propuestas g del proceso MH. Aplicando una distribución uniforme sobre todas las posibles acciones creamos una distribución g simétrica alrededor de todos los estados $|x\rangle$, es decir, $g(x_i \mid x') = g(x' \mid x_i)$.

El segundo bloque DISC, haciendo uso del estado actual en $|x\rangle$ y el estado propuesta en $|t\rangle$, calcula la probabilidad de aceptación α sobre el registro $|\text{acc}\rangle$. La probabilidad de aceptación $\alpha(t, x)$ al estar contenida en el rango continuo $(0, 1]$, no puede representarse sobre un registro cuántico discreto. Por ello, se recurre a discretizar el rango en un conjunto finito de intervalos, donde una regla de discretización $D(x, t)$ asigna un índice de intervalo en función del valor de la probabilidad de aceptación α. La precisión de la representación depende directamente del tamaño del registro $|\text{acc}\rangle$, que determina el número de intervalos disponibles. En conjunto, la puerta DISC se implementa como:

$$\text{DISC} = \sum_{x,t,\text{acc}} |x, t, D(x, t) + \text{acc}\rangle \langle x, t, \text{acc}|\,.$$

Es importante destacar que la puerta DISC en lugar de usar la probabilidad de aceptación α definida en la Ecuación 1, se beneficia del hecho de que la distribución de propuestas g es simétrica, permitiendo simplificar α a la siguiente expresión

$$\alpha(x', x_i) = \text{mín}\left(1, \frac{f(x')}{f(x_i)}\right), \tag{5}$$



(a) 2 Szegedy, 1 agregación (b) 3 Szegedy, 2 agregaciones (c) 4 Szegedy, 3 agregaciones

Fig. 2: Esquema de agregación de múltiples circuitos de Szegedy (SZ) mediante aplicaciones sucesivas del circuito QMCMC (Q). Los registros $|i\rangle$ de los circuitos SZ se combinan por pares a través de los nodos Q, propagando el registro $|x\rangle$ de salida hacia el siguiente nivel hasta obtener una única distribución final.

haciendo la puerta DISC más fácilmente implementable.

El tercer bloque C GROUP es una familia de rotaciones controladas C_i cuyo objetivo es transferir la información contenida en $|\text{acc}\rangle$ al registro $|\text{coin}\rangle$. Cada valor de $|\text{acc}\rangle$ identifica un intervalo de discretización y activa la rotación C_i correspondiente, que se elige de forma que la probabilidad de medir $|1\rangle$ coincida con el punto medio del intervalo, minimizando así el error introducido por la discretización.

El cuarto y último bloque, SHIFT, actualiza el registro de posición $|x\rangle$ de forma controlada por el registro $|\text{coin}\rangle$, implementando así la decisión de aceptación y rechazo del movimiento propuesto.

$$\text{SHIFT} = \left(\sum_{a,x} |a,x\rangle\langle a,x|\right) \otimes |0\rangle\langle 0| + \left(\sum_{a,x} \left|a, x + a + \mathbf{1}_{\{\mathbf{a} \geq \frac{|\mathcal{H}_\mathbf{a}|}{\mathbf{2}}\}} - \frac{|\mathcal{H}_\mathbf{a}|}{\mathbf{2}}\right\rangle \langle a,x|\right) \otimes |1\rangle\langle 1| \tag{6}$$

Para garantizar la iterabilidad del circuito y evitar interferencias no deseadas entre iteraciones debidas a la unitariedad de las operaciones cuánticas; un efecto observable en la evolución del DQW estándar, se aplica un proceso de descomputación al final de cada iteración. Los registros $|t\rangle$ y $|\text{acc}\rangle$ se revierten al estado $|0\rangle$ mediante la aplicación de los operadores inversos TRIAL† y DISC†. Sin embargo, los registros $|a\rangle$ y $|\text{coin}\rangle$ no pueden descomputarse sin alterar el estado del registro $|x\rangle$, que en ese momento ya codifica la distribución de probabilidad acumulada. Por ello, estos registros se renuevan en cada iteración mediante puertas SWAP que los intercambian por registros limpios inicializados a $|0\rangle$. Una limitación de la renovación de registros es que el número total de qubits aumenta linealmente con el número de iteraciones, según

$$(n_a + 1)t + 2n_x + n_{\text{acc}}$$

donde t es el número de iteraciones y n_a, n_x y n_{acc} son los tamaños de cada registro del circuito[1].

Operar el circuito QMCMC con los registros $|a\rangle$ y $|x\rangle$ en superposición, permite evaluar simultáneamente todas las acciones disponibles desde todos los estados del espacio de búsqueda e incorporar dicha información de vuelta al registro $|x\rangle$, actualizando así la distribución representada hacia valores progresivamente más cercanos a π. Esta capacidad de integrar coherentemente la información en $|x\rangle$ es la que dota al circuito de un comportamiento de convergencia asintótica hacia la distribución objetivo π, a diferencia del Quantum Walk de Szegedy, cuya evolución es periódica y no converge hacia ninguna distribución estacionaria. Sin embargo, este comportamiento viene a costa de un crecimiento lineal en el número de qubits por iteración. Cabe destacar también que el QMCMC no presenta una aceleración cuadrática análoga a la del Quantum Walk de Szegedy.

A. QMCMC como método de agregación

Al final del artículo [14] se menciona que el registro $|a\rangle$ y su superposición pueden interpretarse como la distribución de propuestas g del método Metropolis-Hastings, así como la posibilidad de implementar métodos adaptativos con el circuito QMCMC. En los métodos MCMC adaptativos [8], la distribución g deja de ser fija y se actualiza a lo largo de la ejecución en función de la información recogida por la cadena. Esta actualización permite ajustar progresivamente g para reflejar mejor la geometría de la distribución objetivo, mejorando la eficiencia del muestreo sin comprometer la convergencia hacia la distribución estacionaria bajo las condiciones adecuadas.

Resulta difícil incorporar información adaptativa al registro $|a\rangle$, dado que sus estados representan un desplazamiento relativo al estado actual de $|x\rangle$. Para facilitar la incorporación de la adaptabilidad en el circuito, hemos optado por modificar el registro, ahora codificando una distribución de probabilidad sobre el mismo espacio de $|x\rangle$ e interpretando sus estados directamente como los estados propuesta. Con este cambio, el registro $|x\rangle$ tras una iteración, que contiene parte de la información de π, puede retroalimentarse como registro $|a\rangle$ en la siguiente, dotando así a la distribución g de la información adquirida en la iteración anterior e implementando la adaptabilidad. Siguiendo este enfoque, los operadores TRIAL y SHIFT de las Ecuaciones 4 y 6 se modifican a

$$\text{TRIAL} = \sum_{a,x,t} |a,x,a+t\rangle\langle a,x,t|$$

[1]Dado que el tamaño del registro $|t\rangle$ es siempre $n_t = n_x$, el tamaño del registro $|x\rangle$ se tiene en cuenta dos veces en la expresión anterior.

$$\text{SHIFT} = \left(\sum_{a,x} |a,x\rangle\langle a,x|\right) \otimes |0\rangle\langle 0| + \left(\sum_{a,x} |x,a\rangle\langle a,x|\right) \otimes |1\rangle\langle 1|$$

donde el estado $|a\rangle$ pasa directamente como estado propuesta a $|t\rangle$, así como su amplitud de probabilidad, tal y como se ha mencionado previamente. Las puertas DISC y C GROUP se mantienen inalteradas. Con estos cambios, el circuito permite emplear distribuciones alternativas como la distribución de propuestas g, ofreciendo la posibilidad de diseñar mecanismos de propuesta más sofisticados, con el potencial de mejorar tanto la eficiencia como las propiedades de convergencia del método QMCMC.

La Figura 2 ilustra el esquema de ejecución propuesto: se lanzan múltiples circuitos de Szegedy (SZ) en paralelo y sus salidas se agregan mediante aplicaciones sucesivas del circuito QMCMC (Q). En el primer nivel del árbol (lado izquierdo), se extraen los registros $|i\rangle$ de los circuitos Szegedy y son introducidos por pares como registros $|a\rangle$ y $|x\rangle$ del circuito QMCMC, agregando ambas distribuciones en el registro $|x\rangle$ en la salida. En los siguientes niveles, se introducen los resultados intermedios de los diferentes registros $|x\rangle$ y se combinan sucesivamente hasta obtener una única distribución final. De este modo, se busca integrar la aceleración cuadrática en la aproximación a π propia del Quantum Walk de Szegedy con la convergencia asintótica del circuito QMCMC. La Figura 2 muestra el esquema de ejecución para 2, 3 y 4 ejecuciones simultáneas del circuito de Szegedy.

IV. Resultados

En esta sección se presentan los resultados obtenidos mediante simulación, comparando el comportamiento del circuito de Szegedy original con el circuito propuesto. Las simulaciones se han realizado sobre tres distribuciones estacionarias de referencia: (i) una distribución concentrada en un único punto (Subfigura 3a), (ii) una mixtura de gaussianas trimodal con media y amplitud variables (Subfigura 3b), y (iii) la distribución de Boltzmann del modelo de Ising bidimensional de 9 espines, definida sobre el espacio de configuraciones de espines (Subfigura 3c).

La Figura 4 ofrece una visión global del comportamiento de los circuitos evaluados sobre cada una de las distribuciones anteriores. En cada subfigura se representan cuatro curvas, diferenciadas por color, correspondientes a las distintas configuraciones estudiadas: el circuito de Szegedy original (azul) y el circuito propuesto con 2 (naranja), 3 (verde) y 4 (lila) instancias de Szegedy agregadas, según el esquema descrito en la Figura 2. El eje horizontal representa el número de iteraciones aplicadas a los circuitos de Szegedy, mientras que el eje vertical cuantifica la cercanía entre la distribución obtenida y la distribución objetivo π mediante la distancia de variación total D_{TV}, una métrica ampliamente utilizada en la literatura para comparar distribuciones de probabilidad. Formalmente, dada una distribución final d y la distribución objetivo π, dicha distancia se define como:

$$D_{\text{TV}}(d,\pi) = \frac{1}{2}\sum_x |d(x) - \pi(x)|\,.$$

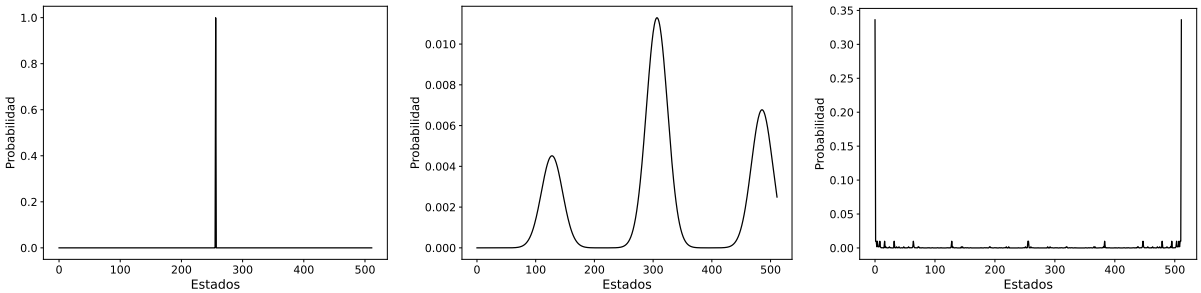
Esta cantidad toma valores en el intervalo $[0,1]$, donde 0 indica coincidencia exacta entre ambas distribuciones y 1 corresponde a la máxima discrepancia posible.

A. Análisis de resultados

Atendiendo en primer lugar a los resultados del circuito de Szegedy estándar (líneas azules), se observan los efectos descritos en la Subsección II-B. En las distribuciones con probabilidad altamente concentrada (Subfiguras 4a y 4c), el solapamiento inicial entre el estado uniforme de partida y el estado estacionario es reducido, lo que se refleja en una distancia de variación total elevada al inicio. En la distribución de gaussianas (Subfigura 4b), al ser más dispersa, dicho solapamiento es mayor y el algoritmo parte desde una posición más cercana a la distribución objetivo. Respecto al ruido espectral, en la distribución concentrada en un punto (Subfigura 4a) se aprecia un comportamiento claramente cíclico: al existir un único autovalor no estacionario λ_2, la evolución es una rotación simple que produce una dinámica sinusoidal bien definida. En los casos de las Subfiguras 4b y 4c, en cambio, la presencia de varios autovalores no estacionarios no nulos introduce ruido espectral, elimina dicho comportamiento y da lugar a dinámicas más irregulares.

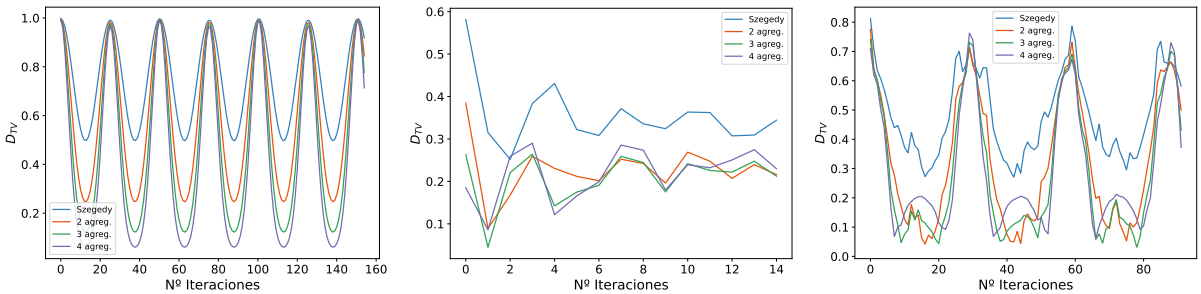
Poniendo el foco en los resultados obtenidos con el circuito propuesto, se observa una mejora generalizada al agregar dos ejecuciones (lineas naranja) del circuito de Szegedy en todos los casos de la Figura 4, reduciendo la distancia a la distribución objetivo de forma consistente independientemente del número de iteraciones. La Subfigura 4a muestra que el incremento en el número de agregaciones continúa mejorando la aproximación, aunque en menor medida, ya que, debido a la convergencia asintótica del QMCMC, a medida que el sistema se aproxima a la distribución estacionaria, la tasa de mejora disminuye. Este comportamiento no se observa en las Subfiguras 4b y 4c, donde, a partir de la segunda agregación, los resultados comienzan a empeorar, siendo especialmente notable el caso de la Subfigura 4c con cuatro agregaciones.

Conviene tener en cuenta que la implementación original del QMCMC emplea una versión simplificada de la probabilidad de aceptación α del algoritmo de Metropolis-Hastings, en la que se asume que la distribución de propuesta g es completamente simétrica. Sin embargo, con la configuración actual del circuito QMCMC donde $|a\rangle$ ha sido modificado, dicha propiedad deja de cumplirse, lo que provoca un cálculo erróneo de α e introduce inevitablemente un sesgo hacia los estados de máxima probabilidad. Calcular fielmente la función de aceptación de la Ecuación 1 requeriría recurrir a técnicas de estimación de



(a) Concentrada en un punto (b) Mixtura de gaussianas (c) Boltzmann (Ising 3x3)

Fig. 3: Las tres distribuciones usadas como distribución objetivo para comparar el circuito Szegedy frente al circuito propuesto: (a) una distribución centrada en un punto, (b) una mixtura de gaussiana trimodal con medias y amplitudes variables y (c) la distribución de Boltzmann resultante de un modelo Ising bidimensional de 9 espines.



(a) Concentrada en un punto (b) Mixtura de gaussianas (c) Boltzmann (Ising 3x3)

Fig. 4: Distancia de variación total entre la distribución resultante y la distribución objetivo (Figura 3), comparando el circuito de Szegedy estándar (azul) con el circuito propuesto con 2 (naranja), 3 (verde) y 4 (lila) agregaciones.

amplitud, lo que elevaría el coste del circuito hasta el punto de anular cualquier ventaja computacional.

La magnitud del sesgo depende de la geometría de la distribución estacionaria π: cuanto más abruptas y abundantes sean las variaciones de probabilidad entre estados vecinos en π, mayor es el sesgo acumulado en cada iteración.

La distribución concentrada en un único punto introduce poco sesgo pese a su forma extrema: al ser un pico rodeado de estados con probabilidades similares entre sí, la mayoría de cocientes $\frac{\pi(x')}{\pi(x)}$ son cercanos a 1. Esto se refleja en la Subfigura 4a, donde aumentar el número de agregaciones sigue siendo beneficioso y el sesgo apenas tiene impacto. El caso opuesto se aprecia en la Subfigura 4b, donde el sesgo detiene la mejora a partir de la segunda agregación. La mixtura de gaussianas presenta una alta variabilidad entre las probabilidades de los estados, lo que amplifica las diferencias en los cocientes $\frac{\pi(x')}{\pi(x)}$ y, con ello, el efecto del sesgo.

La distribución de Ising se sitúa en un punto intermedio. El sesgo que introduce es mayor que en el caso de la Subfigura 4a, ya que ambas distribuciones comparten parte de su estructura, pero no lo suficiente como para anular los beneficios de la agregación en las primeras etapas. Esto se aprecia en la Subfigura 4c: añadir una tercera agregación (verde) sigue siendo ventajoso en la mayoría de los casos, pero con cuatro (lila) el método deja de ser efectivo. Una vez que la aproximación está cerca de π, la ganancia de nuevas agregaciones disminuye y el sesgo pasa a dominar, empeorando el resultado.

B. Costes de ejecución y recursos

El uso de múltiples ejecuciones de Szegedy conlleva un incremento proporcional en el número total de iteraciones ejecutadas respecto al circuito original. Siendo t el número de iteraciones realizadas por cada bloque Szegedy, c circuitos Szegedy ejecutan conjuntamente $c \cdot t$ iteraciones. Para agregar los c circuitos Szegedys y los resultados intermedios a un único registro son necesarias $c-1$ ejecuciones del circuito QMCMC y, asumiendo un coste de una iteración por QMCMC, el número total de iteraciones dado un t del circuito propuesto es $(c+1)t-1$. Suponiendo que $t = \mathcal{O}\left(\frac{1}{\sqrt{\delta}}\right)$, como se discute en la Subsección II-B correspondiente de Szegedy en el marco teórico, el orden de ejecución del circuito propuesto es de $\mathcal{O}\left(\frac{c}{\sqrt{\delta}}\right)$, lo que implica que el coste computacional sigue siendo del mismo orden que el de Szegedy original, $\mathcal{O}\left(\frac{1}{\sqrt{\delta}}\right)$, pero con el factor constante c adicional independiente del tamaño del problema.

En cuanto al coste en recursos, dado que es necesario ejecutar de manera simultánea c circuitos de Szegedy, y cada uno de estos circuitos requiere dos registros de tamaño n qubits, el coste total en términos de qubits es $2cn$. Para la agregación de los circuitos Szegedy y las soluciones intermedias, son reaprovechados los registros anteriores como registros $|a\rangle$ y $|x\rangle$, por lo que el único coste adicional de un circuito QMCMC es un qubit para el registro $|\text{coin}\rangle$. Teniendo en cuenta que son necesarios $c-1$ circuitos QMCMC, el coste total en recursos asciende a

$2cn + c - 1$.

Cabe señalar que esta expresión no incluye los registros auxiliares reutilizables necesarios en los circuitos QMCMC, tales como $|t\rangle$ y $|\text{acc}\rangle$. Por lo tanto, el coste en recursos, de manera formal, se puede expresar como $\mathcal{O}(2cn + c) \approx \mathcal{O}(cn)$.

V. Conclusiones

En este trabajo hemos presentado un esquema que combina el Quantum Walk de Szegedy con el circuito QMCMC con el objetivo de integrar la aceleración cuadrática del primero con la convergencia asintótica garantizada del segundo. La propuesta consiste en ejecutar múltiples instancias del circuito de Szegedy en paralelo y agregar sus distribuciones resultantes mediante iteraciones del circuito QMCMC, reinterpretando el registro $|a\rangle$ como distribución de propuestas adaptativa en lugar de un simple desplazamiento. El análisis de costes muestra que el orden de ejecución del esquema propuesto se mantiene en $\mathcal{O}\left(\frac{c}{\sqrt{\delta}}\right)$, es decir, en el mismo orden que el circuito de Szegedy original, con un factor multiplicativo constante c independiente del tamaño del problema, y con un coste en recursos de $\mathcal{O}(2cn + c)$ qubits.

Los resultados experimentales muestran que la agregación mediante QMCMC mejora consistentemente la calidad de la distribución obtenida respecto al circuito de Szegedy estándar, y que incrementar el número de instancias agregadas produce mejoras progresivas en la aproximación a la distribución estacionaria. Sin embargo, se ha identificado un efecto de sesgo introducido por el circuito QMCMC al operar con una distribución de propuestas asimétrica, cuya magnitud depende de la geometría de la distribución objetivo. Este sesgo resulta especialmente pronunciado en distribuciones con grandes variaciones en la probabilidad de los estados, como en la distribución de mixtura de gaussianas trimodal, donde llega a degradar los resultados respecto al circuito de Szegedy original.

Como líneas de trabajo futuro, identificamos dos direcciones principales. En primer lugar, (i) es necesario abordar el efecto del sesgo introducido por la asimetría de la distribución de propuestas. Una posible vía consiste en incorporar un mecanismo de corrección de la probabilidad de aceptación en la puerta DISC a medida que avanza la ejecución del esquema propuesto, estimando el sesgo acumulado y corrigiéndolo. En segundo lugar, (ii) el presente trabajo se ha desarrollado exclusivamente sobre la formulación original del Quantum Walk de Szegedy, si bien existen variantes y extensiones de este circuito con propiedades espectrales y de convergencia mejoradas. Entre ellas, cabe destacar las versiones que incorporan información del gradiente de la distribución objetivo para sesgar las propuestas hacia regiones de alta probabilidad [17] o formulaciones que modifican los operadores de reflexión para reducir el ruido espectral introducido por la interferencia entre las distintas frecuencias de rotación [18]. Extender el esquema de agregación propuesto a estas versiones modificadas constituye una dirección natural de investigación, con el objetivo de estudiar si el circuito QMCMC es capaz de preservar sus propiedades mejoradas al actuar como mecanismo de agregación, y si la combinación de múltiples instancias amplifica las ganancias observadas en este trabajo.

Agradecimientos

Este trabajo está financiado por las ayudas IKUR-Quantum, y los proyectos IT1504-22 y KK-2025/00098 del Gobierno Vasco y también por los proyectos PID2023-152390NB-I00 y CNS2023-144315, financiadas por el MICIU/AEI/10.13039/9/501100011033, la iniciativa European Union NextGenerationEU/PRTR y los fondos FEDER. El Dr. Javier Navaridas es Investigador Ramón y Cajal, financiado por el Ministerio de Ciencia, Innovación y Universidades (RYC2018-024829-I), por el MCIU/AEI/10.13039/501100011033 y, en su caso, por el programa ESF Investing in your future, o por la iniciativa European Union NextGenerationEU/PRTR. El trabajo en el Center for Theoretical Biological Physics ha sido financiado por las subvenciones NSF-PHY-2019745 y NSF-PHY-2210291, además de la Welch Foundation (C-1792).

Referencias

[1] Daan Frenkel and Berend Smit, *Understanding Molecular Simulation: From Algorithms to Applications*, Academic Press, Inc., USA, 1st edition, 1996.

[2] Nicholas Metropolis, Arianna W. Rosenbluth, and et al., "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 06 1953.

[3] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 04 1970.

[4] Gareth O. Roberts and Richard L. Tweedie, "Exponential convergence of Langevin distributions and their discrete approximations," *Bernoulli*, vol. 2, no. 4, pp. 341–363, 1996.

[5] Simon Duane, Anthony D. Kennedy, Brian J. Pendleton, and Duncan Roweth, "Hybrid Monte Carlo," *Physics Letters B*, vol. 195, no. 2, pp. 216–222, 1987.

[6] David J. Earl and Michael W. Deem, "Parallel tempering: Theory, applications, and new perspectives," *Phys. Chem. Chem. Phys.*, vol. 7, pp. 3910–3916, 2005.

[7] Błażej Miasojedow, Eric Moulines, and Matti Vihola, "An adaptive parallel tempering algorithm," *Journal of Computational and Graphical Statistics*, vol. 22, no. 3, pp. 649–664, 2013.

[8] Heikki Haario, Eero Saksman, and Johanna Tamminen, "An adaptive metropolis algorithm," *Bernoulli*, vol. 7, no. 2, pp. 223–242, 2001.

[9] John Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, pp. 79, Aug. 2018.

[10] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014, arXiv:1411.4028.

[11] Mario Szegedy, "Quantum speed-up of markov chain based algorithms," in *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, Rome, Italy, 2004, pp. 32–41, IEEE Computer Society.

[12] David Layden, Guglielmo Mazzola, Ryan V. Mishmash, Mario Motta, Pawel Wocjan, Jin-Sung Kim, and Sarah Sheldon, "Quantum-enhanced Markov chain Monte Carlo," *Nature*, vol. 619, pp. 282–287, 2023.

[13] Alev Orfi and Dries Sels, "Bounding the speedup of the quantum-enhanced Markov-chain Monte Carlo algorithm," *Physical Review A*, vol. 110, no. 5, pp. 052414, 2024.

[14] Aingeru Ramos, Jose A. Pascual, Javier Navaridas, and Ivan Coluzza, "On the convergence of markov chain distribution within quantum walk circuit subspace," 2025.
[15] Andris Ambainis, Julia Kempe, and Umesh Vazirani, "One-dimensional quantum walks," in *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, New York, NY, USA, 2001, pp. 50–59, ACM.
[16] Pawel Wocjan and Kristan Temme, "Szegedy walk unitaries for quantum maps," *Communications in Mathematical Physics*, vol. 402, pp. 3201, 2023.
[17] Jessica Lemieux, Bettina Heim, David Poulin, Krysta Svore, and Matthias Troyer, "Efficient quantum walk circuits for the Metropolis-Hastings algorithm," *Quantum*, vol. 4, pp. 287, 2020, arXiv:1910.01659.
[18] Pablo Rodenas Ruiz and Baptiste Claudon, "Quantum circuits for the Metropolis-Hastings algorithm," *arXiv preprint*, 2025, arXiv:2506.11576.

Scheduling Hybrid Classical–Quantum Workflows with Session Constraints

Jose A. Pascual[1], Alex Amenabar[1], Unai Garciarena[1] and Javier Navaridas[1]

Abstract— **Hybrid classical–quantum applications alternate between classical computation and quantum circuit execution on remote quantum processors. Modern platforms reduce queue delays through sessions: workflows that submit consecutive jobs within a time limit retain priority access to the processor. In the standard execution model, each workflow runs as a single job that holds all resources until completion; this simplifies session management but leaves the processor idle during classical phases. Decomposing each workflow into independent sub-jobs, one per iteration, allows the HPC scheduler to fill these idle gaps with jobs from other workflows, therefore improving processor utilisation. However, this approach releases CPU resources between sub-jobs, requiring them to be reacquired in competition with other workflows. If classical processing is delayed beyond the session time limit, priority access is lost and waiting times increase, diminishing the benefits of decomposition. We study this scheduling problem by proposing a constraint programming model that captures classical resource constraints, quantum execution ordering, and session requirements, and by designing scheduling strategies that account for session continuity and objective-aware job selection. On a benchmark of 600 synthetic instances, session-aware policies consistently outperform those that ignore sessions across all configurations, reducing the median deviation from the best-known exact solution from over 55% to below 30%. This advantage is robust under different objective settings, indicating that session continuity is a key factor for achieving high-quality schedules in hybrid classical–quantum systems.**

Keywords— **Hybrid classical–quantum computing, workflow scheduling, resource-constrained scheduling, quantum sessions**

I. Introduction

HY brid classical–quantum computing has emerged as the standard approach for executing near-term quantum algorithms [1]. In this model, quantum processors act as accelerators within larger workflows managed by classical computing systems. Algorithms such as the Variational Quantum Eigensolver (VQE) [2], the Quantum Approximate Optimization Algorithm (QAOA) [3], and several quantum machine learning methods follow this pattern, iteratively alternating between classical computation and quantum circuit execution.

In real deployments, hybrid workflows run in environments where classical high-performance computing (HPC) systems interact with remote quantum backends [4], [5]. Each iteration consists of classical preprocessing, quantum circuit execution on a quantum processing unit (QPU), and classical postprocessing that updates parameters or analyses measurement results. Because quantum resources are scarce and shared among many users, QPU waiting time is the main cost in hybrid execution [6]. Modern quantum platforms, specifically IBM Quantum and its Qiskit Runtime framework [7], address this through *execution sessions*: a workflow that submits consecutive quantum jobs within a predefined interactive time-to-live (TTL) retains priority access to the QPU, avoiding the standard queue. Sessions can significantly reduce waiting time, but they require that classical processing between iterations finishes within the TTL, a timing constraint that depends on how classical resources are managed.

A separate challenge is that a workflow running as a single job holds both classical and quantum resources for its entire duration, leaving the QPU idle during classical phases and blocking other workflows. Esposito *et al.* [8], [9] address this by decomposing each hybrid workflow into sub-jobs, one per iteration, submitted independently to the HPC resource manager (e.g., SLURM [10]). This allows the resource manager to interleave sub-jobs from different workflows on the QPU, filling classical idle gaps and reducing both QPU idle time and total wall time. The approach requires moderate changes: sub-jobs exchange state through checkpointing and the only user intervention is splitting the workflow and writing the job scripts.

The sub-job decomposition improves QPU utilisation but makes session continuity harder to guarantee. Between two consecutive iterations, CPU resources are released and must be reacquired by competing with sub-jobs from other workflows. If this competition delays classical preprocessing past the interactive TTL, the session expires and the workflow loses priority access, increasing its next QPU waiting time. The two mechanisms, session-based priority access and sub-job interleaving, are each beneficial on their own, but combining them requires scheduling policies that explicitly account for session continuity under CPU contention.

In this paper, we study the scheduling of hybrid workflows decomposed into sub-jobs that share a classical HPC system of limited capacity and submit quantum jobs to a single QPU. Session constraints impose an interactive TTL and a maximum session TTL on consecutive quantum submissions. The resulting scheduling problem combines resource-constrained scheduling with timing dependencies caused by the quantum backend.

To address this problem, we develop a discrete-time optimisation model that captures classical resource constraints, quantum execution ordering, and session requirements. The model extends classical

[1]Intelligent Systems Group (ISG), Department of Computer Architecture and Technology, University of the Basque Country EHU, e-mail: {joseantonio.pascual, alex.amenabar, unai.garciarena, javier.navaridas}@ehu.eus.

scheduling formulations by adding timing constraints that depend on session state. A constraint programming solver is used to obtain reference solutions for moderate-size instances against which we evaluate five session-aware scheduling policies, including deadline-based prioritisation and an objective-aware quantum job selection policy that balances completion time and session usage.

The experimental evaluation reveals both the difficulty of the problem and the impact of session-aware scheduling. On a benchmark of 600 instances, the exact solver proves optimality in only 5 cases and reaches the prescribed gap tolerance in 54 more, with a median optimality gap of 8.63%. When measured against the best known exact solution, session-aware heuristics achieve a median deviation of approximately 28–29%, while policies that ignore session constraints show median deviations above 55%. In addition, session-aware strategies achieve session success rates above 91% and significantly shorter quantum waiting times. This confirms the importance of preserving session continuity.

The main contributions of this work are a unified CP-SAT formulation for hybrid classical–quantum scheduling with session constraints, five session-aware scheduling policies including an objective-aware variant, and an experimental analysis on 600 instances showing that session-aware strategies consistently outperform session-unaware ones across all configurations and objective settings.

II. Related Work

The scheduling problem studied here combines elements of resource-constrained scheduling, heterogeneous accelerator scheduling, and hybrid classical–quantum workflow management.

The execution of workflows with precedence constraints under limited computational resources is commonly modelled using the resource-constrained project scheduling problem (RCPSP) [11], [12]. Extensions such as RCPSP/max [13] add minimum and maximum time gaps between tasks, allowing more flexible timing. These models are closely related to the hybrid workflows considered in this work, where classical and quantum stages are connected through precedence and timing constraints. Classical RCPSP formulations do not, however, account for external resources whose execution behaviour depends on interaction patterns, as is the case with quantum backends. The same gap applies to standard HPC scheduling policies such as FCFS, priority-based scheduling, and backfilling [14], [15], which prioritise scalability but do not capture the dependency between classical and quantum processing stages.

Heterogeneous DAG schedulers such as HEFT [16] and related list schedulers [17] go further by jointly ordering tasks and assigning them to processors, but assume fixed task durations and do not apply here. Classical HPC resources are shared under a capacity constraint rather than assigned to individual processors, the QPU is a single sequential resource shared by all workflows, and session constraints introduce non-linear costs that fixed-duration models cannot capture. On the quantum side, Ravi *et al.* [6], [18] study quantum cloud scheduling empirically, showing that waiting times can be reduced by more than 3× by predicting queue lengths and hardware trends, but both works focus on dispatching individual jobs rather than iterative hybrid workflows with session continuity requirements.

Several works address the broader challenge of integrating quantum devices into HPC systems. Schulz *et al.* [4] and Elsharkawy *et al.* [5] identify scheduling and resource allocation as key open problems. Weder *et al.* [19] study the orchestration of hybrid applications, and Cranganore *et al.* [20] formalise hybrid workflows and their mapping onto quantum and classical resources. These works call for dedicated scheduling approaches but do not study the decisions that arise when multiple workflows share classical and quantum resources concurrently.

The work most directly related to ours is that of Esposito *et al.* [8], [9], who propose splitting each hybrid workflow into sub-jobs submitted independently to SLURM [10], allowing the scheduler to interleave executions from different workflows and reduce QPU idle time. However, they focus on implementation and QPU utilisation gains and do not consider session-based priority access: once sub-jobs compete for CPU between iterations, classical delays can push the next quantum submission past the inter-submission limit and break the session. Seitz *et al.* [21] address a different aspect by minimising makespan using circuit cutting, and Pehlivanoglu *et al.* [22] jointly optimise queue time and fidelity across heterogeneous cloud providers. IBM has also released open-source SLURM plugins that treat QPUs as standard HPC resources [23].

None of the above works model session-based priority access or study how CPU contention between sub-jobs affects session continuity. Viviani *et al.* [24] recently identified hybrid scheduling as the most underexplored challenge in HPC-QC integration, and the interactive TTL and maximum session TTL constraints that govern priority access in platforms such as Qiskit Runtime [7] are not considered in any of these formulations. This is the problem we study.

III. System Model

We consider a hybrid computing environment composed of a classical HPC system connected to an external quantum backend. Hybrid applications interact iteratively with the quantum processor, executing classical computations between successive quantum circuit evaluations.

The classical system provides a pool of C homogeneous compute resources. The scheduling horizon runs over discrete time steps $t \in \{0, \dots, H\}$, and all start times and durations are integer-valued.

The workload consists of a set $\mathcal{J}$ of hybrid workflows. Each workflow $j \in \mathcal{J}$ performs K_j iterations. Each iteration (j, k), $k \in \{1, \dots, K_j\}$, consists of

Table I: Main notation used in the scheduling model.

Symbol	Description
C	Total classical resource capacity
H	Scheduling horizon
$\mathcal{J}$	Set of workflows
K_j	Number of iterations of workflow j
$p^{\text{pre}}_{j,k}$	Preprocessing duration
$p^{\text{post}}_{j,k}$	Postprocessing duration
$r^{\text{pre}}_{j,k}$	Preprocessing resource demand
$r^{\text{post}}_{j,k}$	Postprocessing resource demand
$q_{j,k}$	Quantum execution duration
a_j	Arrival time of workflow j
θ_j	Interactive TTL
γ_j	Maximum session TTL
$S^{\text{pre}}_{j,k}$	Start time of preprocessing
$S^{q}_{j,k}$	Start time of quantum execution
$S^{\text{post}}_{j,k}$	Start time of postprocessing
$F^{q}_{j,k}$	Quantum job completion time
$\delta_{j,k}$	Session continuation indicator
$S^{\text{sess}}_{j,k}$	Session segment start containing job k
C_j	Completion time of workflow j
U	Total session usage
λ	Session usage penalty weight

three sequential stages:

1. preprocessing on CPU,
2. quantum execution on QPU,
3. postprocessing on CPU.

For each iteration (j,k): $p^{\text{pre}}_{j,k}$ and $p^{\text{post}}_{j,k}$ are the preprocessing and postprocessing durations; $r^{\text{pre}}_{j,k}$ and $r^{\text{post}}_{j,k}$ are the corresponding CPU resource demands; $q_{j,k}$ is the quantum execution duration. The main notation is summarised in Table I.

Each workflow j has an arrival time a_j, an interactive TTL θ_j, and a maximum session TTL γ_j. The interactive TTL θ_j is the maximum time allowed between the completion of one quantum job and the submission of the next; exceeding it deactivates the session. The maximum session TTL γ_j is the total allowed duration from session start, after which the session is permanently closed. The precedence constraints between stages are defined in terms of the start times $S^{\text{pre}}_{j,k}$, $S^{q}_{j,k}$, and $S^{\text{post}}_{j,k}$ of preprocessing, quantum execution, and postprocessing, respectively, for all $j \in \mathcal{J}$ and $k \in \{1,\dots,K_j\}$:

$$S^{\text{pre}}_{j,k} \geq a_j, \tag{1}$$

$$S^{q}_{j,k} \geq S^{\text{pre}}_{j,k} + p^{\text{pre}}_{j,k}, \tag{2}$$

$$S^{\text{post}}_{j,k} \geq S^{q}_{j,k} + q_{j,k}, \tag{3}$$

$$S^{\text{pre}}_{j,k+1} \geq S^{\text{post}}_{j,k} + p^{\text{post}}_{j,k}, \quad k < K_j. \tag{4}$$

Classical preprocessing and postprocessing tasks consume HPC resources. At any time t, total resource usage cannot exceed system capacity:

$$\sum_{i \in \mathcal{A}(t)} r_i \leq C, \tag{5}$$

where $\mathcal{A}(t) = \{i \mid S_i \leq t < S_i + p_i\}$ is the set of classical tasks active at time t, with S_i, p_i, and r_i denoting the start time, duration, and resource demand of task i.

The quantum backend executes jobs sequentially on a single resource. We denote the start and completion times of quantum job (j,k) as $S^{q}_{j,k}$ and $F^{q}_{j,k} = S^{q}_{j,k} + q_{j,k}$, respectively.

Hybrid workflows may execute quantum jobs within sessions. When a session is active, the quantum processor remains reserved for the corresponding workflow between consecutive jobs. This prevents other workflows from using it during idle gaps. A binary indicator $\delta_{j,k}$, defined for $k \in \{1,\dots,K_j - 1\}$, represents whether jobs k and $k+1$ of workflow j belong to the same session. Session feasibility is defined by two conditions. First, consecutive jobs within a session must be submitted within the interactive TTL. When a session is active, jobs receive priority access and incur negligible queue waiting time, so the submission time equals the start time $S^{q}_{j,k+1}$:

$$\delta_{j,k} = 1 \;\Rightarrow\; S^{q}_{j,k+1} - F^{q}_{j,k} \leq \theta_j. \tag{6}$$

Second, all jobs in a session segment must complete within the lifetime limit measured from the segment start $S^{\text{sess}}_{j,k}$:

$$\delta_{j,k} = 1 \;\Rightarrow\; F^{q}_{j,k+1} \leq S^{\text{sess}}_{j,k} + \gamma_j. \tag{7}$$

Here $S^{\text{sess}}_{j,k}$ denotes the start of the session segment containing job k, defined recursively as:

$$S^{\text{sess}}_{j,k} = \begin{cases} S^{q}_{j,k} & \text{if } k = 1 \text{ or } \delta_{j,k-1} = 0, \\ S^{\text{sess}}_{j,k-1} & \text{if } \delta_{j,k-1} = 1. \end{cases} \tag{8}$$

That is, the session start resets to the quantum start time of job k whenever a new session begins, and is inherited from the previous job otherwise. If either condition is violated, the current session ends and the next job starts a new one, incurring standard queue waiting times. Figure 1 illustrates these interactions for a two-iteration workflow.

Quantum processing times $q_{j,k}$ are fixed and independent of session state; sessions affect scheduling decisions and queue priority, not execution durations. The QPU executes at most one job at a time: all quantum execution intervals $[S^{q}_{j,k}, F^{q}_{j,k})$ are pairwise non-overlapping, and session reservation intervals $[F^{q}_{j,k}, S^{q}_{j,k+1})$ when $\delta_{j,k} = 1$ are also exclusive, blocking other quantum jobs during the idle gap.

In the exact CP-SAT formulation, $\delta_{j,k}$ is an explicit binary decision variable and $S^{\text{sess}}_{j,k}$ is computed automatically. In the heuristic scheduler, session continuity is determined implicitly through local checks of θ_j and γ_j at each scheduling decision.

The completion time of workflow j is:

$$C_j = S^{\text{post}}_{j,K_j} + p^{\text{post}}_{j,K_j}. \tag{9}$$

The parameters θ_j and γ_j follow the terminology of IBM Qiskit Runtime. In IBM's implementation, exceeding θ_j causes temporary deactivation rather

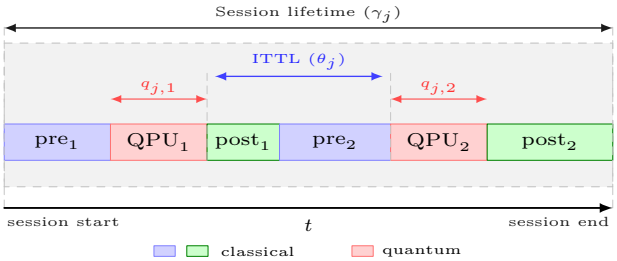


Fig. 1: Timeline of a two-iteration hybrid workflow within a session. The interactive TTL θ_j constrains the gap between consecutive quantum jobs; the session lifetime γ_j bounds the total session duration.

than immediate session closure, and a subsequent job can reactivate the session if it arrives quickly enough. Our model treats both cases as a session break for simplicity, so a violated condition always ends the current session. This is conservative: the actual session benefit in practice may be larger. The model is general: any quantum cloud service that grants priority access based on submission timing and a total session lifetime can be described using θ_j and γ_j. The scheduling problem built on this model is formulated in the next section.

IV. Scheduling Problem

Given the system model defined in Section III, the scheduler must determine the start times $S^{\text{pre}}_{j,k}$, $S^{q}_{j,k}$, and $S^{\text{post}}_{j,k}$ for all workflows $j \in \mathcal{J}$ and iterations $k \in \{1, \ldots, K_j\}$, while respecting precedence constraints, classical resource limits, quantum processor availability, and session constraints. The CP-SAT formulation implements these constraints using standard scheduling primitives: a cumulative constraint for classical resources, a disjunctive resource for the QPU, and optional interval variables to model session reservations between consecutive quantum jobs.

The objective minimises a weighted sum of total completion time and total session usage:

$$\min \sum_{j \in \mathcal{J}} C_j \; + \; \lambda \cdot U, \tag{10}$$

where $\lambda \geq 0$ is a penalty weight and U is the total session usage, defined as:

$$U = \sum_{j \in \mathcal{J}} \sum_{k=1}^{K_j - 1} \delta_{j,k} \left(S^{q}_{j,k+1} - S^{q}_{j,k} \right). \tag{11}$$

When $\delta_{j,k} = 1$, the term contributes the time span from the start of quantum job k to the start of job $k{+}1$, which includes both the quantum execution time $q_{j,k}$ and the reserved idle gap during which the QPU is blocked for other workflows. Including the execution time reflects that the full reservation window has a cost, not only the idle portion. Setting $\lambda = 0$ recovers the pure completion-time objective; larger values penalise reserved QPU idle time and encourage shorter sessions. The values of λ used in experiments are specified in Section VI.

The problem is a variant of the resource-constrained project scheduling problem with maximum time gaps (RCPSP/max) [13], [12], introduced in Section II. Classical stages correspond to activities consuming renewable resources under a cumulative capacity constraint, while quantum executions correspond to activities on a single exclusive resource. Session constraints introduce conditional maximum time gaps between consecutive quantum jobs, creating a dependency between classical and quantum decisions not present in standard RCPSP formulations.

A key difficulty arises from this dependency: classical preprocessing must finish early enough for the next quantum job to be submitted within the interactive TTL θ_j. Since CPU resources are shared, classical decisions that seem good locally may delay critical tasks, break session continuity, and increase quantum waiting times. The problem is NP-hard: it generalises RCPSP, which is already NP-hard [11], and the session variables $\delta_{j,k}$ add a combinatorial structure that grows quickly with the number of workflows and iterations. This is why the exact solver can only certify optimality for small instances, and motivates the heuristic strategies developed in Section V.

V. Scheduling Strategies

Scheduling decisions must coordinate two related processes: the allocation of classical resources to preprocessing and postprocessing tasks, and the sequencing of quantum jobs on the QPU. This dependency is reinforced by session constraints, which introduce timing dependencies between consecutive quantum executions of the same workflow. If preprocessing finishes late, a quantum job may miss its interactive TTL θ_j, breaking session continuity and increasing future waiting times.

A. Scheduling Policies

We design two CPU-level policies and three QPU-level policies. At the CPU level, *CPU-FCFS* schedules classical tasks in arrival order subject to resource constraints, reflecting standard HPC practice without session awareness. *CPU-Deadline* instead prioritises tasks by urgency: it measures the remaining time before the deadline for submitting the next quantum job within θ_j, scheduling tasks at risk of breaking a session earlier. For tasks belonging to the first iteration ($k = 1$), no session is yet active and no deadline applies; such tasks receive the lowest urgency and are effectively scheduled in FCFS order.

At the QPU level, *QPU-FCFS* serves as the session-unaware baseline, executing ready quantum jobs in arrival order. *QPU-Session* extends this by prioritising jobs that can preserve an active session. For $k \geq 2$, job (j, k) is eligible for session continuation if the inter-submission gap and the session lifetime are both satisfied at the current time:

$$S^{q}_{j,k} - F^{q}_{j,k-1} \leq \theta_j \quad \text{and} \quad F^{q}_{j,k} \leq S^{\text{sess}}_{j,k} + \gamma_j.$$

The second condition checks that the current job completes within the session lifetime, which is a sufficient condition for the session to remain valid at dispatch time. Among eligible jobs, the scheduler selects the one with the least remaining time before its interactive TTL expires, favouring workflows closest to losing their session. If no job qualifies, the policy falls back to FCFS. *QPU-Objective* extends QPU-Session further by adding the objective value to the selection score, weighting the potential increase in reserved idle time by λ. Session preference then only activates when scores are tied, so at $\lambda = 0$ the policy reduces to FCFS. *QPU-Objective+* removes this weakness by making session preference explicit and λ-independent: any job that continues an active session is always selected over a non-continuing one, regardless of ready times. Within the session-continuing set, jobs are ranked by a score combining ready time, accumulated idle time, and urgency; non-continuing jobs receive a penalty proportional to θ_j to reflect the expected re-queuing delay. This provides a direct test of whether objective-aware design can close the performance gap with purely session-aware policies.

B. Policy Combinations

We evaluate five configurations that combine one classical and one quantum policy, covering the full range from session-unaware to fully session-aware scheduling:

- *FCFS/FCFS*: no session awareness at either level. This is the baseline, reflecting standard sub-job scheduling without any session-aware logic.
- *FCFS/Session*: session-aware quantum scheduling only, isolating the benefit of prioritising session-preserving jobs.
- *Deadline/Session*: session awareness at both levels, combining deadline-driven classical priorities with session-preserving quantum scheduling.
- *Deadline/Objective*: deadline-aware classical scheduling with objective-guided quantum selection; session preference activates only when scores are tied, so the policy effectively reduces to FCFS at $\lambda = 0$.
- *Deadline/Objective+*: deadline-aware classical scheduling with the enhanced objective-aware quantum policy that explicitly favours session continuation regardless of λ.

C. Online Execution

The policies operate online: decisions are taken as tasks become ready, without advance knowledge of the full workload. At each event, a workflow arrival, a classical task completion, or a QPU job completion, the scheduler fills available CPU capacity according to the active CPU policy and, if the QPU is free, selects one quantum job from the ready queue. For CPU-Deadline, urgency is the remaining time before the next quantum submission must stay within θ_j. For QPU-Session, the ready queue is filtered to jobs that satisfy both TTL conditions immediately, and the one with the least remaining time before its TTL expires is selected; if none qualifies, the policy falls back to FCFS. This contrasts with static schedulers such as HEFT, which require all task parameters in advance and assign tasks to fixed processors. Such algorithms do not apply here because classical resources are cumulative, the QPU is a single exclusive resource shared by all workflows, and session constraints introduce conditional timing dependencies. All heuristic policies run in $O(n \log n)$ time per scheduling event, where n is the number of ready tasks, with session feasibility checks requiring only constant-time lookups. In our experiments, the total scheduling time is below one second even for the largest instances (J=160, K=16), making the overhead negligible.

VI. Experimental Setup

We evaluate the proposed model and scheduling strategies on a benchmark of synthetic instances. The parameter ranges are chosen to reflect realistic hybrid quantum–classical workloads. Processing times are consistent with observations on IBM Quantum systems, where quantum circuit execution is typically short relative to classical processing and queue waiting times [6]. The session parameters span from tight to relaxed constraints, covering the range from highly interactive workflows to the 60-second default interactive TTL of IBM Quantum Premium Plan instances [7].

A. Workload and Instance Generation

Synthetic instances reflect the workflow structure described in Section III. Table II shows a representative instance that illustrates the parameter structure before the full benchmark is described. The benchmark uses all combinations of the following parameters:

- number of workflows: 10, 20, 40, 80, 160,
- iterations per workflow: 2, 4, 8, 16,
- classical resource capacity: 4 or 8,
- session parameters (θ, γ): $(10, 40)$, $(30, 120)$, $(60, 600)$,
- random seeds: 1 to 5.

Preprocessing and postprocessing times $(p^{\text{pre}}_{j,k}, p^{\text{post}}_{j,k})$ are sampled uniformly from $[1, 5]$, quantum execution times $(q_{j,k})$ from $[4, 12]$, and classical resource demands $(r^{\text{pre}}_{j,k}, r^{\text{post}}_{j,k})$ from $[1, 4]$. Workflow arrival times are cumulative sums of inter-arrival times sampled uniformly from $[0, 2]$, producing moderate overlap between workflows. The scheduling horizon H is set automatically using a safety factor of 2.5 applied to a simple workload estimate, with a minimum of 300 time units. The resulting benchmark comprises 600 instances, ranging from small (10 workflows, 2 iterations) to large (160 workflows, 16 iterations).

Table II: Example scheduling instance with three workflows.

Workflow parameters				Iteration parameters					
j	a_j	θ_j	γ_j	k	$p^{\text{pre}}_{j,k}$	$r^{\text{pre}}_{j,k}$	$q_{j,k}$	$p^{\text{post}}_{j,k}$	$r^{\text{post}}_{j,k}$
1	0	60	600	1	3	2	8	2	1
				2	2	1	6	3	2
				3	4	2	7	2	1
2	2	60	600	1	2	1	5	2	1
				2	3	2	9	3	2
3	4	40	500	1	1	1	4	2	1
				2	2	2	6	2	2
				3	3	2	8	3	2
				4	2	1	5	2	1

B. Objective and Scheduling Configurations

All experiments minimise Equation (10), which combines total completion time $\sum_j C_j$ and total session usage U, weighted by λ. We consider two scenarios:

- Main benchmark: $\lambda = 0$, focusing on total completion time.
- Sensitivity analysis: $\lambda \in \{1, 5, 10\}$, capturing the trade-off between completion time and session usage.

We evaluate the five scheduling configurations defined in Section V: FCFS/FCFS (baseline), FCFS/Session, Deadline/Session, Deadline/Objective, and Deadline/Objective+. Performance is measured relative to the best solution found by the exact solver, so the primary question is how close each heuristic comes to the reference, not how heuristics rank among themselves.

C. Solver Configuration and Metrics

The CP-SAT solver [25] is configured with 8 threads and a time limit of 7200 seconds. The relative gap tolerance is set to $\epsilon = 0.01$. The solver uses an automatic search strategy that explores multiple solving approaches in parallel, with presolve enabled. A solution is classified as optimal if the gap is zero, near-optimal if the gap is at most ϵ, or feasible otherwise. The best solution found is used as the reference for heuristic evaluation.

Heuristic quality is measured by the percentage deviation from the reference:

$$\Delta_{\%} = 100 \cdot \frac{z^{\text{heur}} - z^{\text{ref}}}{z^{\text{ref}}}. \tag{12}$$

We also report the session success rate (SSR), defined as the fraction of consecutive quantum job pairs that successfully maintain session continuity, and the average QPU waiting time Q_{wait}. Results are summarised using medians rather than means, since completion time distributions in scheduling problems are typically right-skewed [14].

VII. Exact Solver Performance

Before evaluating heuristic strategies, we analyse the performance of the exact CP-SAT model [25] to characterise the difficulty of the scheduling problem. The solver reports a relative optimality gap: the difference between the best feasible solution found and the best lower bound when the solver stops, expressed as a percentage.

Table III: CP-SAT solver performance by number of workflows (J) and iterations per workflow (K).

J	K	#	Opt	Gap Sat	Feas	Gap (%)
10	2	30	5	25	0	0.91
	4	30	0	0	30	2.91
	8	30	0	0	30	4.59
	16	30	0	0	30	7.46
20	2	30	0	18	12	0.99
	4	30	0	0	30	3.39
	8	30	0	0	30	5.40
	16	30	0	0	30	11.98
40	2	30	0	11	19	1.23
	4	30	0	0	30	6.01
	8	30	0	0	30	8.36
	16	30	0	0	30	20.56
80	2	30	0	0	30	7.93
	4	30	0	0	30	12.04
	8	30	0	0	30	20.40
	16	30	0	0	30	34.56
160	2	30	0	0	30	14.51
	4	30	0	0	30	16.21
	8	30	0	0	30	29.87
	16	30	0	0	30	48.88

Across all 600 instances, the solver finds a feasible solution in every case, but proves optimality in only 5 and reaches the gap tolerance in 54 more. The overall median gap is 8.63%, showing that certified optimality is difficult to achieve even within the two-hour limit.

Table III reveals two clear trends. More workflows degrade performance: at $K = 2$, the median gap grows from 0.91% at $J = 10$ to 14.51% at $J = 160$, reflecting growing contention for shared CPU and QPU resources. Longer iteration chains have an even stronger effect: at $J = 160$, the gap rises from 14.51% at $K = 2$ to 48.88% at $K = 16$, and all optimal and near-optimal solutions disappear.

Both trends have a clear cause. More workflows increase resource contention, making the problem harder to solve. Longer iteration chains create sequences of time-dependent decisions connected by session constraints, which is more difficult. Iteration depth has a stronger effect than workflow count: while $K = 2$ instances admit certified solutions at small J, increasing K eliminates such guarantees even when J remains moderate. This pattern is visible in Table III: the gap grows much faster across rows (increasing K) than across groups (increasing J).

These results confirm that exact optimisation does not scale as J or K grows, even though feasible solutions are always found. This motivates scalable heuristic approaches that exploit session structure directly.

For small instances ($K = 2$), the solver gap is below 1% and the best known solution is effectively optimal. For large instances ($J = 160$, $K = 16$), the gap reaches 48.88% and the solution quality is weaker. Since all policies are evaluated against the same solution on each instance, any quality bias affects all policies equally. On large instances where the exact solver performs poorly, heuristics can occasionally produce better solutions than the reference, giving negative $\Delta_{\%}$ values. This occurs most often for session-aware policies (51 and 49 instances out of

Table IV: Heuristic performance across four values of λ. † denotes $p < 0.001$ vs. FCFS/Session (paired Wilcoxon signed-rank test, two-sided).

λ	Policy	Med. $\Delta_{\%}$	SSR	Q_{wait}
0	**FCFS/Session**	**28.75**	**0.913**	**320.3**
	Deadline/Session	29.15†	0.912	321.2
	Deadline/Objective+	42.99†	**0.913**	399.2
	Deadline/Objective	54.94†	0.788	425.9
	FCFS/FCFS	55.02†	0.788	424.6
1	**FCFS/Session**	**27.57**	**0.912**	**296.6**
	Deadline/Session	28.13†	0.911	297.4
	Deadline/Objective+	42.63†	**0.912**	366.5
	FCFS/FCFS	52.18†	0.781	392.2
	Deadline/Objective	52.31†	0.777	393.7
5	**FCFS/Session**	**12.93**	0.841	**240.0**
	Deadline/Session	13.20†	**0.843**	240.5
	Deadline/Objective+	32.39†	**0.843**	297.4
	FCFS/FCFS	46.12†	0.452	336.7
	Deadline/Objective	47.15†	0.450	337.3
10	**FCFS/Session**	**2.64**	0.842	**202.8**
	Deadline/Session	2.89†	0.844	203.2
	Deadline/Objective+	16.89†	**0.845**	266.4
	FCFS/FCFS	36.86†	0.205	330.9
	Deadline/Objective	37.05†	0.196	331.8

600 for FCFS/Session and Deadline/Session respectively), which further confirms that their advantage is not an artefact of a weak reference.

VIII. Heuristic Performance

In this section we compare the five scheduling policies against the best solution found by the exact solver. The main question is whether session-aware policies produce better schedules than session-unaware ones, and by how much. Table IV shows a clear and stable ranking across all values of λ. FCFS/Session achieves the lowest median $\Delta_{\%}$ in every configuration, followed closely by Deadline/Session. Both session-aware policies outperform all other configurations by a large and statistically significant margin ($p < 0.001$ for all comparisons, paired Wilcoxon signed-rank tests).

The observed difference between FCFS/Session and Deadline/Session is small: the median gap between the two ranges from 0.25% to 0.40% across all λ values. The key distinction is not between these two policies, but between any session-aware policy and any non-session-aware policy, where the gap exceeds 25%. Figure 2 shows the full distribution at $\lambda = 0$. Session-aware policies not only have lower medians, but also tighter interquartile ranges (19–41%). Deadline/Objective+ spreads from 27% to 73%, showing that its advantage over non-session policies is concentrated in easier instances.

Making session preference explicit in Deadline/Objective+ reduces the median deviation by 10%–20% compared to the original Deadline/Objective. This confirms that the original policy underperformed partly because its session preference was inactive at $\lambda = 0$. Still, Deadline/Objective+ is 14%–19% behind FCFS/Session across all configurations.

Figure 3 shows that this gap depends strongly on iteration depth K. At $K = 2$, Deadline/Objective+ (22.2%) nearly matches FCFS/Session (20.2%), a

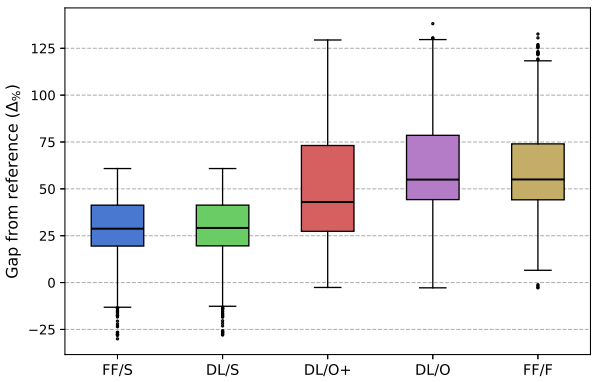


Fig. 2: Distribution of $\Delta_{\%}$ at $\lambda = 0$ across 600 instances.

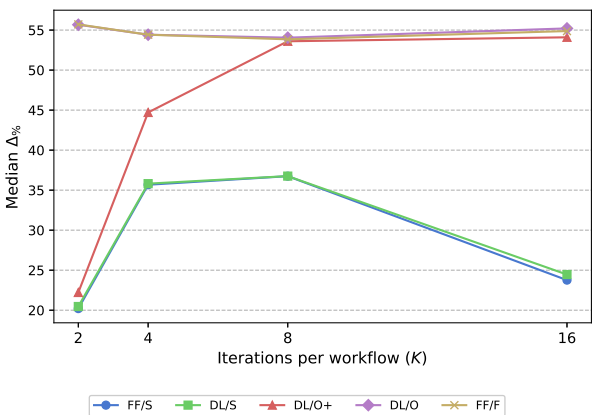


Fig. 3: Median $\Delta_{\%}$ at $\lambda = 0$ as a function of iterations per workflow K.

difference of only 2%. At $K = 16$, Deadline/Objective+ (54.1%) converges to FCFS/FCFS (54.9%), losing all advantage from its objective-aware design. Session-aware policies remain largely insensitive to K, staying between 20% and 37% across all depths. Objective-aware selection works well when sessions are short, but fails to maintain the correct global ordering as the sequence of decisions grows.

The gap between Deadline/Objective+ and FCFS/Session is not explained by session continuity: both achieve SSR ≈ 0.913 at $\lambda = 0$ and ≈ 0.843 at $\lambda = 10$. The difference shows up in QPU waiting time: Deadline/Objective+ has $Q_{\text{wait}} = 399$ versus 320 for FCFS/Session at $\lambda = 0$. The objective-aware selection chooses the right session-continuing job, but introduces idle gaps by prioritising high-urgency jobs at the expense of other ready work. Good scheduling performance therefore requires both preserving sessions and scheduling jobs efficiently within them.

FCFS/Session, the simplest session-aware policy, leads consistently across all configurations. This shows that the problem rewards session continuity over algorithmic complexity.

IX. Impact of Session Usage Penalty

We now examine how varying the session usage penalty affects the results. Figure 4 shows the median gap and SSR for penalty values of 0, 1, 5, and 10. As the penalty increases, $\Delta_{\%}$ decreases for all policies (top figure). For FCFS/Session, the median gap falls

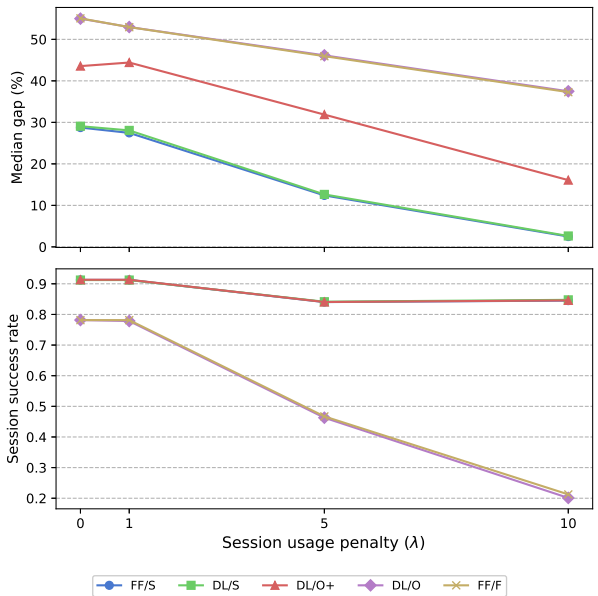


Fig. 4: Effect of the session usage penalty λ on median gap (top) and session success rate (bottom).

from 28.75% at zero penalty to 2.64% at penalty 10. This drop occurs mainly because penalising session usage makes the exact problem harder to solve: the solver must balance two competing objectives, which makes it harder to prove tight lower bounds, so the reference solutions it finds are weaker and the heuristic deviations appear smaller. The ranking of policies is stable across all penalty values. The bottom figure shows the SSR behaviour. FCFS/Session, Deadline/Session, and Deadline/Objective+ all maintain a rate of 0.84–0.91 throughout. Deadline/Objective behaves differently: its SSR drops from 0.788 at zero penalty to 0.196 at penalty 10, as it increasingly breaks sessions to avoid the QPU idle time penalty that λ imposes. Deadline/Objective+ avoids this drop through its explicit session preference. Yet the high SSR does not translate into a proportional reduction in $\Delta_{\%}$, confirming that session continuity is necessary but not sufficient for high scheduling performance.

To assess whether the session-aware advantage depends on how constrained the sessions are, we break down results at zero penalty by the three session parameter configurations. Table V summarises the results. When sessions are tight (θ=10, γ=40), FCFS/FCFS maintains session continuity only 62.9% of the time, because it frequently submits the next quantum job too late and loses priority access. This leads to a median $\Delta_{\%}$ of 91.7%, compared to 23.3% for FCFS/Session, a gap of over 68%. As sessions become more relaxed, FCFS/FCFS naturally meets the TTL more often even without any session awareness, so its SSR rises to 0.881 at θ=60 and the gap decreases to 12%. The session-aware advantage therefore depends on session tightness, but does not disappear even under relaxed constraints.

By contrast, splitting results by classical resource capacity reveals almost no effect: FCFS/Session achieves a median $\Delta_{\%}$ of 28.4% at C=4 and 29.0% at

Table V: Median $\Delta_{\%}$ and SSR for FCFS/Session and FCFS/FCFS at zero penalty, broken down by session parameters (θ, γ) and classical resource capacity C.

Grouping	Policy	Med. $\Delta_{\%}$	SSR
θ=10, γ=40	FCFS/Session	**23.3**	0.758
	FCFS/FCFS	91.7	0.629
θ=30, γ=120	FCFS/Session	**34.3**	0.981
	FCFS/FCFS	53.3	0.855
θ=60, γ=600	FCFS/Session	**36.0**	1.000
	FCFS/FCFS	48.2	0.881
C=4	FCFS/Session	**28.4**	0.912
	FCFS/FCFS	54.9	0.785
C=8	FCFS/Session	**29.0**	0.915
	FCFS/FCFS	55.2	0.792

C=8, a difference of less than 1%. The same holds for FCFS/FCFS (54.9% vs. 55.2%). Both policies face the same CPU contention at a given C, so the session-aware advantage does not depend on capacity level. What matters is how each policy responds to that contention on the QPU side: session-aware policies recover priority access that session-unaware ones lose.

X. Conclusions and Future Work

This paper studied the scheduling of hybrid classical–quantum workflows under session-based execution models. We proposed a unified formulation that combines classical resource constraints, quantum execution ordering, and session interaction requirements, and solved it with a constraint programming approach. The exact solver analysis shows that the problem is computationally challenging: optimality can be certified only for small instances, and the difficulty grows quickly with workflow length due to the timing dependencies caused by session constraints.

To address scalability, we evaluated five heuristic scheduling strategies with varying degrees of session and objective awareness. The results are clear and consistent: preserving session continuity is the key factor for achieving high-quality schedules. Session-aware policies reduce the deviation from the exact reference from over 55% to below 30% compared to session-unaware approaches, and this advantage is robust across all objective configurations and problem sizes.

A key finding comes from the Deadline/Objective+ policy, which makes session continuation explicit and independent of λ. It outperforms the original Deadline/Objective by 10%–20%, but is still 14%–19% behind FCFS/Session despite achieving the same session success rate. The difference shows up as longer QPU waiting times, showing that good scheduling requires both preserving sessions and ordering jobs efficiently within them, and that the simplest session-aware policy, FCFS/Session, is sufficient to achieve this across all configurations.

One limitation of this work is that all processing times are deterministic. In practice, classical and quantum execution times vary due to system load,

hardware calibration, and compilation differences. Since FCFS/Session makes decisions one at a time as tasks arrive, without planning ahead, it does not depend on knowing future durations. This suggests it may perform well under variable conditions, but we have not tested this and it is worth studying.

Several directions follow from this work. The most immediate is testing the policies under variable processing times and queue dynamics, and under higher CPU load than the moderate utilisation used here (around 34% at $C = 4$). Both would better stress-test the session-aware policies and clarify how much variability and contention they can tolerate. A natural complement is comparing the sub-job model studied here against the single-job allocation model, where each workflow holds its CPU resources continuously from admission to completion [23]. Under single-job allocation, classical contention cannot cause session breaks; comparing the two models would clarify when sub-job scheduling with session-aware policies is preferable and when holding resources continuously is the better strategy.

Beyond the current setting, extending the model to multi-QPU environments is important, since backend selection and load balancing add a new dimension to the scheduling problem. Finally, validating the results on real workloads, using traces from VQE or QAOA executions on cloud backends, would strengthen the relevance of the findings.

Acknowledgements

This work was supported by the Basque Government under Projects KK-2023/00012, KK-2023/00090, KK-2025/00098, and IT1504-22. It was also supported in part by Grant CNS2023-144315, funded by MICIU/AEI/10.13039/501100011033 and the European Union NextGenerationEU/PRTR, and in part by Grant PID2023-152390NB-I00, funded by MICIU/AEI/10.13039/501100011033 and FEDER.

References

[1] John Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, pp. 79, 2018.

[2] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature Communications*, vol. 5, pp. 4213, 2014.

[3] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.

[4] Martin Schulz, Martin Ruefenacht, Dieter Kranzlmüller, and Laura B. Schulz, "Accelerating HPC with quantum computing: It is a software challenge too," *Computing in Science & Engineering*, vol. 24, no. 4, pp. 60–64, 2022.

[5] Amr Elsharkawy, Xiao-Ting Michelle To, Philipp Seitz, Yanbin Chen, Yannick Stade, Manuel Geiger, Qunsheng Huang, Xiaorang Guo, Muhammad Arslan Ansari, Martin Ruefenacht, Laura Schulz, Sven Karlsson, Christian B. Mendl, Dieter Kranzlmüller, and Martin Schulz, "Challenges in HPCQC integration," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2023, vol. 2, pp. 405–406.

[6] Gokul Subramanian Ravi, Kaitlin N. Smith, Pranav Gokhale, and Frederic T. Chong, "Quantum computing in the cloud: Analyzing job and machine characteristics," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2021, pp. 39–50.

[7] IBM Quantum, "Run jobs in a session," `https://quantum.cloud.ibm.com/docs/en/guides/run-jobs-session`, 2023, Accessed 2025.

[8] Aniello Esposito, Jessica R. Jones, Stéphane Cabaniols, and David Brayford, "A hybrid classical-quantum HPC workload," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. 2023, vol. 2, pp. 117–121, IEEE.

[9] Aniello Esposito and Utz-Uwe Haus, "SLURM heterogeneous jobs for hybrid classical-quantum workflows," 2025.

[10] Andy B. Yoo, Morris A. Jette, and Mark Grondona, "SLURM: Simple Linux utility for resource management," in *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*. 2003, vol. 2862 of *Lecture Notes in Computer Science*, pp. 44–60, Springer.

[11] Rainer Kolisch and Sönke Hartmann, "Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis," *Project Scheduling*, 1999.

[12] Sönke Hartmann and Dirk Briskorn, "A survey of variants and extensions of the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 207, no. 1, pp. 1–14, 2010.

[13] M. Bartusch, R.H. Möhring, and F.J. Radermacher, "Scheduling project networks with resource constraints and time windows," *Annals of Operations Research*, vol. 16, no. 1, pp. 199–240, 1988.

[14] Dror G. Feitelson, *Workload Modeling for Computer Systems Performance Evaluation*, Cambridge University Press, 2015.

[15] David A. Lifka, "The ANL/IBM SP scheduling system," in *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*. 1995, vol. 949 of *Lecture Notes in Computer Science*, pp. 295–303, Springer.

[16] Haluk Topcuoglu, Salim Hariri, and Min-You Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[17] Yu-Kwong Kwok and Ishfaq Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, pp. 406–471, 1999.

[18] Gokul Subramanian Ravi, Kaitlin N. Smith, Prakash Murali, and Frederic T. Chong, "Adaptive job and resource management for the growing quantum cloud," in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2021, pp. 301–312.

[19] Benjamin Weder, Johanna Barzen, Frank Leymann, and Michael Zimmermann, "Hybrid quantum applications need two orchestrations in superposition: A software architecture perspective," in *Proceedings of the 18th IEEE International Conference on Web Services (ICWS)*. IEEE, 2021, pp. 1–13.

[20] Sandeep Suresh Cranganore, Vincenzo De Maio, Ivona Brandic, and Ewa Deelman, "Paving the way to hybrid quantum–classical scientific workflows," *Future Generation Computer Systems*, vol. 158, pp. 346–366, 2024.

[21] Philipp Seitz, Manuel Geiger, Christian Ufrecht, Axel Plinge, Christopher Mutschler, Daniel D. Scherer, and Christian B. Mendl, "SCIM MILQ: An HPC quantum scheduler," in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. 2024, pp. 292–298, IEEE.

[22] Sinan Pehlivanoglu et al., "Qurator: Scheduling hybrid quantum-classical workflows across heterogeneous cloud providers," 2026.

[23] IBM Quantum, "Building software for quantum-centric supercomputing," `https://www.ibm.com/quantum/blog/qcsc-software`, 2025, Accessed 2025.

[24] Paolo Viviani, Roberto Rocco, Matteo Barbieri, Gabriella Bettonte, Elisabetta Boella, Marco Cipollini, Jonathan Frassineti, Fulvio Ganz, Sara Marzella, Daniele Ottaviani, Simone Rizzo, Alberto Scionti, Chiara Vercellino, Giacomo Vitali, Olivier Terzo, Bartolomeo Montrucchio, and Daniele Gregori, "Assessing the elephant in the room in scheduling for current hybrid HPC-QC clusters," in *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2025.

[25] Laurent Perron and Vincent Furnon, "OR-Tools CP-SAT Solver," Google, `https://developers.google.com/optimization`, 2022.

Evaluación de prestaciones (3/3)

U-ATOMS: Estrategia Universal de Asignación de Hilos Basada en Transporte Óptimo para Procesadores Multinúcleo SMT

Marta Navarro[1], Vicent Pallardó-Julià[2], Lucia Pons[1], Salvador Petit[1], Maria E. Gómez[1] y Julio Sahuquillo[1]

Resumen— **Los procesadores con *Simultaneous Multithreading* (SMT) mejoran el *throughput* del sistema al compartir recursos del núcleo entre múltiples hilos. Sin embargo, esta compartición provoca una severa interferencia intranúcleo que puede degradar el tiempo de ejecución de las aplicaciones. Para mitigar este problema, otros trabajos han propuesto políticas de asignación de hilos a núcleos (*thread-to-core*, T2C). No obstante, las soluciones del estado del arte presentan limitaciones críticas: o bien se centran en un único recurso (p. ej., balanceo de accesos a la L1D), o bien utilizan modelos de regresión lineal (MRL) que requieren un costoso entrenamiento *offline* y dependen de contadores hardware fuertemente ligados a una microarquitectura específica.**

Este artículo presenta U-ATOMS, una novedosa política T2C fundamentada en la teoría del Transporte Óptimo (TO). Al utilizar los Ciclos Por Instrucción (CPI) como métrica, U-ATOMS captura implícitamente la interferencia en todos los recursos compartidos para emparejar los hilos dinámicamente. Al requerir exclusivamente contadores universales (ciclos e instrucciones), U-ATOMS es estrictamente independiente de la mezcla y de la microarquitectura, eliminando cualquier necesidad de entrenamiento.

La evaluación sobre un Intel Xeon demuestra que U-ATOMS mejora el tiempo de ejecución en un 42.3 % de media respecto a Linux en mezclas limitadas por memoria. Comparada con las políticas del estado del arte, U-ATOMS presenta 4× menos de *overhead*, consolidándose como una solución altamente escalable y viable para sistemas operativos modernos.

Palabras clave— **Procesadores SMT, Políticas de asignación de hilos a núcleos, Transporte Óptimo, Intel Xeon**

I. Introducción

LOS procesadores con *Simultaneous Multithreading* (SMT) [1] mejoran el *throughput* del procesador y la utilización de los recursos en comparación con los procesadores de un solo hilo (*single-threaded*), al permitir la ejecución concurrente de múltiples hilos en un mismo núcleo físico. Estas ventajas arquitectónicas han llevado a los principales fabricantes de la industria, como Intel, IBM, AMD y ARM, a adoptar el paradigma SMT en sus diseños comerciales.

No obstante, la compartición de la mayoría de los recursos intranúcleo (p. ej., la caché de datos L1) induce una fuerte interferencia destructiva entre las aplicaciones concurrentes. Como resultado, las prestaciones individuales de las aplicaciones pueden sufrir una degradación severa, mostrando una alta variabilidad que depende intrínsecamente del comportamiento de la aplicación coejecutada (*co-runner*). Para abordar este problema, se han propuesto políticas de asignación de hilos a núcleos (*thread-to-core*, T2C) [2], [3] diseñadas para identificar parejas de aplicaciones que presenten simbiosis y asignarlas al mismo núcleo SMT.

Algunos trabajos previos identificaron la caché de datos L1 como un cuello de botella crítico en algunos procesadores comerciales, proponiendo políticas T2C enfocadas en balancear los accesos a la caché de datos L1 [4] o a través de la jerarquía de memoria compartida [5]. Sin embargo, estas estrategias fallan ante mezclas de aplicaciones (*workloads*) heterogéneas que combinan aplicaciones limitadas por el *frontend* o el *backend*. Para superar este punto débil, otras propuestas [2], [3] modelan la interferencia entre aplicaciones en los principales recursos compartidos del núcleo (p. ej., caché de instrucciones, unidades aritméticas, caché de datos L1, ...) mediante la construcción de modelos de regresión lineal (MRL). Estos modelos difieren en los componentes utilizados para estimar la interferencia y en el procesador utilizado para el trabajo. Aunque efectivos, estos modelos presentan dos limitaciones fundamentales. Primero, dependen de contadores de prestaciones del *Performance Monitoring Unit* (PMU) altamente específicos de la arquitectura subyacente (como ocurre con el IBM POWER8 [6] o en nuestra propuesta previa para ARM [3]), lo que rompe la portabilidad de la política. Por ejemplo, el modelo desarrollado en [2] para el IBM POWER8 solo puede usarse para procesadores IBM, ya que los contadores de prestaciones utilizados no están disponibles en procesadores de otras marcas. Segundo, los coeficientes del modelo MRL se ajustan mediante un entrenamiento *offline* sobre un conjunto de *workloads* concreto. Esto liga inherentemente el modelo a la caracterización de dichos *workloads*, impidiendo su generalización ante aplicaciones con comportamientos distintos a los del entrenamiento.

Para superar estas limitaciones, este artículo propone U-ATOMS, una política T2C novedosa que captura de forma implícita la interferencia en todos los recursos compartidos–*incore* y *offcore*– utilizando los *ciclos por instrucción* (CPI) como métrica universal para emparejar aplicaciones en núcleos SMT. El enfoque propuesto aborda las dos principales limitacio-

[1]Dpto. de Informática de Sistemas y Computadores, Universitat Politècnica de València, e-mail: {marnaed,lupones,spetit,megomez,jsahuqui}@disca.upv.es
[2]TU Wien, Vienna, Austria, e-mail: vicent.pallardo.julia@tuwien.ac.at

nes de los MRLs; por un lado, U-ATOMS es independiente de la mezcla (*workload-agnostic*) y no requiere entrenamiento *offline*; por otro lado, no depende de la microarquitectura (*microarchitecture-agnostic*), ya que utiliza exclusivamente los contadores de ciclos e instrucciones, los cuales están universalmente disponibles en la PMU de cualquier procesador actual.

U-ATOMS se fundamenta en la teoría del Transporte Óptimo (TO) [7] para estimar la mejor combinación de parejas de aplicaciones. El objetivo de TO es determinar el método más eficiente para reasignar masa de una distribución a otra, minimizando una función de coste asociada; en este trabajo, esta función se define como la suma de los CPIs de las aplicaciones ejecutadas en el mismo núcleo. Esto implica encontrar la combinación de parejas que minimiza el coste total de transporte basándose en una métrica (CPI individual por aplicación). La teoría de TO ha sido ampliamente estudiada desde una perspectiva teórica [8], [9], junto con el desarrollo de métodos numéricos, lo que ha facilitado su aplicación en diversos contextos. Algunos ejemplos incluyen sistemas de telecomunicaciones [10], [11], procesamiento de imágenes y señales [12], [13] o *Machine Learning* [14], [15], entre otros. Hasta donde sabemos, este trabajo es pionero en la adopción del transporte óptimo en la arquitectura de computadores para mitigar la interferencia intranúcleo en SMT.

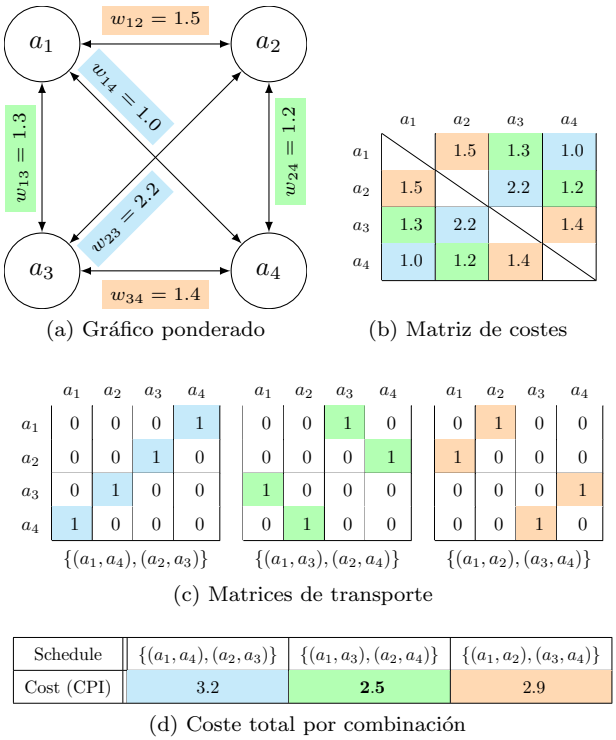
La evaluación experimental sobre una plataforma Intel Xeon demuestra que U-ATOMS supera al planificador por defecto de Linux (*Completely Fair Scheduler*) con una mejora media del 42.3 % en el tiempo de ejecución para mezclas principalmente limitadas por la jerarquía de memoria.

La principal contribución de este artículo es U-ATOMS, una política de asignación T2C que cumple tres características principales: basada en CPI, independiente de la mezcla y de la microarquitectura. Los resultados muestran que U-ATOMS supera a las políticas T2C existentes [3], [4] basadas en MRLs y reducción de contención.

II. Caracterización del Problema: Analogía entre la Asignación T2C y el Problema TO

Para ilustrar cómo la teoría del Transporte Óptimo (TO) permite resolver la asignación T2C, esta sección introduce primero el problema de TO desde su perspectiva económica original. Supongamos que una empresa tiene como objetivo transportar mercancías entre distintos países minimizando los costes de transporte. Este problema puede modelarse como un grafo ponderado: los nodos representan los países, los arcos representan las rutas de transporte y los pesos de dichos arcos corresponden a los costes. La teoría de TO determina la estrategia de transporte óptima para minimizar el coste total.

Esta idea puede trasladarse al problema de asignación T2C, donde las aplicaciones sustituyen a los países, y el coste de transporte corresponde a la métrica a minimizar–nuestra métrica de coste es el



(a) Gráfico ponderado (b) Matriz de costes

(c) Matrices de transporte

Schedule	$\{(a_1, a_4), (a_2, a_3)\}$	$\{(a_1, a_3), (a_2, a_4)\}$	$\{(a_1, a_2), (a_3, a_4)\}$
Cost (CPI)	3.2	**2.5**	2.9

(d) Coste total por combinación

Fig. 1: Ejemplo práctico de la aplicación de TO para T2C.

CPI (ciclos por instrucción). Dos aplicaciones ejecutándose en el mismo núcleo están enlazadas por un arco ponderado, donde el peso es igual a la suma del CPI de ambas aplicaciones. El objetivo es emparejar aplicaciones para minimizar la suma total de los pesos de los arcos, de forma análoga a la minimización de los costes de transporte. Cada pareja será asignada a un núcleo físico del procesador SMT.

Con fines ilustrativos, consideremos el ejemplo práctico mostrado en la Figura 1. Supongamos una mezcla de cuatro aplicaciones (a_1 a a_4) en un procesador de dos núcleos, y que la métrica objetivo a minimizar es el CPI total (es decir, la suma de los cuatro CPIs). Dado que cada aplicación puede emparejarse con cualquier otra en el mismo núcleo, representamos este escenario como un *grafo ponderado completo y no dirigido*, donde el peso entre dos aplicaciones, a_i y a_j, se denota por w_{ij} (Figura 1a). El peso del arco entre dos nodos denota la suma de los CPI de las dos aplicaciones cuando ambas comparten el mismo núcleo SMT. El grafo puede representarse como una matriz tal y como se muestra en la Figura 1b. Esta matriz se conoce como la *matriz de costes* en la teoría TO.

El planificador tiene como objetivo determinar la mejor combinación de parejas de aplicaciones, es decir, aquella que minimiza el CPI total. Dado que hay 4 aplicaciones en este ejemplo, existen 3 posibles conjuntos de parejas o *planificaciones* (*schedules*): i) $\{(a_1, a_4), (a_2, a_3)\}$, ii) $\{(a_1, a_3), (a_2, a_4)\}$, y iii) $\{(a_1, a_2), (a_3, a_4)\}$. En adelante, cada combinación puede definirse con una matriz de transporte M tal que la celda $M_{ij} = 1$ si la pareja (a_i, a_j) se considera en la planificación y $M_{ij} = 0$ en caso contrario (Figura 1c). Utilizando esta representación, defini-

mos el CPI total para la planificación dada como $CPI_M = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} M_{ij}w_{ji}$, donde w denota la matriz de costes empíricos y M la matriz de transporte de la planificación evaluada.

En este escenario reducido, es fácil identificar la mejor planificación, la cual implica emparejar (a_1, a_3) y (a_2, a_4), dando como resultado el CPI total más bajo (2.5 en la Figura 1d). Sin embargo, el número de planificaciones posibles aumenta de forma factorial con el número de aplicaciones [16], haciendo que el problema suponga un desafío. Más concretamente, para $2p$ aplicaciones, cada planificación está dada por p parejas y el número de posibles planificaciones es $(2p)!/(2^p p!)$. Por ejemplo, para 10 aplicaciones, esto resulta en 945 posibles planificaciones. En este trabajo, aprovechamos la teoría del transporte óptimo para abordar este reto.

III. Formulación Matemática

El problema del Transporte Óptimo (TO) puede formularse matemáticamente como un problema de programación lineal convexa, representando una relajación del problema combinatorio de transportar una distribución a otra [17], [18], [19]. En el contexto de este trabajo, ambas distribuciones son discretas y uniformes, cuyos nodos son el conjunto de aplicaciones. En este escenario, cada aplicación está igualmente representada tanto en la distribución de entrada como en la de salida, y el objetivo del problema TO es transportar esta distribución sobre sí misma. En términos generales, este proceso define una redistribución óptima de la masa de cada aplicación a_i sobre el conjunto de aplicaciones, sujeta a ciertas restricciones.

A continuación, se presentan algunos conceptos fundamentales para ayudar a comprender el enfoque de TO implementado en nuestra política de asignación T2C, organizados en dos subsecciones. Primero, el problema se describe formalmente y se formula matemáticamente. Segundo, se discute la optimización numérica utilizada en este artículo para resolver el problema de TO. Nos gustaría remarcar que los conceptos presentados simplifican el método TO para adaptarlo al contexto de la política de asignación T2C diseñada.

A. Formulación del Problema

Sea un grafo completo no dirigido G, cuyo conjunto de nodos se denota por $A = \{a_i\}_{i=1}^{n}$, y donde la matriz $w = \{w_{i,j}\}_{i,j=1}^{n}$ agrupa los pesos de cada arco de G. En la Figura 1a se proporciona un esquema para $n = 4$ nodos. En adelante, emplearemos a_i para referirnos a la i-ésima aplicación, y w_{ij} para denotar el coste de emparejar las aplicaciones a_i y a_j, para todo i, j.

Definimos la distribución probabilística uniforme discreta del sistema como:

$$\mu(B) = \frac{1}{n}\sum_{i=1}^{n}\delta_{a_i}(B), \quad \text{tal que} \quad \delta_{a_i}(B) = \begin{cases} 1, & \text{si } a_i \in B, \\ 0, & \text{en otro caso.} \end{cases} \tag{1}$$

donde $B \subseteq A$, y δ_{a_i} es la medida de Dirac en a_i. La medida μ asigna una masa probabilística de $1/n$ a cada nodo de G. Por lo tanto, en nuestro contexto, esto implica que cada aplicación a_i tiene una representación igual en la distribución, estableciendo un punto de partida equitativo.

En este marco, el problema TO se formula para minimizar el coste total del transporte, dado por el producto interno de las matrices, M y w, como:

$$\inf_{M\in\mathcal{S}_1} \langle M, w\rangle = \inf_{M\in\mathcal{S}_1} \sum_{i=1}^{n}\sum_{j=1}^{n} M_{ij}w_{ji}, \tag{2}$$

En la minimización anterior, M se define sobre el siguiente conjunto factible:

$$\mathcal{S}_1 := \left\{M \in [0,1]^{n\times n} \text{ tal que } M\mathbf{1} = M^T\mathbf{1} = \mathbf{1}\right\}. \tag{3}$$

donde $\mathbf{1}$ representa el vector unitario. Nótese que la definición del conjunto $\mathcal{S}_1$ garantiza la conservación de masa por aplicación. Es directo que un minimizador de (2) no es necesariamente una aplicación biyectiva (uno a uno), lo que significa que no define una planificación como esperamos en la Figura 1. Además, si restringimos $\mathcal{S}_1$ a matrices de planificación, el problema (2) dejaría de ser convexo, lo que incrementaría la complejidad de su resolución [20], [21].

Para abordar este problema y encontrar una única solución, añadimos un término cuadrático en (2) como regularizador. Gracias a este ajuste, la optimización gana estabilidad y eficiencia al encontrar una solución, como se muestra en problemas a gran escala (p. ej., [22], [23], [20]). Así, la regularización cuadrática de (2) se define como:

$$\inf_{M\in\mathcal{S}_1} \mathcal{F}(M, w) = \inf_{M\in\mathcal{S}_1} \langle M, w\rangle + \frac{\alpha}{2}\sum_{i,j} w_{ij}\|M_i - M_j\|_2^2, \tag{4}$$

donde $\mathcal{F}(M, w)$ es la nueva función objetivo, $\alpha > 0$, M_i se refiere a la i-ésima columna de M y $\|\cdot\|_2$ a la norma euclidiana en $\mathbb{R}^2$.

B. Optimización Numérica: Método de Frank-Wolfe

Se pueden utilizar distintos métodos de optimización numérica para resolver el problema TO, como el propuesto por Frank-Wolfe, basado en Gauss-Seidel y métodos de formulación proyectiva [21]. En este trabajo, utilizamos el primero, que es el método estándar y resulta eficiente para resolver el problema cuadrático [20].

El método de Frank-Wolfe [24] es un algoritmo iterativo de gradiente de primer orden diseñado específicamente para la optimización convexa con restricciones [25], cuya eficiencia para resolver (4) está garantizada mediante una técnica de punto interior [26].

Este método consiste en un algoritmo que implementa un bucle principal de dos pasos, el cual itera hasta que la matriz de transporte M converge:

$$1^{er}\ paso: \quad \tilde{M}^k \in \arg\min_{\tilde{M}\in\mathcal{S}_1} \langle \nabla\mathcal{F}(M^k, w), \tilde{M}\rangle, \quad (5)$$

$$2^{do}\ paso: \quad M^{k+1} \leftarrow (1-\tau_k)M^k + \tau_k\tilde{M}^k. \quad (6)$$

El primer paso obtiene una matriz intermedia para definir la dirección en la que minimizar la función objetivo $\mathcal{F}$, donde k indica la iteración y $\nabla\mathcal{F}(M^k, w)$ denota una matriz cuyas componentes se definen como:

$$(\nabla\mathcal{F}(M^k, w))_{ij} = w_{ij} + \alpha\sum_{j=1}^{n} w_{ij}(M^k_{ij} - M^k_{jj}). \quad (7)$$

El segundo paso (véase la Ecuación (6)) utiliza la salida del primer paso ($\tilde{M}^k$) para obtener una matriz de transporte M^{k+1} para la siguiente iteración. Esto se realiza utilizando la variable de tamaño de paso (*step size*) τ correspondiente a la k-ésima iteración (τ_k).

IV. La Política U-ATOMS

Esta sección detalla la implementación de U-ATOMS, organizada en cuatro partes principales: i) aplicación de la estrategia, ii) llenado inicial de la matriz, iii) implementación de TO propuesta, y iv) cómo procede U-ATOMS cuando una nueva aplicación llega al sistema.

Aplicación de U-ATOMS. Este proceso implica monitorizar los contadores de prestaciones de ciclos (`cycles`) e instrucciones (`instructions`), actualizar los costes de la matriz, aplicar TO para identificar las mejores parejas y asignar estos pares a los núcleos. En los experimentos, la estrategia se ejecuta al final de cada intervalo de ejecución de 100 ms. Antes de aplicar U-ATOMS por primera vez, la matriz de costes debe llenarse tal y como se detalla a continuación.

Llenado Inicial de la Matriz. Dado que la matriz de costes está vacía al comienzo de la ejecución de la mezcla, se dedican unos pocos intervalos (*quantums*) a llenar la matriz en lo que denominamos *fase de calentamiento*. Por ejemplo, supongamos que el procesador ejecuta 8 aplicaciones (cuatro parejas). Entonces, el número de celdas en la matriz es 64 (8x8), 56 excluyendo la diagonal principal. Dado que la matriz es simétrica (es decir, el valor de la celda (i, j) coincide con el valor de la celda (j, i)), solo es necesario medir 28 valores diferentes. Por lo tanto, se necesitan 7 intervalos para llenar la matriz, ya que en cada intervalo se llenan 4 celdas (cuatro parejas en ejecución). En general, para n aplicaciones, el número de celdas a llenar viene dado por $\binom{n}{2}$, lo cual, dividido por el número de parejas, da el número de intervalos de la fase de calentamiento. Para garantizar que las celdas diagonales de la matriz de costes (p. ej., w_{ij} donde $i = j$) nunca se seleccionen, ya que una aplicación no puede emparejarse consigo misma, estas celdas se rellenan con un valor sustancialmente alto, evitando que se consideren como opciones de emparejamiento viables.

Implementación de la propuesta TO. La optimización TO implementada consta de dos bucles compuestos por los dos pasos de optimización numérica discutidos en la Sección III-B. En el primer paso, utilizamos el algoritmo *Fast Network Simplex* presentado por Bonneel et al. en [27] para calcular la matriz intermedia $\tilde{M}^k$, la cual se utiliza luego para definir el gradiente $\Delta M^k = \tilde{M}^k - M^k$. Este gradiente es necesario para obtener la dirección en la que minimizar la función objetivo $\mathcal{F}$ y para configurar la búsqueda lineal (*line search*) sobre τ_k en el segundo paso. Respecto al segundo paso, implementamos el algoritmo de búsqueda lineal de Armijo [28] para determinar τ_k. Este algoritmo encuentra un τ_k óptimo para garantizar un progreso eficiente hacia la minimización de la función objetivo. Una vez que se obtiene τ_k, se utiliza para calcular la matriz de transporte para la siguiente iteración, M^{k+1}. Estos pasos se repiten hasta que se cumplen los criterios de estabilización, lo cual ocurre cuando se satisface al menos una de dos condiciones. La primera condición comprueba si la diferencia absoluta entre las matrices M^k y M^{k-1} es lo suficientemente pequeña (p. ej., inferior a 10^{-6}), mientras que la segunda se centra en la diferencia relativa. En particular, calculamos si $\frac{|M^{k-1}-M^k|}{|M^k|+10^{-9}}$ es inferior a 10^{-9}. Tanto los límites absolutos como los relativos se establecen de acuerdo con los valores por defecto en implementaciones de optimización típicas[1].

Llegada de nuevas aplicaciones al sistema. Siempre que una aplicación llega al sistema, la matriz de costes debe actualizarse con la información sobre la aplicación entrante. En este caso, solo es necesario actualizar las celdas de la matriz correspondientes a los pares potenciales que involucran a la nueva aplicación. Para este propósito, la aplicación se empareja con todas las demás de forma *round-robin* durante los siguientes intervalos. Nótese que U-ATOMS es independiente de la mezcla; es decir, funciona a pesar de que la aplicación que entra al sistema sea una aplicación no vista previamente (sin necesidad de volver a entrenar). En tal caso, esta situación no podría ser gestionada por un enfoque basado en MRL.

V. Entorno Experimental y Metodología

A. Plataforma

Hemos implementado U-ATOMS en un Intel Xeon Silver 4210R [29] basado en la microarquitectura Cascade Lake [30], con una memoria principal DRAM de 32GB. Las capacidades de la caché de instrucciones L1, la caché de datos L1, la L2 y la L3 son de 32KB, 32KB, 10MB y 13.8MB, respectivamente. El sistema ejecuta una versión de Linux 5.4.0-99-generic. La política se implementó en un *framework* interno desarrollado para estudios de evaluación de presaciones por nuestro grupo. Desde este *framework*

[1]Por ejemplo, véase `https://pythonot.github.io/gen_modules/ot.optim.html#ot.optim.cg`.

se lanzan las mezclas de aplicaciones a ejecutar, se utiliza la herramienta `Perf` [31] para configurar los contadores de prestaciones con los eventos hardware seleccionados a medir, recopilando los resultados durante la ejecución de la mezcla. El *framework* lee los contadores de prestaciones en cada intervalo de ejecución definido, donde se aplica U-ATOMS. Se utiliza la llamada al sistema `sched_affinity` [32] para asignar las aplicaciones a los núcleos.

B. Metodología de Evaluación

Para evaluar las políticas T2C, utilizamos mezclas que incluyen aplicaciones con diferentes características desde la perspectiva del consumo de recursos. Nótese que si las aplicaciones presentan un comportamiento homogéneo, ninguna estrategia de planificación podría proporcionar mejoras de prestaciones debido al poco margen de mejora.

B.1 Diseño de las Mezclas

Hemos utilizado mezclas de aplicaciones compuestas por 8 aplicaciones de las suites SPEC CPU2006 [33] y CPU2017 [34]. Para diseñar mezclas con comportamiento heterogéneo, primero caracterizamos el comportamiento estático de cada aplicación ejecutándose en solitario. Debido a las características de la PMU de Intel, esta caracterización se realizó en la etapa de ejecución (*etapa de issue*). Más concretamente, en esta etapa, calculamos la fracción de *issue slots* donde se emitieron instrucciones y la fracción de slots donde no se emitió ninguna instrucción debido a bloqueos (*stalls*) causados por componentes críticos del *frontend* o del *backend*. En este sentido, desglosamos los stalls debidos al *backend*, tales como las cachés L1, L2 y L3, así como la memoria principal y los componentes del núcleo. La categoría *core* se refiere a los stalls causados por riesgos estructurales debidos a los operadores aritméticos.

La Figura 2 presenta los resultados para las 42 aplicaciones ejecutadas en solitario durante 60 segundos. Se puede apreciar una amplia variedad de comportamientos entre las aplicaciones. De acuerdo con estos resultados, las aplicaciones pueden clasificarse en 3 grupos principales dependiendo del componente particular que causa una alta fracción de los stalls: la caché de datos L1 (L1), el *Frontend* (FE) y la memoria principal (MM). Podemos ver que los valores de los componentes MM y FE presentan el rango de variación más amplio. Para propósitos experimentales, asumimos que una aplicación pertenece a los grupos MM, FE y L1 si la fracción de stalls para ese componente es mayor que 22 %, 20 % y 5 %, respectivamente.

Las mezclas de 8 aplicaciones se construyeron aleatoriamente basándose en el criterio de que cada mezcla debía contener al menos una aplicación del grupo FE, una del grupo MM y tres del grupo L1. La razón por la que el número de aplicaciones del grupo L1 es tres se debe a su enorme impacto en las prestaciones. Nótese que esta caché introduce una alta interferencia intranúcleo a pesar de su tamaño

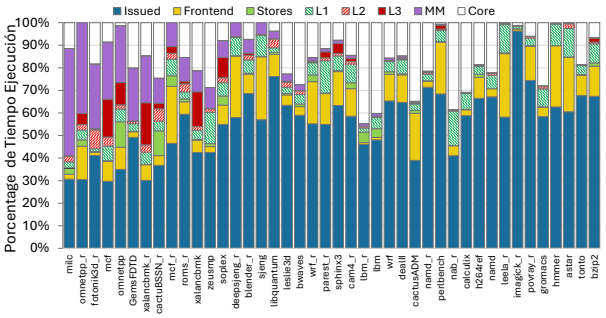


Fig. 2: Caracterización de las aplicaciones en ejecución de un solo hilo (ST).

relativamente pequeño en ejecución individual. La aplicación de este criterio da como resultado 5 tipos de mezcla de 8 aplicaciones, denotados por la terna (xL1,yFE,zMM). Las ternas específicas son: (3L1,3FE,2MM), (3L1,3FE,2MM), (3L1,2FE,3MM), (4L1,1FE,3MM) y (4L1,2FE,2MM), a las cuales nos referiremos como tipo de mezcla [a], [b], [c], [d] y [e], respectivamente. Construimos seis mezclas aleatorias para cada tipo, dando un total de 30 mezclas (`c0-c29`).

B.2 Metodología de Medida de la Mezcla

Como se mencionó anteriormente, para la caracterización individual, cada aplicación se ejecutó en solitario durante 60 segundos. El número de instrucciones finalizadas durante la ejecución se considera como el número objetivo para cada aplicación. Al ejecutar una mezcla compuesta por 8 aplicaciones, leemos los contadores de prestaciones en cada intervalo de tiempo de 100 ms. Debido a la interferencia, cada ejecución requiere un número diferente de intervalos para completarse, por lo tanto, cada aplicación se ejecuta hasta que se alcanza el número objetivo. La mezcla se mantiene en ejecución hasta que la aplicación más lenta alcanza su número objetivo de instrucciones; es decir, las aplicaciones que terminan antes se relanzan para mantener un número constante de aplicaciones en ejecución.

VI. Evaluación de Prestaciones

A. Políticas Evaluadas

Esta sección compara U-ATOMS frente a SYNPA [3], una política reciente basada en modelos de regresión lineal (MRL), y el enfoque Balancing L1D [4] (BaL1D). Estas políticas se resumen a continuación para hacer que el artículo sea autocontenido. Se pueden encontrar más detalles en los artículos originales.

SYNPA, originalmente diseñada para contadores de prestaciones específicos de ARM, fue adaptada para el procesador Intel utilizado en este trabajo, dando como resultado $SYNPA3_{intel}$. La política adaptada contabiliza las instrucciones emitidas (*issued*) y los bloqueos (*stalls*) en la etapa de *issue* en lugar de en la etapa de *dispatch* como se hace en los procesadores ARM [3]. Los valores recopilados se utilizan para caracterizar el comportamiento de la aplicación. Para este fin, los *issue slots* se agrupan en tres categorías

principales dependiendo de si se emiten instrucciones, si no se emite ninguna instrucción debido a un bloqueo causado por el *frontend* del núcleo (p. ej., un fallo en la caché de instrucciones L1), o si surge un bloqueo en el *backend* del núcleo (p. ej., un fallo en la caché de datos L1). Esta política construye un modelo de regresión lineal, basado en las tres categorías, que se utiliza para predecir las prestaciones de las distintas parejas y elegir la mejor combinación. Se requiere un entrenamiento *offline* para obtener los coeficientes del modelo.

BaL1D balancea los accesos entre las cachés de datos L1 de todos los núcleos con el objetivo de minimizar la contención en este recurso, la cual los autores demuestran que es crítica en la mayoría de las mezclas. Para este fin, el enfoque empareja las aplicaciones con la cantidad de accesos más altos y más bajos a la caché de datos L1, la segunda con más accesos con la segunda con menos, y así sucesivamente. En este trabajo, utilizamos contadores equivalentes al enfoque original, ya que los mismos contadores no están disponibles en nuestro procesador.

B. Evaluación de prestaciones

Para evaluar U-ATOMS, $SYNPA3_{intel}$ y BaL1D, comparamos sus prestaciones en términos de tiempo de ejecución (*Turnaround Time*, TT) y sobrecarga (*overhead*). La Figura 3 presenta los resultados frente al planificador de Linux. En general, U-ATOMS supera a $SYNPA3_{intel}$ y a BaL1D en todos los tipos de mezclas. Las tres políticas rinden de manera similar en mezclas con menos aplicaciones dominadas por el *backend* (es decir, los tipos de mezcla [a] y [b], que incluyen mezclas o *combinaciones* de `c0` a `c11`), pero las diferencias se vuelven más pronunciadas cuando crece el número de aplicaciones dependientes del *backend*.

Notablemente, $SYNPA3_{intel}$ presenta caídas de prestaciones significativas (p. ej., de un 15 % en el TT en `c27`) debido a su dependencia de coeficientes de regresión preentrenados, lo que limita su adaptabilidad. BaL1D, que no necesita entrenamiento, depende de las características de la mezcla, logrando mejoras en el TT de apenas un 7.2 % y un 3.2 % en los tipos de mezcla [a] y [b], respectivamente.

Por el contrario, U-ATOMS actualiza dinámicamente las decisiones de planificación en tiempo de ejecución en cada intervalo, basándose únicamente en valores obtenidos durante la ejecución. Esta adaptabilidad lo hace particularmente más efectivo para mezclas con recursos limitados (tipos [c], [d] y [e]). Por ejemplo, en la mezcla `c21` (tipo [d]), U-ATOMS logra un *speedup* notable de alrededor del 60 % en términos de TT.

C. Comparativa del Tiempo de Overhead.

Para evaluar el tiempo de *overhead*, el tiempo total empleado por cada política se desglosó en tres componentes principales: i) *preprocesamiento*, que consiste en leer los contadores de prestaciones y preparar los valores para el algoritmo (p. ej., obtener las categorías y normalizar los valores) si es necesario; ii) *algoritmo*, es decir, aplicar el algoritmo en sí y almacenar los resultados en la estructura correspondiente (p. ej., un *array* o un grafo); y iii) *asignación*, que consiste en leer la estructura, seleccionar las parejas y asignarlos en los núcleos. La Tabla I presenta los resultados para las tres políticas estudiadas.

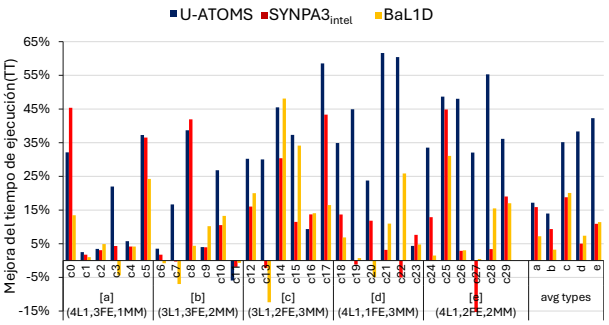


Fig. 3: Mejora de TT de U-ATOMS, $SYNPA3_{intel}$ y BaL1D frente a Linux.

Políticas	Overhead (ms)			
	Preproc.	Algoritmo	Asignac.	Total (%)
U-ATOMS	1.19	0.26	0.86	2.31 %
$SYNPA3_{intel}$	2.66	0.88+0.17	0.66	4.37 %
BaL1D	0.20	0.10	0.67	0.97 %

Tabla I: Comparativa del *overhead* de las políticas estudiadas.

Una observación interesante es que el *preprocesamiento* es el componente que más contribuye al tiempo de overhead. La razón principal es el procesamiento requerido que $SYNPA3_{intel}$ necesita en términos de rellenar y preparar las estructuras antes de aplicar el algoritmo. Sin embargo, U-ATOMS solo requiere actualizar la matriz de transporte.

De forma contraintuitiva, aplicar el algoritmo en sí mismo incurre en mucho menos *overhead* que todo el preprocesamiento. En este sentido, U-ATOMS presenta 4× menos *overhead* en el componente del algoritmo (es decir, 0.26 ms frente a 1.05 ms) que $SYNPA3_{intel}$. La razón es doble: por un lado, la matriz de transporte en U-ATOMS converge en solo dos iteraciones del bucle en casi todos los intervalos; por otro lado, $SYNPA3_{intel}$ requiere tres acciones principales muy costosas (aplicar el modelo inverso, aplicar el modelo directo y aplicar el algoritmo de Blossom). Respecto a BaL1D, U-ATOMS incurre en el doble de su *overhead*, ya que el primero solo requiere balancear los accesos a L1, lo que lo hace más simple.

VII. Trabajo Relacionado

Algunos trabajos previos [4] identificaron la caché de datos L1 como uno de los principales recursos de contención intranúcleo en los procesadores actuales. Dado que esta caché actúa como un importante cuello de botella de prestaciones, los autores propusieron balancear los accesos entre las cachés de datos L1 de todos los núcleos SMT. La ventaja principal de este enfoque es que es independiente de la mezcla, y solo requiere de dos contadores de prestaciones (accesos a L1 y ciclos).

Los modelos de regresión lineal se han aplicado ampliamente en el pasado para reducir la interferencia entre aplicaciones en procesadores SMT. Moseley et al. [35] aplican modelado lineal y particionamiento recursivo para predecir los incrementos de prestaciones al ejecutar dos aplicaciones simultáneamente. Radojkovic et al. [36] presentan un método de inferencia estadística para optimizar la asignación de tareas. Eyerman y Eeckhout [37] presentan un modelo analítico que predice la degradación de prestaciones de las aplicaciones utilizando pilas de prestaciones (*performance stacks*), aunque requiere de contadores hardware personalizados no disponibles en procesadores comerciales, lo que limita su aplicabilidad. Para abordar esta limitación, Feliu et al. [2], [38] utilizan la pila de CPI del IBM POWER8 para construir un modelo de regresión lineal con el fin de planificar las aplicaciones. De forma similar, Navarro et al. [3] presentan SYNPA, una política de asignación de hilos a núcleos para procesadores SMT de ARM para minimizar la interferencia intranúcleo y mejorar las prestaciones de las aplicaciones. Utiliza la pila de Ciclos de Instrucciones y Stalls (*Instructions and Stalls Cycles*, ISC) para modelar la distribución de ciclos de ejecución en procesadores SMT y emplea un modelo de regresión lineal para obtener la predicción de las degradaciones de prestaciones, la cual se utiliza para seleccionar las parejas simbióticas. Aunque estos últimos modelos son aplicables a procesadores reales, los enfoques existentes están restringidos por dos grandes limitaciones: la dependencia de contadores hardware concretos ligados a procesadores específicos y una efectividad limitada exclusivamente a las mezclas utilizadas para obtener los coeficientes del modelo.

VIII. Conclusiones

Las políticas de asignación T2C existentes basadas en la estimación de interferencias para predecir las prestaciones de parejas de aplicaciones se basan en modelos de regresión lineal. En este artículo, hemos propuesto U-ATOMS, que aborda las principales debilidades de los enfoques basados en MRL: entrenamiento *offline* del modelo, dependencia de la mezcla y dependencia de la microarquitectura. U-ATOMS aborda las primeras por diseño, gracias a la aplicación del problema de Transporte Óptimo (TO) para seleccionar las mejores combinaciones de parejas de aplicaciones. Este enfoque solo requiere trabajar con matrices cuyos datos se recopilan únicamente en tiempo de ejecución y aplicar dos funciones matemáticas, *Network Simplex* y Armijo, lo que hace que U-ATOMS sea independiente de la mezcla. Por otro lado, U-ATOMS persigue minimizar la suma de los CPI de las parejas de aplicaciones que se ejecutan en cada núcleo, requiriendo como entrada únicamente dos contadores de prestaciones disponibles universalmente, instrucciones y ciclos, comunes a todos los procesadores modernos, lo que hace a U-ATOMS independiente de la microarquitectura.

Los resultados muestran que U-ATOMS mejora significativamente a $SYNPA3_{intel}$ y BaL1D. En promedio, U-ATOMS logra mejoras de prestaciones sobre Linux 4× mayores que los enfoques estudiados. Además, esto se consigue con casi la mitad de *overhead* que los trabajos basados en MRL. Por último, nos gustaría enfatizar que las características de U-ATOMS lo convierten en una política muy adecuada para ser implementada en sistemas operativos modernos implementados en procesadores SMT.

Agradecimientos

Este trabajo ha sido financiado por MICIU/AEI/10.13039/501100011033/ FEDER, UE, a través de los proyectos PID2024-158682OB-C31 y PID2021-123627OB-C51, junto con la Generalitat Valenciana a través del proyecto CIPROM/2024/7. Marta Navarro tiene la beca de Subvenciones para la contratación de personal investigador predoctoral de CIACIF/2021/413.

Referencias

[1] David M. Tullsen et al., "Simultaneous multithreading: Maximizing on-chip parallelism," ISCA '95, p. 392–403.
[2] Josué Feliu et al., "Symbiotic job scheduling on the IBM POWER8," in *HPCA*, 2016, pp. 669–680.
[3] Marta Navarro et al., "SYNPA: SMT performance analysis and allocation of threads to cores in ARM processors," in *IPDPS*, 2024, pp. 705–715.
[4] Josué Feliu et al., "L1-bandwidth aware thread allocation in multicore smt processors," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, 2013, pp. 123–132.
[5] Josué Feliu et al., "Understanding cache hierarchy contention in cmps to improve job scheduling," in *IEEE 26th IPDPS*, 2012, pp. 508–519.
[6] B. Balaram et al., "IBM POWER8 processor core microarchitecture," *IBM Journal of Research and Development*, vol. 59, no. 1, pp. 2:1–2:21, 2015.
[7] Gaspard Monge, "Mémoire sur la théorie des déblais et des remblais," *Mem. Math. Phys. Acad. Royale Sci.*, pp. 666–704, 1781.
[8] Y. Brenier, "Polar factorization and monotone rearrangement of vector-valued functions," *Communications on Pure and Applied Mathematics*, vol. 44, no. 4, pp. 375–417, 1991.
[9] W. Gangbo and R. J. McCann, "The geometry of optimal transportation," *Acta Mathematica*, vol. 177, no. 2, pp. 113–161, 1996.
[10] V. I. Oliker, "Near radially symmetric solutions of an inverse problem in geometric optics," *Inverse Problems*, vol. 3, pp. 743–756, 1987.
[11] X.-J. Wang, "On the design of a reflector antenna," *Inverse Problems*, vol. 12, pp. 351–375, 1996.
[12] P. Li et al., "A novel earth mover's distance methodology for image matching with gaussian mixture models," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 1689–1696.
[13] S. Kolouri et al., "The radon cumulative distribution transform and its application to image classification," *IEEE Transactions on Image Processing*, vol. 25, no. 2, pp. 920–934, 2016.
[14] S. Kolouri et al., "Optimal mass transport: Signal processing and machine-learning applications," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 43–59, 2017.
[15] Eduardo Fernandes Montesuma et al., "Recent advances in optimal transport for machine learning," *arXiv preprint arXiv:2306.16156*, 2023.
[16] Christos H Papadimitriou and Kenneth Steiglitz, *Combinatorial optimization, algorithms and complexity*, Prentice Hall, 1982.
[17] Gabriel Peyré and Marco Cuturi, "Computational optimal transport: With applications to data science," *Foundations and Trends in Machine Learning*, vol. 11, no. 5-6, pp. 355–607, 2019.
[18] L. Ambrosio and N. Gigli, *Modelling and Optimisation of*

Flows on Networks, chapter A User's Guide to Optimal Transport, Springer, 2013.
[19] Filippo Santambrogio, *Optimal transport for applied mathematicians*, vol. 55, Birkhäuser Cham, 2015.
[20] S. Ferradans et al., "Regularized discrete optimal transport," *SIAM J. Imaging Sci.*, vol. 7, no. 3, pp. 1853–1882, 2014.
[21] D. A. Lorenz et al., "Quadratically regularized optimal transport," *Appl. Math. Optim.*, vol. 33, pp. 1919–1949, 2021.
[22] *Lightspeed computation of optimal transport*, 2013.
[23] M. Essid and J.M. Solomon, "Quadratically regularized optimal transport on graphs," *SIAM J. Sci. Comput.*, vol. 40, no. 4, pp. A1961–A1986, 2018.
[24] M. Frank and P. Wolfe, "An algorithm for quadratic programming," *Nav. Res. Logist. Q.*, vol. 3, no. 1, pp. 95–110, 1956.
[25] M. Zaslavskiy et al., "A path following algorithm for the graph matching problem," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, pp. 2227–2242, 2009.
[26] Y. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Methods in Convex Programming*, vol. 13 of *SIAM Stud. Appl. Math.*, SIAM, 1994.
[27] Nicolas Bonneel et al., "Displacement interpolation using lagrangian mass transport," *ACM Trans. Graph.*, vol. 30, no. 6, pp. 1–12, Dec. 2011.
[28] L. Armijo, "Minimization of functions having lipschitz continuous first partial derivatives," *Pacific J. Math.*, vol. 16, no. 1, pp. 1–3, 1966.
[29] "Intel Xeon Silver 4210R," `https://en.wikichip.org/wiki/intel/xeon_silver/4210r`, Accessed: 2025-03-12.
[30] "Cascade Lake - Microarchitectures - Intel," `https://en.wikichip.org/wiki/intel/microarchitectures/cascade_lake`, Accessed: 2025-03-12.
[31] "Perf: Linux profiling with performance counters," 2023, Accessed: 2025-03-12.
[32] "sched_seaffinity(2) - Linux manual page," 2023, Accessed: 2025-03-12.
[33] John L Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.
[34] Ankur Limaye et al., "A workload characterization of the spec cpu2017 benchmark suite," in *ISPASS*, 2018, pp. 149–158.
[35] T Moseley et al., "Methods for modeling resource contention on simultaneous multithreading processors," in *International Conference on Computer Design: VLSI in Computers and Processors*, 2005, pp. 373–380.
[36] Petar Radojković et al., "Optimal task assignment in multithreaded processors: A statistical approach," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012, pp. 235–248.
[37] Stijn Eyerman and Lieven Eeckhout, "Probabilistic Job Symbiosis Modeling for SMT Processor Scheduling," in *ASPLOS*, 2010, pp. 91–102.
[38] Josué Feliu et al., "Improving IBM POWER8 Performance Through Symbiotic Job Scheduling," *IEEE TPDS*, vol. 28, no. 10, pp. 2838–2851, 2017.

Espera activa de bajo consumo en sistemas de tiempo real: caracterización de TPAUSE sobre hardware COTS

Álex Gracia[1], Héctor Blanco-Alcaine[2], Zoltan Fodor[2], David Zage[2], José Luis Briz[1], Juan Segarra[1]

Resumen— **En sistemas de tiempo real estricto, eludir la latencia del planificador tradicionalmente requiere espera activa (`rdtscp`), maximizando la precisión temporal a costa de un alto consumo energético. Este artículo evalúa la instrucción `TPAUSE` (Intel `WAITPKG`) como alternativa de retardo en espacio de usuario sobre Linux `PREEMPT_RT`. Demostramos que `TPAUSE` (modo C0.1) mantiene un *jitter* competitivo en reposo y reduce la potencia dinámica más del 35 % para esperas mayores a 2 µs. Bajo contención extrema, `TPAUSE` ofrece mayor estabilidad temporal: al detener el *pipeline* y evita la penalización por saturación de recursos compartidos que aparece en el peor caso con espera activa. Así, `TPAUSE` resulta óptimo para equilibrar precisión por debajo del microsegundo y eficiencia.**

Palabras clave— **Sistemas de tiempo real, espera activa, `TPAUSE`, determinismo temporal, *jitter*, eficiencia energética, `PREEMPT_RT`.**

I. Introducción

La creciente complejidad de los sistemas críticos en sectores como la automoción o la Industria 4.0 ha consolidado el uso de plataformas multinúcleo comerciales gestionadas mediante GNU/Linux, impulsado por el parche `PREEMPT_RT` [1], [2]. Generalmente dichas plataformas están configuradas para optimizar el rendimiento en lugar de la estabilidad temporal necesaria en sistemas de tiempo real. Sin embargo, la contención en recursos compartidos y los mecanismos de ahorro de energía introducen variabilidades temporales (*jitter*) imprevisibles, que requieren caracterización empírica.

Así mismo, los hilos de alta prioridad en aplicaciones de tiempo real normalmente suspenden su ejecución utilizando temporizadores de alta resolución, como `hr_timmers` [3]. Con todo, la precisión de estos mecanismos resulta insuficiente para escenarios que exigen restricciones por debajo del microsegundo. A esta escala, la sobrecarga del subsistema de temporizadores del kernel y los cambios de contexto introducen desviaciones estocásticas excesivas [4], limitando el uso de primitivas de bloqueo convencionales.

Para eludir la latencia del planificador y garantizar una activación inmediata, los diseñadores recurren a técnicas de espera activa (*busy-wait*), reteniendo la CPU [5], [6]. Esta estrategia maximiza la estabilidad temporal, pero conlleva un coste elevado en términos de consumo energético y eficiencia del sistema.

[1]Departamento de Informática e Ingeniería de Sistemas - I3A, Universidad de Zaragoza, e-mail: {`alex.gracia, briz, jsegarra`}`@unizar.es`

[2]Intel Corporation, e-mail: {`hector.blanco.alcaine, zoltan.fodor, david.zage`}`@intel.com`

Para implementar la espera activa existen instrucciones como `MWAIT`, restringida a modo privilegiado (anillo 0). Para balancear accesibilidad y eficiencia, Intel introdujo las extensiones `WAITPKG` en la ISA, que incluyen `TPAUSE` y `UMWAIT`, permitiendo al espacio de usuario transitar a estados de bajo consumo.

Esta contribución analiza estas estrategias de espera activa disponibles en arquitecturas x86 contemporáneas, caracterizando y comparando el muestreo tradicional de alta resolución por *software* (mediante `rdtscp`) con el soportado por esta nueva generación de mecanismos, específicamente la instrucción `TPAUSE`. Se evalúan empíricamente compromisos entre precisión temporal y huella energética en sistemas de tiempo real estricto.

El resto de este artículo se estructura de la siguiente manera: la sec. II revisa el trabajo relacionado; la sec. III describe los mecanismos evaluados; la sec. IV detalla la metodología experimental; la sec. V analiza los resultados empíricos y, finalmente, la sec. VI expone las conclusiones y el trabajo futuro.

II. Trabajos relacionados

Esta sección revisa las principales técnicas para caracterizar latencias en Linux y la evolución de los mecanismos de espera del procesador.

La variabilidad temporal en plataformas *commercial off-the-shelf* (COTS) de tiempo real viene realizándose con metodologías como la utilizada en [7], apoyadas en herramientas como `cyclictest` [8], [9]. Trabajos como [1] han demostrado que eventos opacos del *hardware*, como las Interrupciones de Gestión del Sistema (SMIs), afectan al tiempo de ejecución y requieren soluciones específicas.

Es habitual maximizar la precisión temporal mediante el muestreo continuo del *contador de marcas de tiempo* (TSC) vía `rdtsc`/`rdtscp` [10], a costa de mantener el núcleo en estado energético C0 ACPI [11]. Esto último maximiza el consumo dinámico, con riesgo de caída del rendimiento por efecto de la regulación térmica (*thermal throttling*) [12]. Históricamente, las opciones para mitigar este coste energético a nivel de instrucción han sido escasas. La instrucción `PAUSE` se introdujo con el objetivo de reducir las penalizaciones de pipeline, añadiendo un pequeño retardo que modera la ejecución especulativa descontrolada. De este modo, se evitan los vaciados de pipeline (*pipeline flushes*) al salir del bucle de espera y se liberan recursos para otros hilos lógicos

en sistemas con *Hyper-Threading* [13]. No obstante, dicho retardo es demasiado breve como para producir un ahorro energético apreciable [14]. Las instrucciones `MONITOR` y `MWAIT` permiten descender a *estados C* de ahorro energético más profundos (p.e., C1), reduciendo el consumo drásticamente [15], pero ambas instrucciones se ejecutan en modo privilegiado, impidiendo su uso directo en aplicaciones de tiempo real en espacio de usuario e introduciendo *latencias de activación* que dependen de la profundidad del estado C [16]. Recientemente se ha demostrado que estas instrucciones reducen el consumo energético entre un 20 % y 70 % frente al *busy-wait* heredado en procesadores heterogéneos modernos [17]. Sin embargo, la metodología seguida en [17]se centra exclusivamente en el rendimiento medio (*throughput*) y filtra los valores extremos para estabilizar las mediciones. En el dominio del tiempo real estricto (*hard real-time*), sin embargo, estos valores extremos no se pueden despreciar como ruido estadístico, sino que son críticos para estimar el tiempo de ejecución de peor caso (WCET). Dado que la planificabilidad ha de mantenerse incluso con dichos valores extremos, es común sobrestimar la potencia de cálculo requerida para estos sistemas, que queda ociosa la mayor parte del tiempo. Esto último demuestra la importancia de conseguir la máxima precisión posible en los métodos de medición. Validar empíricamente el comportamiento temporal límite de estas nuevas instrucciones sigue siendo un desafío abierto, objeto de la esta contribución.

III. Mecanismos para la Ejecución Temporizada

Las aplicaciones de tiempo real requieren mecanismos precisos para controlar el tiempo de activación de las tareas. Esta sección detalla el comportamiento arquitectónico de los métodos evaluados en este trabajo, desde las suspensiones estándar del kernel hasta las modernas extensiones del repertorio de instrucciones en el espacio de usuario.

A. Suspensión en el espacio del kernel: temporizadores de alta resolución

El mecanismo estándar para la ejecución periódica en Linux es el subsistema de temporizadores de alta resolución (`hrtimers`). Cuando un hilo invoca `clock_nanosleep`, el kernel programa un temporizador por *hardware* y suspende la tarea, cediendo la CPU. Al expirar, una interrupción *hardware* activa la correspondiente rutina de servicio (ISR), despertando al hilo e invocando al planificador. Este enfoque es eficiente energéticamente, ya que permite a la CPU entrar en estados de suspensión profunda, pero introduce una latencia significativa y variable: la ruta de activación implica la gestión de interrupciones, cambios de contexto y la interferencia del planificador [5]. Esta sobrecarga es excesiva para bucles de control que requieren precisión por debajo del microsegundo, requiriendo estrategias alternativas que mantengan la CPU activa.

B. Espera activa precisa: muestreo serializado del TSC

Para eliminar la latencia del planificador, las tareas de tiempo real emplean bucles de espera activa que muestrean continuamente el TSC. En arquitecturas modernas superescalares con ejecución fuera de orden no resulta adecuado leer el TSC mediante `rdtsc`, ya que al reordenar dinámicamente dicha instrucción, el procesador puede no leer en el instante deseado. Para evitarlo, se recurre a la instrucción `rdtscp`, que lee el contador tras la jubilación de todas las instrucciones precedentes. Así, el bucle de espera activa va tomando muestras hasta que el valor obtenido supera el plazo especificado (*deadline*). Este método proporciona la mayor precisión con respecto al plazo, pero fuerza al núcleo a permanecer en el estado C0 (completamente activo), incrementando la potencia dinámica y, en consecuencia, las necesidades de disipación térmica. A efectos experimentales, en este trabajo tomamos esta aproximación como referencia de alto rendimiento y alto consumo energético.

C. Retardo gestionado por hardware: la instrucción *TPAUSE*

La reciente extensión `WAITPKG` del juego de instrucciones de Intel x86-64 [13] busca un compromiso entre la precisión de la espera activa y la eficiencia energética de la suspensión de actividad, inviable para ejecuciones temporizadas por su alto coste de reactivación tras entrar en estados de suspensión. Esta extensión incluye instrucciones como `UMWAIT`, dirigida por eventos, y la instrucción `TPAUSE` (*Timed Pause*).

A diferencia de la instrucción heredada `PAUSE`, que proporciona un retardo breve y fijo en el *pipeline*, `TPAUSE` permite que el programa (modo no protegido, anillo 3) pase a un estado de bajo consumo dependiente de la implementación, con un tiempo de espera configurable, sin afectar al resto de hilos de ejecución. `TPAUSE` especifica un *deadline* absoluto del TSC en los registros implícitos `EDX:EAX`, implementando efectivamente una espera gestionada por *hardware*. Además, `TPAUSE` permite especificar el nivel de ahorro energético solicitado:

- *Estado C0.1, optimizado para la latencia.* El procesador entra en un modo de suspensión ligera diseñado para un despertar rápido, asegurando una penalización mínima en la latencia de salida para esperas cortas.
- *Estado C0.2, optimizado para la energía.* El procesador entra en un modo de suspensión más profundo. Aunque proporciona mayores ahorros de energía, incurre en una mayor latencia de activación debido al tiempo necesario para restaurar el estado microarquitectónico [17].

Debido a las características descritas, en este trabajo nos centramos en `TPAUSE` como mecanismo de espera alternativo a la espera activa implementada mediante `rdtscp`. Conviene señalar que el estado microarquitectónico efectivo que adopta el procesador

al ejecutar `TPAUSE` depende de la implementación, por lo que las métricas exactas de consumo y latencia varían según la generación del núcleo.

Nuestro objetivo específico es determinar si `TPAUSE`, configurado para situarse en el estado C0.1, puede mantener la precisión de un bucle continuo con `rdtscp` por debajo del microsegundo, reduciendo a la vez el consumo de energía.

IV. Metodología experimental

En esta sección se describe la plataforma de experimentación y las métricas utilizadas.

A. Plataforma de experimentación

Para garantizar la reproducibilidad y minimizar la interferencia del sistema operativo, todos los experimentos se ejecutan sobre una configuración de *hardware* y *software* optimizada para tiempo real.

Se ha utilizado un procesador Intel Core i5-13600HE de 13ª generación (*Raptor Lake*), que integra núcleos de alto rendimiento (P-cores) y núcleos de alta eficiencia (E-cores). Los experimentos utilizan exclusivamente P-cores. A fin de maximizar el determinismo temporal y evitar sobrecargas de latencia por migración de tareas entre ambos tipos de núcleos, todos los hilos experimentales se asignan estáticamente (*pinned*) a P-cores aislados mediante `sched_setaffinity`.

Se utiliza una distribución GNU/Linux Debian 13 (Trixie) con kernel 6.19. El sistema se ha configurado siguiendo las directrices de Intel para sistemas de tiempo real, activando el modo *Time Coordinated Computing*[1] (TCC) [18]. Los núcleos objetivo están completamente aislados del planificador del sistema operativo, tareas administrativas e interrupciones de *hardware*. De este modo, se consigue un entorno donde las latencias medidas reflejan el comportamiento microarquitectónico de los hilos objetivo, sin interferencia de actividades del sistema operativo. Dichos hilos se ejecutan bajo planificación `SCHED_FIFO` con prioridad 98.

El rendimiento se evalúa bajo dos escenarios operativos distintos: un sistema en reposo para establecer la línea base de latencia, y un sistema bajo condiciones sintéticas de estrés extremo. Para este último, se utiliza *hackbench*[19] como herramienta de generación de carga de CPU, asegurando así que las garantías de tiempo real se mantengan incluso en situaciones de alta contención de recursos.

B. Métricas de rendimiento

Para aislar y caracterizar el comportamiento de cada mecanismo, se ha definido un conjunto de métricas estrictas centradas en el determinismo temporal, la utilización de recursos y el consumo de energía diferencial.

Definimos la *precisión de planificación* A_{sched} como la diferencia entre el instante de activación deseado t_{target} y el instante de activación conseguido t_{wakeup} (instante de ejecución de la primera instrucción tras el período de espera):

$$A_{sched} = t_{wakeup} - t_{target}$$

Los valores obtenidos serán siempre positivos, y si son excesivos podrían incurrir en incumplimiento de plazos en tareas de tiempo real.

A efectos experimentales en este trabajo, definimos el *jitter* como la diferencia entre la precisión máxima y mínima observadas durante la ejecución de la prueba:

$$Jitter = \text{máx}(A_{sched}) - \text{mín}(A_{sched})$$

Esta formulación basada en el peor caso es obligatoria para la validación de sistemas críticos, ya que los promedios estadísticos ocultan los picos de latencia.

Por otra parte, aunque la interfaz Intel RAPL (*Running Average Power Limit*) proporciona un acceso conveniente a los contadores de energía del *hardware*, presenta limitaciones de precisión temporal, porque los contadores se actualizan a una frecuencia aproximada de 1 kHz (una vez cada milisegundo). Dado que los intervalos de espera evaluados oscilan típicamente en el rango de los microsegundos, RAPL no permite observar la energía consumida por una única operación entre actualizaciones. Para superar esta limitación, empleamos una estrategia de medición diferencial en estado estacionario. Como referencia de potencia del sistema, forzamos al núcleo analizado a entrar en el estado de reposo C1 usando `HLT` mientras el resto de núcleos mantiene la carga del experimento, y medimos dicha referencia (P_{base}). A continuación, ejecutamos en bucle el mecanismo a evaluar (e.g., `TPAUSE`). Específicamente, se ejecutan n iteraciones consecutivas del mecanismo por cada lectura del TSC. El valor de n se elige de modo que el tiempo total de ejecución del bloque sea significativamente mayor que la ventana de actualización de RAPL, permitiendo así acumular datos de energía suficientes para derivar un consumo medio estable ($P_{measured_loop}$). Con lo anterior, podemos calcular el consumo del mecanismo de espera como el incremento del consumo con respecto a la referencia del nucleo en reposo:

$$P_{\Delta} = P_{measured_loop} - P_{base}$$

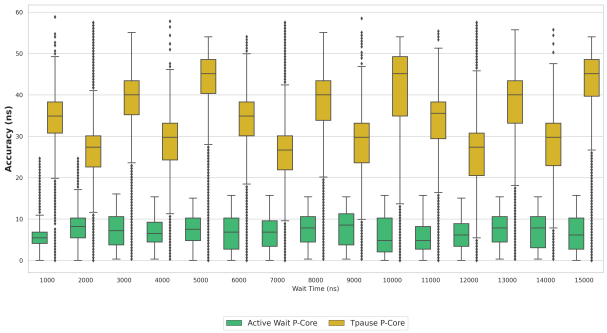
V. Resultados de evaluación

Ejecutadas las pruebas de estrés y caracterización en el i5-13600, los resultados se presentan en términos de estabilidad temporal y perfil de consumo energético.

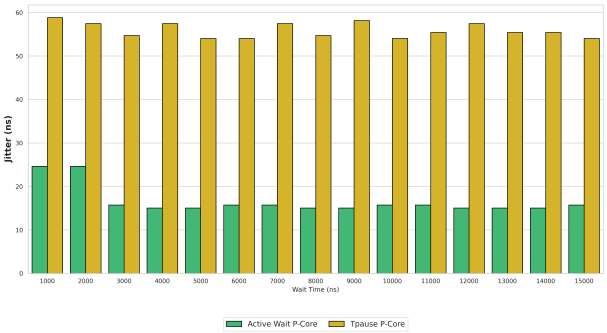
A. Estabilidad temporal en estado de reposo

En primer lugar se caracteriza el comportamiento de los mecanismos en un sistema en estado de reposo (*idle*) para establecer la referencia de latencia microarquitectónica. En esta configuración, el núcleo analizado está completamente aislado (*core pinning*),

[1]El modo TCC, activado desde BIOS, optimiza el *hardware* para máxima predictibilidad temporal.

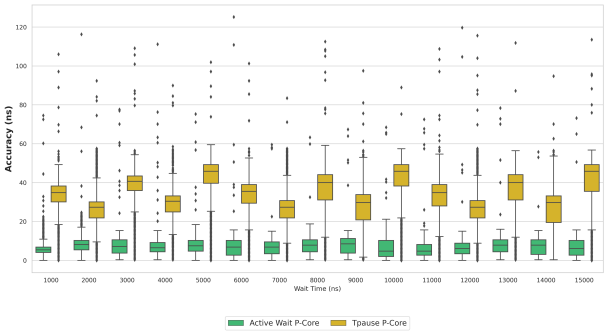


(a) Distribución de la precisión de planificación (A_{sched}).

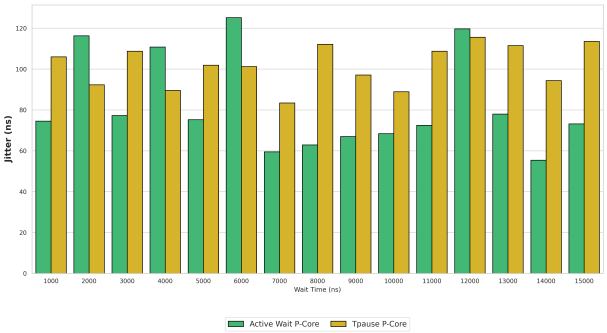


(b) Peor caso de Jitter absoluto.

Fig. 1: Evaluación de la estabilidad temporal en estado de reposo para núcleos P mediante espera activa (`rdtscp`), en verde, y `TPAUSE C0.1`, en amarillo.



(a) Distribución de la precisión de planificación (A_{sched}).



(b) Peor caso de Jitter absoluto.

Fig. 2: Evaluación de la estabilidad temporal bajo carga sintética extrema mediante `rdtscp` (verde) y `TPAUSE C0.1` (amarillo).

mientras que el resto de los núcleos ejecutan las tareas del sistema operativo. Esto minimiza la interferencia del planificador y permite observar el comportamiento intrínseco del *hardware*.

Para analizar su estabilidad, nos basamos en las métricas de precisión de planificación A_{sched} y *jitter* definidas en la sec. IV. Los resultados experimentales para el rango de esperas de 1 a 15 µs se muestran en la fig. 1. Los diagramas de caja (fig. 1a) representan la distribución completa de A_{sched}. La línea central de cada caja marca la mediana, los extremos de la caja delimitan el rango intercuartil (percentil 25 al 75), y los puntos individuales en forma de diamante representan los valores atípicos, críticos en la determinación del tiempo de ejecución de peor caso (WCET). El gráfico de barras (fig. 1b) muestra el *jitter*. Cada barra simboliza la diferencia absoluta entre el despertar más tardío y el más temprano observados para ese intervalo de espera específico.

La diferencia entre ambas estrategias es apreciable. Como muestra la fig. 1b, la instrucción `TPAUSE` (C0.1) presenta un *jitter* máximo que roza los 60 ns. Por el contrario, la espera activa tradicional (`rdtscp`) mantiene un *jitter* por debajo de los 30 ns para todos los tiempos de espera superiores a los 2 µs. Esta diferencia también es patente en la dispersión de los resultados (fig. 1a). Mientras que la espera activa agrupa la inmensa mayoría de sus activaciones con una desviación inferior a 10 ns respecto al objetivo, `TPAUSE` exhibe una variabilidad mayor. Aunque `TPAUSE` C0.1 solicita un estado de suspensión ligera, el proceso de reactivación del reloj del núcleo y la restauración del estado microarquitectónico al expirar el temporizador conllevan una penalización estocástica que duplica el margen de error frente a la retención completa de la CPU.

B. *Impacto de la contención de recursos (pruebas de estrés)*

Una vez evaluada la latencia sin carga, se evalúa la robustez de ambos mecanismos en condiciones realistas de alta carga. En este experimento, mientras un núcleo ejecuta los mecanismos de espera, el resto de núcleos del sistema se someten a una carga sintética extrema. Este escenario induce una alta contención en los recursos de la plataforma compartidos entre los núcleos.

Los resultados bajo este escenario de estrés se presentan en la fig. 2. Es patente el deterioro en la estabilidad temporal del sistema respecto al estado de reposo (fig. 1), pero el impacto es marcadamente asimétrico según el mecanismo empleado. En los diagramas de caja (fig. 2a), la mediana del método de espera activa (`rdtscp`) se mantiene competitiva. Sin embargo, aparecen significatimente valores atípicos extremos. Las latencias máximas de activación se disparan hasta rozar los 130 ns, como se observa, por ejemplo, en el intervalo de 6 µs. El gráfico de barras (fig. 2b) destaca que, bajo estrés continuo, la dispersión en el peor caso del muestreo convencional (`rdtscp`) se vuelve errática y, en múltiples intervalos (e.g., 2, 6, 12 y 14 μs), llega a superar al *jitter* de `TPAUSE`. Mientras que `TPAUSE` se mantiene en un rango razonablemente estable (acotado mayoritariamente entre 85 y 115 ns), el *busy-wait* presenta mayores fluctuaciones. Este comportamiento donde un

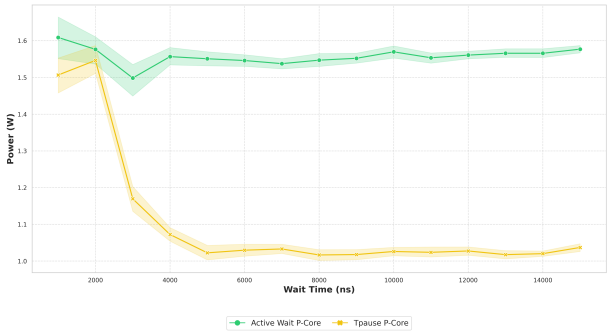


Fig. 3: Consumo de potencia dinámica diferencial por núcleo en vatios (W) en función del tiempo de espera solicitado. Las líneas sólidas muestran la media y las áreas sombreadas representan los intervalos de confianza.

mecanismo de suspensión ligera mejora la estabilidad temporal en el peor caso a la espera activa se explica por la presencia de valores atípicos que elevan el *jitter* máximo en ambos métodos. Dado que ambos mecanismos recurren al TSC para determinar el instante de finalización de la espera, nuestra hipótesis es que dichos valores atípicos podrían estar originados por latencias anómalas en el acceso a dicho contador, posiblemente agravadas por la contención de recursos microarquitectónicos compartidos bajo carga extrema.

C. Eficiencia energética diferencial

Una vez establecido el impacto de ambos mecanismos en la estabilidad temporal, el objetivo final de esta evaluación es cuantificar el coste energético asociado a cada estrategia. Para ello, utilizamos la metodología de medición diferencial descrita en la sec. IV, forzando al núcleo objetivo a entrar en estado de reposo C1 como referencia. La fig. 3 presenta los resultados de potencia dinámica media consumida por el núcleo durante esperas en el rango de 1 a 15 µs.

La espera activa tradicional (`rdtscp`) presenta una estabilidad alta: el consumo dinámico se mantiene prácticamente plano y constante, en un rango de 1.55 W a 1.6 W aproximadamente, para todos los tiempos de espera evaluados (línea verde en fig. 3). Esto confirma que mantener el núcleo en estado C0 encuestando el TSC mediante `rdtscp` maximiza la potencia dinámica sin importar la duración de la espera.

Por el contrario, la instrucción `TPAUSE` (C0.1, en amarillo) exhibe un comportamiento energético no lineal y altamente dinámico. En tiempos de espera muy cortos (< 2 µs), el consumo energético es elevado y similar al de la espera activa, manteniéndose por encima de 1.5 W a 1 µs. Sin embargo, a partir de ese punto, se observa una reducción drástica y pronunciada del consumo. En el intervalo de 4 µs, la potencia ya ha caído por debajo de 1.1 W, y para esperas superiores a los 5 µs, el consumo se estabiliza en una meseta de bajo consumo entre 1.02 W y 1.04 W para el resto de los intervalos, aproximadamente.

Estos resultados energéticos, combinados con los de estabilidad bajo estrés (sec. V-B), consolidan `TPAUSE` C0.1 como la mejor solución para sistemas COTS multinúcleo. Hasta ahora, garantizar precisión temporal sin depender del planificador del sistema operativo obligaba a recurrir a la espera activa, lo que mantiene el núcleo ejecutando instrucciones a plena potencia durante toda la fase de espera. En cambio, `TPAUSE` C0.1 ofrece una precisión de peor caso competitiva y reduce drásticamente el consumo al mantener el núcleo en un estado de baja potencia durante la espera.

VI. Conclusiones y Trabajo Futuro

Se ha presentado una caracterización empírica de la instrucción `TPAUSE` (C0.1) como alternativa a la espera activa (`rdtscp`) en espacio de usuario bajo Linux con expulsión `PREEMPT_RT` y configuración *software/hardware* para tiempo real. Los resultados en estado de reposo demuestran que `TPAUSE` mantiene una precisión competitiva (peor caso < 60 ns) comparado con la espera activa (< 30 ns). Para intervalos superiores a 2 µs, `TPAUSE` logra reducir el consumo de potencia dinámica en más de un 35 %, mitigando drásticamente el impacto térmico y energético asociado a la espera activa.

Las pruebas bajo carga extrema han revelado un comportamiento contraintuitivo respecto a la estabilidad temporal. Monopolizar el *pipeline* garantiza en general la máxima estabilidad; sin embargo, el continuo acceso a registros y memoria efectuado por `rdtscp` introduce contención en el acceso a recursos compartidos, disparando las latencias máximas por encima de los 120 ns. Por el contrario, al delegar la espera al *hardware* y pausar el núcleo, `TPAUSE` es inmune a la congestión en cache y bus, mostrando un *jitter* de peor caso mucho más estable y predecible bajo alta carga del sistema.

Según estos datos experimentales, `TPAUSE` en modo C0.1 representa un mecanismo idóneo para bucles de control de alta frecuencia en plataformas COTS, ofreciendo un equilibrio óptimo entre determinismo bajo estrés y eficiencia. El trabajo futuro se centrará en utilizar *contadores de monitorización de rendimiento* (PMCs) para perfilar a bajo nivel la contención microarquitectónica observada, extender la evaluación a los núcleos de alta eficiencia (E-cores) en arquitecturas híbridas, e integrar esta instrucción en *middleware* industrial. y de robótica como ROS 2.

Agradecimientos

Este trabajo ha sido financiado por MICIU/AEI/10.13039/501100011033 (proyecto PID2022-136454NB-C22), y por el Departamento de Ciencia, Universidad y Sociedad del Conocimiento del Gobierno de Aragon (ayuda a grupo de investigación de referencia T58_23R).

Referencias

[1] Federico Reghenzani, Giuseppe Massari, and William Fornaciari, "The real-time linux kernel: A survey on PREEMPT_RT," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 1, pp. 1–36, 2019.

[2] Steven Rostedt and Darren Hart, "Internals of the RT patch," in *Proceedings of the Linux Symposium*, 2007, vol. 2.

[3] Robert Love, *Linux Kernel Development*, Pearson Education, 3rd edition, 2010.

[4] Francis M David, Jeffrey C Carlyle, and Roy H Campbell, "Context switch overheads for linux on arm platforms," in *Proceedings of the 2007 workshop on Experimental computer science*. ACM, 2007, p. 3.

[5] Thomas Gleixner and Douglas Niehaus, "Hrtimers and beyond: transforming the linux time subsystems," in *Proceedings of the Linux Symposium*, 2006, vol. 1, pp. 333–346.

[6] Rob Williams, *Real-Time Systems Development*, Butterworth-Heinemann, 2008, See Chapter on Polling vs Interrupts.

[7] Jeremy Brown and Brad Martin, "How fast is fast enough? choosing between xenomai and linux for real-time applications," in *Proceedings of the Real-Time Linux Workshop (RTLWS)*, 2010, pp. 1–11.

[8] Dakshina Dasari, Birgit Akesson, Vincent Nelis, Mojtaba Amani, and Stefan M. Petters, "Identifying the sources of unpredictability in COTS-based multicore systems," in *18th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2013, pp. 1–8.

[9] The Linux Foundation, "Real-time linux wiki: Howto - cyclictest," `https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest/start`, n.d., Accessed: 2024-05-06.

[10] Gabriele Paoloni, *How to Benchmark Code Execution Times on Intel® IA-32 and IA-64 Instruction Set Architectures*, Intel Corporation, 2010, White Paper.

[11] UEFI Forum, *Advanced Configuration and Power Interface (ACPI) Specification, Version 6.5*, 2022, Accessed: 2026.

[12] Daniel Hackenberg, Thomas Ilsche, Robert Schöne, et al., "Power measurement techniques on standard compute nodes: A quantitative comparison," *Performance Evaluation*, vol. 1, pp. 1–15, 2015.

[13] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual*, Intel Corporation, 2023, Volume 2: Instruction Set Reference.

[14] Alexey Kukanov, "The why and how of the pause instruction," *Intel Software Network*, 2012.

[15] Efraim Rotem, Alon Naveh, Doron Rajwan, Avinash Ananthakrishnan, and Eliezer Weissmann, "Power management architecture of the 6th generation Intel Core microarchitecture," in *IEEE Hot Chips 27 Symposium (HCS)*. IEEE, 2015, pp. 1–38.

[16] Robert Schöne, Daniel Molka, et al., "Latency-aware power management for real-time systems," in *IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2019.

[17] Malte-Christian Kuns, Hannes Tröpgen, and Robert Schöne, "An analysis of user-space idle state instructions on x86 processors," in *Proceedings of the 16th ACM/SPEC International Conference on Performance Engineering (ICPE)*, Toronto, ON, Canada, 2025, pp. 232–239, ACM.

[18] Intel Corporation, *Intel® Time Coordinated Computing (TCC) User Guide (Public-Version 1)*, Jan. 2026, Document No. 831067.

[19] Clark Williams and John Kacur, "rt-tests: A collection of latency testing tools for the linux kernel," 2024, Accedido: marzo 2026.

[20] Intel Corporation, *Intel® Ethernet Controller I210: Datasheet*, January 2013, Rev. 3.1.

[21] Fabian Rauscher, Andreas Kogler, Jonas Juffinger, and Daniel Gruss, "Idleleak: Exploiting idle state side effects for information leakage," *Proceedings 2024 Network and Distributed System Security Symposium*, 2024.

[22] Daniel Hackenberg, Thomas Ilsche, Robert Schöne, Daniel Molka, Maik Schmidt, and Wolfgang E Nagel, "Power measurement techniques on standard compute nodes: A quantitative comparison," *Performance Evaluation of Computer and Communication Systems (PERFORM)*, pp. 194–204, 2013.

[23] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman, *Linux Device Drivers*, O'Reilly Media, 3rd edition, 2005.

Forge2: Plataforma de Experimentación para una Refrigeración Líquida Directa con Cambio de Fase

Pedro Chaves, Enrique Torres, Alejandro Valero y Víctor Viñals[1]

Resumen— **Este trabajo presenta Forge2, una plataforma experimental para el control, medida, registro y visualización de variables térmicas, eléctricas y de rendimiento en un procesador Intel Core Ultra desbloqueado y refrigerado mediante un circuito de refrigeración líquida directa. La plataforma integra mecanismos de control del punto de operación del procesador y del subsistema de refrigeración, instrumentación adicional propia para la medida de temperaturas, caudal y potencia, así como una infraestructura** *software* **capaz de unificar series temporales procedentes de fuentes heterogéneas. Sobre esta base, se presentan tres ensayos representativos: la inferencia térmica del mapa de núcleos a partir de su respuesta térmica, la caracterización experimental de la curva voltaje-frecuencia y el estudio dinámico del sistema bajo dos regímenes extremos de refrigeración. Los resultados obtenidos permiten validar la plataforma y mostrar su utilidad para la caracterización térmica y energética de procesadores comerciales. Forge2 constituye una base útil para estudios posteriores sobre caracterización energética y evaluación de nuevas soluciones de refrigeración líquida basadas en emulsiones con cambio de fase.**

Palabras clave— **caracterización térmica, instrumentación experimental, procesadores híbridos.**

I. Introducción

El auge actual de la Inteligencia Artificial (IA) está impulsando a la computación de altas prestaciones (HPC) desde su nicho tradicional en la supercomputación hacia mercados más modestos, puesto que los fabricantes de chips intuyen beneficios en todos los frentes posibles. De esta manera, en todo el continuo de recursos de computación, desde las CPUs hasta los ASICs, pasando por las GPUs y FPGAs, observamos cómo aparecen silicios de gran superficie que funcionan a frecuencias extraordinarias.

Actualmente, chips comerciales como la GPU NVIDIA Blackwell B200 suben la apuesta, precisando la disipación de un *Thermal Design Power* (TDP) de hasta 1200 W en un área aproximada de 1600 mm^2 [1]. Esta elevada concentración de potencia da lugar a ***hotspots*** con gradientes térmicos intensos que afectan al rendimiento sostenido y a la fiabilidad. En este sentido, una gestión térmica inadecuada no solo reduce el rendimiento y acelera el envejecimiento de los componentes, sino que también puede provocar errores e incluso dañar el *hardware*, lo que convierte a la refrigeración en un aspecto crítico de diseño y gestión de estos sistemas [2]. Cualquier solución de refrigeración busca reducir la excursión térmica y mantener temperaturas más estables para prolongar la vida útil del hardware, evitar el ***throttling térmico*** (i.e., reducción automática de la frecuencia y/o voltaje de un procesador para limitar la temperatura y evitar sobrecalentamiento) y permitir una mayor densidad de cómputo en racks HPC o centros GPU-intensivos.



Fig. 1: Vista general de Forge2.

Este trabajo presenta la plataforma **Forge2** (véase la Figura 1). Esta plataforma establece la primera iteración de un proyecto interdisciplinar entre ingenieros térmicos y arquitectos de computadores para analizar una alternativa disruptiva de **refrigeración líquida directa (DLC)** de chips. **Forge2** es una plataforma para el control, medida, registro y visualización de series temporales de variables eléctricas, térmicas y de rendimiento en un chip multiprocesador desbloqueado (permite ajustar por separado frecuencias y voltajes) con dos clases energéticas de núcleos.

Las soluciones actuales basadas en **DLC** o *heat pipes*, aunque mejoran la gestión térmica, siguen presentando limitaciones en términos de capacidad efectiva de extracción de calor, integración a nivel de

[1]Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Instituto de Investigación en Ingeniería de Aragón – I3A, España, e-mails: {844390,ktm,alvabre,victor}@unizar.es.

sistema y respuesta ante cargas térmicas altamente dinámicas, lo que restringe su rendimiento en escenarios de alta densidad de potencia [3]. En este sentido, proponemos estudiar una DLC cuyo fluido sea una ***emulsión***, es decir, un líquido que mantiene en suspensión pequeñas partículas sólidas que se funden (que cambian de fase) al circular por el bloque térmico que está en contacto con el chip.

Las emulsiones que buscamos consisten en una dispersión de micropartículas de PCM, generalmente una parafina, estabilizadas mediante agentes emulsionantes o surfactantes en un fluido continuo, comúnmente agua. Esta dispersión permite aprovechar la capacidad del PCM para almacenar o liberar grandes cantidades de energía térmica durante su cambio de fase sólido-líquido o viceversa, mientras se mantiene en forma líquida, permitiendo así su bombeo. Además, la transferencia de calor ocurre a una temperatura aproximadamente constante, lo que mejora la eficiencia del sistema de refrigeración, al reducir la temperatura del fluido como consecuencia de una mayor capacidad calorífica [4].

Esperamos que el uso de emulsiones PCM contribuya a reducir corrientes de fuga, uniformizar la distribución térmica, facilitar el DVFS y disminuir los ciclos térmicos del encapsulado.

Así pues, nuestro objetivo principal es estudiar en un sistema comercial los beneficios potenciales de una refrigeración líquida directa basada en **emulsiones con cambio de fase**. Sin embargo, antes de abordar ese estudio, es necesario disponer de una **plataforma experimental** bien instrumentada, reproducible y de fácil uso. En este trabajo nos centramos en esa primera etapa mediante la caracterización de Forge2 y la validación de sus capacidades de **control**, **monitorización** y **análisis**. En este contexto, se estudian aspectos no triviales del sistema actual, como la inferencia térmica del mapa de núcleos, la curva voltaje/frecuencia observada y la respuesta dinámica bajo distintos regímenes de refrigeración. Al mismo tiempo, buscamos que la plataforma pueda servir en el futuro como herramienta accesible para usuarios no expertos interesados en la optimización energética de aplicaciones intensivas en consumo.

El resto del artículo se organiza como sigue. La Sección II proporciona los objetivos y requisitos de la plataforma. La Sección III describe la plataforma. La Sección IV presenta las herramientas *software*. La Sección V muestra la metodología empleada. La Sección VI cubre la evaluación experimental. La Sección VII resume los trabajos relacionados. Finalmente, la Sección VIII presenta las conclusiones y trabajo futuro.

II. Objetivos y Requisitos

Forge2 es una plataforma de experimentación de procesadores Intel Core Ultra desbloqueados con refrigeración DLC. Sus diferentes elementos se han dispuesto en un plano horizontal para facilitar su manipulación e instrumentación. Para utilizar Forge2 como banco de ensayos en el estudio de DLC y de futuras emulsiones con cambio de fase, se deben satisfacer una serie de requisitos de control, medida y monitorización. La Tabla I enumera los elementos físicos, lógicos y la instrumentación empleada.

Tabla I: Enumeración de elementos físicos, elementos lógicos e instrumentación en Forge2.

Elementos físicos
- Placa base: ASUS PRIME Z890-P WIFI, formato ATX, chipset Intel Z890. - Chip multiprocesador: Intel Core Ultra 7 265KF (Series 200), socket LGA1851. - Memoria: 32GB (2 x DIMM Kingston DDR5). - Fuente de alimentación: Be quiet PurePower 12M ATX 3.0 850 W, 80 PLUS Gold ($\eta = 93{,}2\,\%$). - Equipo de DLC: Alphacool Core Hurrican 240mm XT45 Hard-Tube.
Elementos lógicos
- BIOS: Version 3002, 11,76 MB, 2026/01/30 para ASUS PRIME Z890-P WIFI. - Sistema operativo: Windows 11 Enterprise (Microsoft). - Control de frecuencias y tensiones: Intel XTU. - Control del enfriamiento: FanControl (Rémi Mercier). - Monitorización: HWiNFO (Martin Malík). - Monitorización: Intel PCM (Performance Counter Monitor). - Control de experimentos: scripts de Python. - Comunicación: MQTT (estándar abierto de OASIS). - Base de datos: InfluxDB (código abierto). - Visualización: Grafana (Grafana Labs). - Carga de trabajo: pruebas propias de Intel XTU y Prime95 (GIMPS).
Instrumentación
Preexistente - Desde la placa base: potencias, tensiones, temperaturas, r.p.m., etc. - Desde el chip: tensiones, frecuencias, temperaturas, instrucciones, ciclos, etc.
Añadida - Sensor Alphacool ES HighFlow: caudal en el circuito DLC y temperatura de agua en la entrada del bloque CPU. - Alphacool Eiszapfen: otro sensor de temperatura de agua en la salida del bloque de CPU. - Varios sensores de temperatura del aire en la entrada y salida del intercambiador de calor del Alphacool XT45. - Los valores se recogen en ForgeSense, un microcontrolador Espressif ESP32 de forma directa (caudal, 1 muestra/s) o a través del ADC ADS1256 (24 bits, 10 muestras/s) y se envían al PC por USB. - Analizador de potencia PPA500 (Newtons4th ltd.) en la toma de AC, antes de la fuente; envío al PC por USB.

A. Objetivos Funcionales

Forge2 permite **controlar** la ejecución de programas de prueba en un chip multiprocesador refrigerado con un equipo DLC para poder **medir** de forma continua y **registrar** de forma compacta y permanente series temporales de:

1. Valores de las variables de control aplicadas (e.g., valores de V y F para cada núcleo o velocidades de bombeo y ventilación).
2. Rendimientos software (e.g., tiempo de ejecución o IPC).
3. Magnitudes físicas (e.g., potencia, T_{core}, $T_{package}$, caudal de bombeo, temperaturas en varios puntos del circuito de enfriamiento o temperaturas en la placa base).

Forge2 también incluye herramientas de **visualiza-**

Tabla II: Ejemplos de control y medida en Forge2.

Magnitud	Tipo	Origen
Carga de trabajo	Control	Prime95, XTU
Frecuencia	Control	BIOS, XTU
Voltaje	Control	BIOS, XTU
Políticas DVFS	Control	BIOS, XTU, SO
RPM Ventiladores	Control	FanControl
RPM bomba	Control	FanControl
Caudal líquido	Medida	ForgeSense
Temp. núcleos	Medida	HWiNFO
Temp. aire	Medida	ForgeSense
Temp. líquido	Medida	ForgeSense
Instrucciones, ciclos	Medida	Intel PCM
Potencias eléctricas	Medida	CPU, placa base, PPA500

ción, tanto interactiva como de las series temporales almacenadas. La Tabla II detalla una serie de ejemplos de acciones de control y medidas de interés.

B. Requisitos

Buscamos obtener series temporales con valores que provienen de orígenes distintos, con frecuencias de muestreo diferentes. Además, el conjunto placa base/procesador/sistema operativo dispone de muchos mecanismos adaptativos que persiguen alcanzar de forma automática una relación prestaciones/energía adecuada. En este contexto, los requisitos del proyecto que enumeramos a continuación suponen retos no triviales a superar:

1. **Reproducibilidad**. Supone entender bien los mecanismos DVFS automáticos y de protección térmica, así como conectarlos o desconectarlos desde la BIOS, el Sistema Operativo (SO) y/o desde Intel XTU. Por ejemplo, las tecnologías Intel *SpeedStep* (IST), *Enhanced* IST (EIST) e Intel *Speed Shift* (HWP, *Hardware Controlled Performance*) tienen en común el ajuste del punto de operación en la curva V/F para conseguir un estado ACPI P (*Performance*) con un comportamiento prestaciones/potencia adecuado; IST y EIST se realizan desde el SO, vía interrupción, con latencias elevadas (20-100 ms), mientras que HWP realiza lo mismo pero más rápido (1-15 ms) sin intervención del SO desde la *Power Unit* (PUNIT), que es un microcontrolador dedicado al control térmico y energético de todos los núcleos.
2. **Sincronización**. Es necesario organizar las marcas de tiempo para integrar fuentes de datos con frecuencias de muestreo diferentes.
3. **Control del régimen de operación de los núcleos**. Es necesario poder activar y desactivar núcleos de forma individual, modificando su régimen de operación en voltaje y frecuencia.
4. **Control del régimen de operación de la refrigeración**. Es necesario controlar el régimen de operación de la extracción de calor mediante el control del caudal del circuito líquido y de la ventilación del radiador (intercambiador de calor aire-agua).
5. **Monitorización de rendimientos y temperaturas**. Es necesario registrar la actividad de los núcleos (instrucciones retiradas, ciclos, tiempo de ejecución, etc.) y sus temperaturas; puede ser conveniente en el futuro añadir actividad y temperaturas del *uncore* (caches privadas, cache compartida, red de interconexión D2D, etc.).

III. Detalle de los Elementos Hardware

El sistema experimental se basa en un procesador **Intel Core Ultra 7 265KF** perteneciente a la familia Arrow Lake-S para equipos de sobremesa [5]. Se trata de una CPU híbrida de 20 núcleos, organizada en 8 *performance cores* (P-cores) y 12 *efficient cores* (E-cores). A su vez, los E-cores están agrupados en grupos (*clústeres*) de 4 compartiendo una cache L2 de 4 MiB. La frecuencia base es de 3,9 GHz y la turbo de hasta 5,5 GHz en el caso de los P-cores. Estas frecuencias son 3,3 y 4,6 GHz, respectivamente, en los E-cores. El rango de potencia TDP es de 125 W de *Processor Base Power*, mientras que la potencia máxima alcanza los 250 W (*Maximum Turbo Power*).

La elección de este procesador respondió a varios criterios. En primer lugar, su arquitectura híbrida permite estudiar de forma diferenciada el comportamiento térmico y energético de P-cores y E-cores bajo condiciones de carga controladas. En segundo lugar, la ausencia de *multithreading* simplifica la asignación de carga y la interpretación de las medidas, al evitar ambigüedades asociadas a hilos lógicos compartiendo un mismo núcleo físico. En tercer lugar, la variante 265KF (K: desbloqueado, F: ausencia de gráficos integrados) pertenece a la gama desbloqueada orientada a *overclocking*, lo que, junto con las capacidades de ajuste avanzadas de la generación Core Ultra Series 2 [6], la hace especialmente adecuada para experimentos de control preciso de frecuencia en incrementos de 16,6 MHz, incluyendo ajuste individual por núcleo, así como de voltaje y potencia.

Por último, también se consideró su relación coste-prestaciones frente al procesador Intel Core Ultra 9 285K: aunque este último incrementa el número total de núcleos hasta 24 mediante la incorporación de cuatro E-cores adicionales, su precio era sensiblemente superior, lo que hacía del 265KF una opción más equilibrada para la plataforma experimental [7].

Dado que el procesador empleado pertenece a la gama desbloqueada de Intel, fue necesario utilizar una placa base basada en *chipset* Z890 para poder acceder a las funciones avanzadas de ajuste y overclocking de la plataforma. Se empleó la placa base **ASUS PRIME Z890-P WIFI** [8], una placa ATX con *socket* LGA1851. La placa base incorpora un subsistema de alimentación basado en 18 etapas DrMOS de 80 A. En nuestro caso, esta elección resulta adecuada por ofrecer una base estable para experimentos prolongados de control de frecuencia, voltaje y potencia, así como por proporcionar acceso en BIOS a las opciones necesarias para explotar las capacidades del procesador.

Como sistema de refrigeración líquida se empleó un equipo **Alphacool Core Hurrican 240mm XT45 HardTube** [9]. Su elección respondió a que pro-

porciona un conjunto completo y, al mismo tiempo, suficientemente modular para una plataforma experimental: incorpora un intercambiador de calor agua-aire (radiador de cobre pintado de negro) de $271,5 \times 124 \times 45\ mm$ refrigerado por dos ventiladores regulables, un bloque de CPU con base de cobre niquelado y una bomba regulable por PWM con un caudal máximo especificado de 340 L/h.

El circuito de refrigeración también se instrumentó con un sensor Alphacool ES HighFlow de caudal y temperatura [10], así como con un sensor de temperatura Alphacool Eiszapfen adicional [11], instalados de forma que permitieran monitorizar tanto la temperatura del líquido a la entrada y a la salida del bloque de CPU como el caudal real de circulación en el circuito. En nuestro caso, estas características resultan relevantes por ofrecer capacidad de disipación suficiente para el procesador, facilitar el control reproducible de ventiladores y bomba, integrar de forma segura medidores y permitir futuras modificaciones o añadir instrumentación al circuito.

Adicionalmente, se ha diseñado y fabricado un subsistema de adquisición e instrumentación específico, basado en un microcontrolador Espressif ESP32 y un conversor analógico-digital (ADC), destinado a la adquisición de hasta ocho canales de temperatura que denominamos **ForgeSense**. Concretamente, este sistema permite instrumentar con sondas NTC de pequeño tamaño y bajo tiempo de respuesta distintos puntos de entrada y salida de aire en torno a los ventiladores, así como medir el caudal en litros por minuto (previamente calibrado) y las temperaturas del circuito líquido.

Las muestras se transmiten al PC por USB tras aplicar un filtrado combinado basado en mediana y suavizado exponencial para reducir el ruido y valores espurios sin perder capacidad de seguimiento dinámico. El sistema de adquisición se apoya en un ADC ADS1256 de 24 bits y hasta 30 kS/s, configurado en nuestro caso a una frecuencia de muestreo de 10 Hz por canal. La precisión efectiva alcanzable depende no solo del convertidor, sino también del ruido, la referencia y la calibración de la cadena completa de medida. Por ello, las sondas se calibran respecto a un termopar calibrado de referencia.

La potencia eléctrica total se mide colocando un analizador de potencia **PPA500** de Newtons4th ltd. en serie con la toma AC. El analizador se lee mediante conexión USB; la comparación de este valor con el agregado de potencias en los chips de procesamiento y memoria nos permite contabilizar las pérdidas en el chipset, reguladores, fuente y otros.

IV. Herramientas Software

Forge2 está compuesto por tres tipos de herramientas software: a) *control*, para fijar el punto de operación del sistema, b) *monitorización*, para observar el comportamiento térmico, energético y computacional, y c) *orquestación, almacenamiento y visualización*, para automatizar experimentos y gestionar los datos generados

A. Herramientas de Control

Las herramientas de control permiten conseguir regímenes de funcionamiento reproducibles, tanto en lo relativo a la generación de calor como a su extracción mediante el subsistema de refrigeración.

Un procesador desbloqueado, como el 265KF, puede ejecutar el mismo programa con diferentes pares V/F y varios mecanismos, habilitados o no, de protección o de ajustes prestaciones/potencia. Denominamos *punto de operación* a una configuración completa de todos los parámetros y mecanismos que afectan a la velocidad y consumo del procesador. Un punto de operación es *estable* para un programa si permite una ejecución completa sin "colgarse".

La herramienta básica para generar puntos de operación es la **interfaz de configuración UEFI BIOS** (*UEFI Setup Utility*). Es una interfaz gráfica sencilla, basada en menús. Reside, junto con los drivers del sistema, en un chip de memoria NVRAM de 32 MiB con acceso SPI. Permite establecer de forma interactiva todos los ajustes posibles, ya sean valores numéricos o ajustes de comportamiento. Por ejemplo, pueden fijarse los valores de frecuencia (multiplicadores de la frecuencia base) y voltaje para cada núcleo, o los límites de potencia y temperatura. Pero también es posible activar o desactivar mecanismos automáticos; por ejemplo, si en la UEFI activamos el *Thermal Monitor*, estamos dando permiso a la PUNIT para que active el *clock gating* en la etapa de *issue* ante una emergencia térmica o de consumo, es decir, que ralentice el lanzamiento de instrucciones si se han superado unos determinados umbrales numéricos. La funcionalidad **ASUS *User Profile*** permite almacenar y recuperar puntos de operación, en forma de configuraciones UEFI, que se almacenan en el mismo chip NVRAM, incluyendo importación/exportación mediante USB [12].

Cambiar el punto de operación del procesador sin el procedimiento anterior, y sin tener que reiniciar el sistema cada vez es un requisito imprescindible para experimentar de forma eficiente. Una posibilidad es escribir los *Model Specific Registers* (MSR) –*frequency ratio*, *voltage offset*, etc.– o bien de forma directa, o de forma indirecta, a través del Mailbox de la unidad de gestión de energía (PUNIT), programando los drivers apropiados en Linux o Windows. Si bien esta es la opción de mayor control, también supone un riesgo y esfuerzo notables. En Forge2 hemos seleccionado una alternativa de Intel para Windows, la utilidad de usuario *Extreme Tuning Utility* **(XTU)**, que permite fijar puntos de operación sin necesidad de reiniciar la máquina [13, 14]. A su vez, esta elección ha determinado el uso del **SO Windows** y el desarrollo de nuestro software en dicho contexto.

XTU es una herramienta concebida para controlar procesadores desbloqueados, principalmente de las series K y X, incluyendo variantes como la KF. La Figura 2 muestra la interfaz de *Advanced Tuning* de XTU para el caso de los P-cores en nuestro procesador. Desde su menú superior se accede a distintos

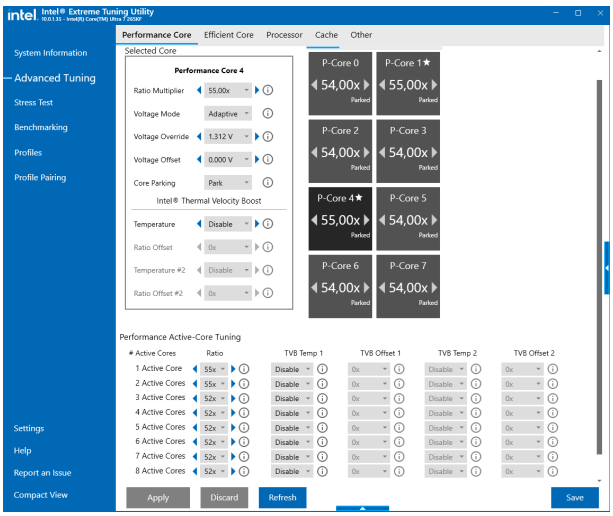


Fig. 2: Captura del menú *Advanced Tuning* de Intel XTU para el ajuste de los P-cores en el Intel Core Ultra 7 265KF.

grupos de configuración, incluyendo P-cores, E-cores, procesador, cache y otras opciones de ajuste global. En el caso de los P-cores, XTU permite definir de forma individual parámetros como el multiplicador (*ratio*), el modo de voltaje y sus valores de *override* u *offset*, así como fijar límites dependientes del número de núcleos activos. Esta organización facilita explorar y aplicar de forma sistemática distintos puntos de operación sin necesidad de reiniciar el sistema.

Además de las opciones de ajuste, XTU incorpora programas de *stress* que permiten comprobar de forma interactiva la estabilidad de la configuración activa, así como un sistema de "perfiles" para almacenar distintos puntos de operación previamente validados. Esto permite garantizar la reproducibilidad entre experimentos, incluso cuando los ensayos se realizan en momentos distintos o tras cambios intermedios en la configuración de la plataforma. En Forge2 se aplicamos estos perfiles de forma automatizada desde aplicaciones externas, lo que permite integrarlos dentro de campañas experimentales que cambian los puntos de operación de forma dinámica. En la propia interfaz se observa, además, que Intel establece en fábrica dos P-cores como *favored cores*; en el procesador empleado corresponden a los P-core 1 y P-core 4, identificados con un símbolo de estrella.

Completamos la definición de punto de operación fijando en el sistema de refrigeración una cierta capacidad de evacuación del calor al ambiente. Para ello Forge2 usa **FanControl**, que es una utilidad de software libre para Windows [15]. Permite gobernar la velocidad de la bomba y de los ventiladores del intercambiador de calor (r.p.m.). Tras evaluar varias opciones, se adoptó FanControl porque permite definir curvas de control basadas en distintos sensores, incluyendo tanto magnitudes internas (e.g., temperatura del procesador) como sensores externos (e.g., temperatura del líquido). Las curvas hacen posible mantener regímenes de funcionamiento estables en condiciones nominales y, a la vez, forzar incrementos rápidos de ventilación o caudal cuando se superan determinados umbrales, mejorando la seguridad de los ensayos. Además, FanControl también permite almacenar múltiples configuraciones y seleccionar una de ellas mediante argumentos de línea de comandos durante la ejecución, lo que resulta útil para automatizar el uso de distintas políticas de enfriamiento y asegurar la reproducibilidad.

B. Herramientas de Monitorización

Las herramientas de monitorización seleccionadas, HWiNFO e Intel PCM, permiten observar y capturar el estado térmico, energético y computacional de la plataforma durante los ensayos.

HWiNFO es un software libre para Windows que permite manejar la mayoría de variables de estado de la plataforma [16, 17]. Los valores que captura y la frecuencia de muestreo son programables. La serie temporal de valores se puede visualizar y almacenar en un fichero. HWiNFO también expone los datos mediante un interfaz de memoria compartida permitiendo el acceso compartido a aplicaciones externas.

Por un lado, HWiNFO reúne la información que procede del chip multiprocesador (voltaje, frecuencia y temperatura de cada núcleo, consumo del *package*, etc.), de la placa base (consumos y temperaturas de subsistemas, valores del regulador de voltaje que alimenta al socket LGA1851, etc.) y parte de la refrigeración (r.p.m. de bomba y ventiladores).

Por otro, HWiNFO permite integrar las medidas de sensores externos (e.g., desde ForgeSense o Intel PCM). El resultado que conseguimos es disponer de toda la información visible desde una misma aplicación.

Se evaluó LibreHardwareMonitor como alternativa, pero se optó finalmente por HWiNFO por ofrecer una cobertura más completa de sensores del procesador, de la placa base y permitir una integración más fácil de nuestra instrumentación.

Intel *Performance Counter Monitor* (**PCM**) permite obtener métricas de actividad real del procesador a partir de contadores hardware, incluyendo instrucciones retiradas, ciclos, número de instrucciones por ciclo (IPC) y otros indicadores relacionados con el estado ACPI, las caches y la memoria [18]. El número de instrucciones y el IPC son una medida directa del trabajo computacional realizado en cada experimento.

C. Herramientas de Orquestación, Almacenamiento y Visualización

La automatización de los experimentos se implementa mediante *scripts* en **Python**, responsables de coordinar la ejecución de las pruebas, controlar la configuración de Forge2 y recoger las métricas generadas por las distintas herramientas. Estas medidas se envían mediante *Message Queuing Telemetry Transport* (**MQTT**) a un servidor externo, donde **InfluxDB** se emplea como base de datos (BD) de series temporales [19] y **Grafana** como herramienta de visualización remota y análisis de los resultados [20]. Esta arquitectura resulta particularmente convenien-

te, ya que Forge2 busca ofrecer la evolución temporal de las magnitudes medidas —y no solo en sus valores medios o finales—, permitiendo analizar la dinámica térmica y energética en cada experimento.

Concretamente, un script se encarga de leer los datos de ForgeSense y entregárselos a HWiNFO para la visualización interactiva y a su vez enviarlos al servidor de la BD. Otro script inicia Intel PCM, recoge sus datos y al igual que el anterior los envía a HWiNFO y a la BD. Un tercer script accede a la memoria compartida de HWiNFO y envía a la BD otras magnitudes relevantes. Las magnitudes que recoge cada uno de ellos y el periodo de envío es configurable. Los datos quedan etiquetados con la marca de tiempo al ser insertados en la BD.

V. Metodología

Esta sección describimos la carga de trabajo empleada y el concepto de afinidad que permite estresar núcleos concretos y desactivar el resto.

A. Carga de Trabajo

La plataforma permite ejecutar cualquier carga de trabajo o *benchmark* que resulte de interés. En este trabajos usamos dos tipos. Por un lado, se utilizan las pruebas integradas en **Intel XTU**, útiles para comprobaciones interactivas rápidas y para validar la estabilidad del sistema en puntos de operación extremos desde el propio entorno Windows. Por otro lado, se recurre a las *prueba de tortura* (*torture test*) de **Prime95**, un estándar en la industria para verificar la estabilidad del hardware, que permite generar una carga computacional sostenida, tanto en pruebas interactivas como desde la línea de comandos [21,22].

Prime95 se basa en la ejecución del algoritmo de Lucas-Lehmer para verificar la primalidad de un número de Mersenne (número de la forma $M_p = 2^p - 1$, donde p a su vez es primo). Prime95 ejecuta de forma iterativa transformadas de Fourier rápidas (FFTs) que implican una carga aritmética sostenida de multiplicaciones vectorizadas en coma flotante. En cada iteración se comprueba la corrección aritmética, y si se detecta error, se interpreta que el procesador ha fallado, ya sea por falta de voltaje o exceso de temperatura, se detiene el trabajo y se notifica el fallo [23].

El usuario puede escoger tres opciones: a) FFTs Pequeñas donde los datos caben en los primeros niveles de cache, b) FFTs Grandes donde los datos no caben ni siquiera en las caches de último nivel y es necesario acceder continuamente a memoria principal y c) un método Mezcla (*Blend*) que alterna de forma cíclica entre los dos anteriores y es el estándar para comprobar la estabilidad general del sistema. En la práctica, esto permite generar escenarios experimentales reproducibles con distinto énfasis en cómputo, disipación térmica y memoria.

B. Afinidad

La migración de procesos es una herramienta muy útil del planificador del SO para prevenir sobrecargas térmicas, pero en procesadores híbridos como el 265KF, donde los P-cores y los E-cores presentan distinto comportamiento en frecuencia, consumo y temperatura, incluso existe hardware dedicado para guiar la migración y perseguir buenos compromisos rendimiento/consumo, como el *Intel Thread Director*.

Ahora bien, para experimentar de forma reproducible hay que inhibir la migración y tomar el control de la asignación proceso-núcleo. Para ello se utiliza la "afinidad" de procesos o hilos a núcleos. Si la afinidad se limita a un único núcleo, el proceso queda anclado a él; si se extiende a un grupo de núcleos, el SO puede asignarlo dinámicamente dentro de ese grupo, pero no fuera de él.

En Windows, la afinidad se controla desde el propio SO o desde las funciones de afinidad de proceso o hilo, por ejemplo, desde Python mediante *ProcessorAffinity* [24]; en Linux, mediante `sched_setaffinity()` o herramientas equivalentes como *taskset*. En ambos casos, la selección real se realiza sobre identificadores lógicos, por lo que es necesario mapear previamente qué identificadores corresponden a cada núcleo físico o a cada tipo de núcleo.

VI. Resultados Experimentales

En esta sección presentamos tres experimentos. En primer lugar, se presenta un experimento de inferencia térmica que permite reconstruir la disposición relativa de P-cores y E-cores a partir de su respuesta térmica. A continuación, se estudian la curva voltaje-frecuencia de cada núcleo para identificar puntos de operación estables. Finalmente, se analiza la respuesta dinámica del sistema en dos regímenes extremos de refrigeración con el fin de observar de forma conjunta la generación y extracción de calor.

A. Inferencia Térmica del Mapa de Núcleos

Como primer resultado, y a modo de **prueba de concepto** del sistema de medida, se realizó un experimento sencillo pero ilustrativo: inferir la correspondencia lógica-física de los P-cores y E-cores en el procesador Intel Core Ultra 7 265KF, únicamente a partir de su **respuesta térmica**.

En el momento de realizar este trabajo no conocemos documento oficial de Intel que relacione la numeración lógica de los núcleos con su posición dentro del chip, por lo que este experimento permite, además de explorar la arquitectura del procesador, validar que la instrumentación es capaz de capturar **interacciones térmicas locales** entre núcleos próximos.

El procedimiento consiste en ejecutar un único *worker* de Prime95 fijado mediante afinidad a un núcleo concreto mientras se registran las temperaturas. Repitiendo el ensayo sobre distintos P-cores y E-cores, se observa no solo el aumento de temperatura del núcleo cargado, sino también la respuesta secundaria del resto. La idea básica es que los incrementos más rápidos y más intensos en otros núcleos se correlacionan con la **vecindad física**. Por ejemplo, cuando se carga P4 su temperatura sube hasta

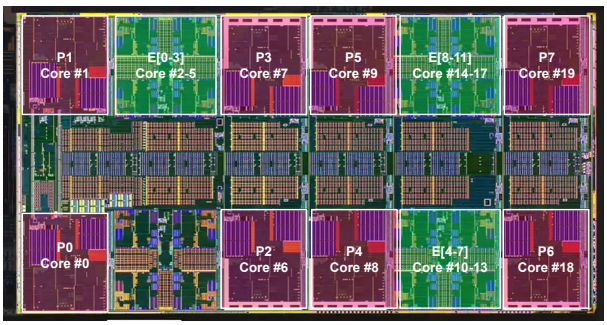


Fig. 3: Mapa físico estimado de P-cores y E-cores del Intel Core Ultra 7 265KF. En cada bloque se indica tanto la denominación empleada en este trabajo (e.g., P1 o E[0–3]) como el identificador lógico expuesto por el SO (e.g., *Core #1*).

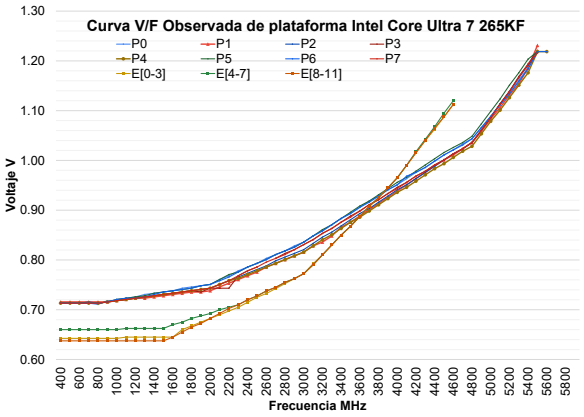


Fig. 4: Curvas V/F observadas en los P-cores y en los tres clústeres de E-cores del procesador Intel Core Ultra 7 265KF.

66 ºC, mientras que P2 y E6 muestran una respuesta secundaria de 37 ºC, con el resto del grupo E4–E7 en torno a 36 ºC. Los demás núcleos permanecen cerca de 31 ºC. A partir de la comparación sistemática de estos patrones fue posible **reconstruir el mapa** mostrado en la Figura 3.

El resultado obtenido permite distinguir con claridad los **clústeres de E-cores** y su posición relativa respecto a los P-cores en nuestro sistema, así como comprobar que la numeración lógica expuesta por el sistema no sigue una disposición espacial evidente. Aunque este procedimiento no permite fijar de forma unívoca la orientación absoluta de este conjunto de 12 chiplets —por ejemplo, pueden existir soluciones equivalentes bajo simetrías especulares—, sí expone de forma robusta la estructura de **vecindad térmica**. Por ello, este experimento constituye una curiosidad útil desde el punto de vista arquitectónico y, al mismo tiempo, una validación visual y fácil de interpretar de la resolución dinámica alcanzada por Forge2.

B. Curvas V/F

La relación entre voltaje y frecuencia constituye una de las características más relevantes del **punto de operación** de un procesador, ya que condiciona de forma directa tanto su consumo, que es exactamente la potencia térmica disipada. En consecuencia, disponer de una aproximación de las **curvas V/F** resulta de interés para conocer la "calidad del silicio" de cada chiplet (*silicon quality*) y observar la variabilidad de fabricación.

En generaciones previas de procesadores Intel desbloqueados era posible acceder con facilidad a las curvas V/F programadas de fábrica para cada core (un cierto número de parejas V/F sobre las que se interpola). Sin embargo, en la plataforma utilizada esta información no aparece directamente en términos absolutos, sino a través de mecanismos de ajuste de ***offsets*** accesibles desde UEFI o XTU. En este experimento no se modifican dichos *offsets*, de modo que el objetivo no es reconstruir la curva interna exacta definida por el fabricante, sino obtener las **curvas V/F observadas** a partir del comportamiento real de la plataforma.

Para ello, se registran de forma simultánea la frecuencia y el voltaje observados en los distintos P-cores y clústeres de E-cores durante un intervalo de operación controlado. A partir de estas muestras, y tras descartar valores espurios, se conservan para cada frecuencia los **pares V/F de voltaje mínimo**, obteniendo así la aproximación experimental a la curvas programadas que se muestra en la Figura 4.

Los resultados muestran, en primer lugar, que los P-cores presentan curvas muy próximas entre sí, con diferencias reducidas incluso en la zona de frecuencias más elevadas. Por su parte, los tres clústeres de E-cores muestran un comportamiento claramente diferenciado respecto a los P-cores, desplazado hacia tensiones y frecuencias iniciales inferiores, aunque en la parte alta del rango de frecuencias alcanzan tensiones superiores a las observadas en los P-cores.

Centrándonos en los clústeres de E-cores, el grupo E[4–7] presenta un pequeño desfase respecto a los otros dos clústeres a bajas frecuencias, mientras que en la parte alta de la curva las diferencias entre grupos se reducen. En cuanto a los P-cores, procede decir que en nuestro chip, P1 y P4 se han identificado por Intel en fabricación como *favored cores*, y resulta que observamos que se sitúan entre los de menor voltaje para una misma frecuencia, aunque sin una separación realmente marcada respecto a P2, P3 o P7.

Los resultados obtenidos sugieren además que la relación V/F no es lineal, sino más bien cuadrática. Observamos que aparecen varios **cambios de pendiente** compatibles con una **definición discreta** de puntos V/F y una **interpolación** entre ellos. Hasta donde sabemos, ni Intel ni la UEFI exponen de forma explícita la correspondencia exacta entre cada punto y su ratio asociado; únicamente se observan puntos numerados sobre los que puede aplicarse un *offset*. Experimentalmente, en los P-cores estos cambios de pendiente se aprecian en torno a 800, 2000, 3000, 4800, 5400 y 5500 MHz, con indicios de un punto adicional intermedio alrededor de 5200 MHz. En los clústeres de E-cores, los cambios de pendiente se observan en 800, 1500, 2300, 3000, 4000 y 4600 MHz.

En conjunto, este experimento proporciona una referencia útil para interpretar resultados posteriores y

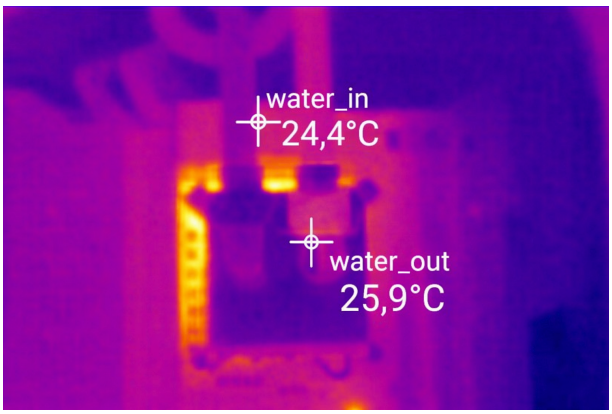


Fig. 5: Termografía de Forge2 durante operación, mostrando el bloque de CPU, la zona de regulación de potencia de la placa base y dos puntos de medida de temperatura del líquido, a la entrada (`water_in`) y a la salida (`water_out`) del bloque.

para seleccionar puntos de operación representativos en ensayos térmicos y energéticos. Más que una reconstrucción exacta de la tablas V/F de los núcleos, las curvas obtenidas deben entenderse como una **caracterización experimental** de nuestro Intel Core Ultra 7 265KF.

C. Respuesta Dinámica Cambiando el Régimen de Refrigeración

Con el fin de caracterizar la **capacidad de extracción de calor** de la plataforma, se plantea un experimento que fija un **punto de operación** único del procesador y compara su respuesta térmica bajo dos **regímenes extremos de refrigeración**. En ambos casos, el ensayo parte de una **temperatura basal del agua** similar y la carga aplicada consiste en 20 *workers* de Prime95 con FFT pequeñas, de alta localidad, manteniendo constante el punto de operación del procesador durante todo el ensayo.

La Figura 5 muestra una termografía de la plataforma durante operación, en la que se distingue el bloque de CPU, la zona de regulación de potencia de la placa base y los puntos de medida de temperatura del líquido a la entrada y a la salida del bloque. Esta imagen permite contextualizar el experimento y visualizar la distribución térmica de los elementos más relevantes del sistema.

En una primera fase, la bomba y los ventiladores del intercambiador funcionan a su régimen máximo, registrando la evolución temporal del sistema hasta alcanzar el **equilibrio térmico**. A continuación, se enfría de nuevo el circuito hasta recuperar una temperatura basal del agua próxima a 26 °C y se repite el mismo ensayo con la bomba y los ventiladores en su régimen mínimo estable. En todo momento se monitoriza la temperatura del agua a la entrada y a la salida del bloque, el caudal del circuito, las revoluciones por minuto de la bomba y de los ventiladores, las temperaturas del aire antes y después del intercambiador y la **telemetría interna del procesador**, incluyendo temperaturas por núcleo y *package*, frecuencia y voltaje por núcleo, potencia del *package*, indicadores de *throttling* térmico, instrucciones retiradas, ciclos activos e IPC por núcleo.

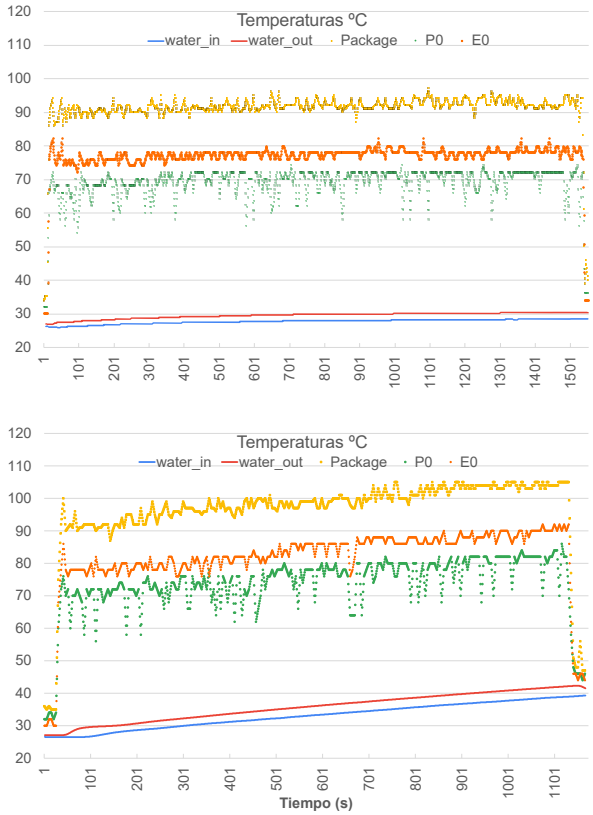


Fig. 6: Evolución temporal de temperaturas representativas bajo dos regímenes de refrigeración (eje X en segundos). Arriba: régimen de refrigeración máxima. Abajo: régimen de refrigeración mínima.

En ambos ensayos se observa un cambio brusco del punto de operación del procesador al aplicarse la carga. Antes del inicio, los núcleos se mantienen en reposo alrededor de 400 MHz, con tensiones próximas a 0,71 V en los P-cores y 0,64 V en los E-cores. Tras aplicar la carga, las frecuencias se estabilizan en torno a 5200 MHz para los P-cores y 4400 MHz para los E-cores, de acuerdo con los límites previamente fijados en XTU, mientras que los voltajes ascienden aproximadamente hasta 1,17 V y 1,10 V, respectivamente. De forma coherente con ello, el IPC pasa de valores prácticamente nulos en reposo a valores medios de 3,57 en los P-cores y 2,22 en los E-cores, mientras que la potencia del *package* aumenta desde unos 20 W en reposo hasta aproximadamente 272 W en régimen sostenido, con un pico inicial puntual de 306 W. La temperatura del aire ambiente durante los ensayos se mantuvo a 19,7 °C.

Las diferencias entre ambos casos aparecen principalmente en la **capacidad de extracción de calor**. En el régimen mínimo, la bomba gira a 2166 r.p.m. y los ventiladores a 291 r.p.m., con un caudal que aumenta de aproximadamente 0,7 a 0,8 L/min conforme se calienta el líquido. En el régimen máximo, la bomba gira a 4450 r.p.m. y los ventiladores a 2750 r.p.m. manteniéndose el caudal en torno a 2,6 L/min.

La Figura 6 muestra la evolución temporal de temperaturas representativas del procesador y del circui-

to líquido diferenciando entre regímenes de refrigeración máxima y mínima. En cada caso se representa la temperatura del *package*, la de un P-core y un E-core representativos, así como la temperatura del agua a la entrada y a la salida del bloque. En los resultados que siguen, el instante inicial segundo $t = 0$ corresponde a la aplicación de la carga una vez estabilizada la temperatura basal del agua.

Bajo **refrigeración máxima**, la temperatura del procesador experimenta un incremento rápido tras la aplicación de la carga y alcanza un régimen casi estacionario en un intervalo corto de tiempo. En cambio, la temperatura del agua evoluciona de forma mucho más lenta, aumentando progresivamente durante varios minutos hasta aproximarse a su nuevo **equilibrio térmico** (entrada a 28,5, salida a 30,4 °C). La diferencia entre la respuesta casi inmediata del *package* y la evolución más lenta del líquido pone de manifiesto la distinta **constante de tiempo** del procesador y del circuito de refrigeración. Durante este ensayo, la elevada velocidad de la bomba y ventiladores permite mantener la temperatura del procesador en una zona estable, mientras que el agua presenta un gradiente moderado entre entrada y salida.

Bajo **refrigeración mínima**, la temperatura del *package* aumenta rápidamente hasta valores próximos a 105 °C, y algunos núcleos comienzan a mostrar episodios de ***throttling* térmico** a partir del segundo 400 y generalizados desde el 800. Al mismo tiempo, la temperatura del agua sigue aumentando durante todo el ensayo **sin alcanzar un equilibrio térmico** claro, lo que indica que la capacidad de extracción de calor del sistema resulta insuficiente para sostener de forma estable este punto de operación.

Globalmente, los dos ensayos delimitan con claridad el rango de funcionamiento térmico de la plataforma para este punto de operación del procesador. Bajo refrigeración máxima, el sistema es capaz de evacuar de forma estable la potencia disipada, manteniendo la temperatura del procesador en una zona cuasi estacionaria mientras el circuito líquido evoluciona más lentamente hacia su equilibrio. En cambio, bajo refrigeración mínima, la capacidad de extracción de calor resulta insuficiente: la temperatura del *package* alcanza valores próximos al límite térmico, aparecen situaciones de *throttling* térmico y la temperatura del agua sigue creciendo sin llegar a estabilizarse dentro del intervalo observado. Estos resultados ponen de manifiesto tanto la distinta constante de tiempo del procesador y del circuito líquido como la fuerte influencia del caudal y de la ventilación sobre la **estabilidad térmica** del sistema.

VII. Trabajos Relacionados

La caracterización térmica de procesadores ha sido abordada mediante enfoques experimentales que combinan medición directa, análisis dinámico bajo distintas cargas de trabajo e instrumentación térmica integrada. En este contexto, Mesa-Martínez *et al.* muestran la naturaleza dinámica de los perfiles térmicos y las limitaciones de los modelos estáticos [25], mientras que Rotem *et al.* describen el papel de los sensores *on-die* y su relación con consumo y mecanismos de gestión energética [26]. Complementariamente, otros trabajos han empleado termografía infrarroja de alta resolución para identificar *hotspots* y estudiar el acoplamiento térmico entre bloques funcionales en procesadores heterogéneos [27]. En conjunto, estas aproximaciones constituyen una base sólida para la validación de modelos térmicos y el diseño de estrategias avanzadas de gestión térmica.

Los antecedentes más directos de Forge2 dentro del propio grupo se remontan a trabajos previos sobre docencia e instrumentación energética en sistemas comerciales. Asín-Pérez *et al.* propusieron la introducción de conceptos de potencia y energía en laboratorios de arquitectura de computadores [28]. Posteriormente, Gutiérrez *et al.* presentaron **Forge**, una plataforma multipropósito para la medida de energía y temperatura en PCs convencionales [29]. Finalmente, Gutiérrez-Verde *et al.* profundizaron en esta línea con una aproximación práctica al estudio de energía y temperatura en cursos de arquitectura [30].

Finalmente, se han explorado sistemas de refrigeración basados en materiales PCM como alternativa para la gestión térmica de dispositivos electrónicos de alta densidad de potencia. En este sentido, Kim *et al.* han demostrado que la integración de PCM en disipadores permite amortiguar transitorios térmicos y reducir tanto la temperatura máxima como las oscilaciones térmicas bajo cargas dinámicas [31,32]. En definitiva, estos trabajos confirman que los PCM contribuyen a reducir *hotspots* y mejorar la estabilidad térmica, si bien su aplicación en procesadores comerciales de altas prestaciones sigue siendo limitada debido a restricciones experimentales y a la baja conductividad térmica inherente a estos materiales.

VIII. Conclusiones y Trabajo Futuro

Este trabajo ha presentado **Forge2**, una **plataforma experimental** para el control, medida, registro y visualización de variables térmicas, eléctricas y de rendimiento en un procesador Intel Core Ultra desbloqueado con refrigeración líquida directa. La plataforma integra control del punto de operación del procesador y del subsistema de refrigeración, junto con instrumentación adicional propia y herramientas software capaces de unificar series temporales procedentes de fuentes distintas.

Los resultados obtenidos muestran que Forge2 permite caracterizar de forma útil el comportamiento térmico y energético de la plataforma. El experimento de inferencia térmica permite reconstruir de forma aproximada la disposición relativa de P-cores y E-cores a partir de su **respuesta térmica**. La caracterización de la **curva V/F** observada proporciona una referencia experimental de los puntos de operación de núcleos y clústeres. Finalmente, el estudio dinámico bajo dos regímenes extremos de **refrigeración** pone de manifiesto la fuerte influencia del caudal y de la ventilación sobre la estabilidad térmica del sistema, delimitando con claridad un régimen

de funcionamiento estable y otro en el que aparecen episodios de ***throttling* térmico**.

Como trabajo futuro, el objetivo principal es emplear Forge2 como banco de ensayos para estudiar **emulsiones con cambio de fase** como alternativa de refrigeración líquida directa. Además, se plantea profundizar en la caracterización energética del procesador, incluyendo el efecto de la temperatura sobre la potencia estática y la eficiencia computacional.

Agradecimientos

Este trabajo ha sido financiado por el PROGRAMA IMPULSO 2025: acciones multidisciplinares, seminales, para investigadores/as del I3A y por MICIU/AEI/10.13039/501100011033 (PID2022-136454NB-C22). Los autores agradecen a Mónica Delgado, Ana Lázaro y Pablo Castellanos, compañeras/ro del Grupo de Ingeniería Térmica y Sistemas Energéticos (GITSE) de la Universidad de Zaragoza, por su colaboración y respaldo en el desarrollo de este trabajo.

Referencias

[1] A. Tirumala and R. Wong. Nvidia blackwell platform: Advancing generative ai and accelerated computing. In *2024 IEEE Hot Chips 36 Symposium (HCS)*. IEEE, 2024.

[2] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, page 2–13, 2003.

[3] R. Kong, H. Zhang, M. Tang, H. Zou, C. Tian, and T. Ding. Enhancing data center cooling efficiency and ability: A comprehensive review of direct liquid cooling technologies. *Energy*, 308:132846, 2024.

[4] M. Delgado, A. Lázaro, J. Mazo, and B. Zalba. Review on phase change material emulsions and microencapsulated phase change material slurries: Materials, heat transfer studies and applications. *Renewable and Sustainable Energy Reviews*, 16(1):253–273, 2012.

[5] Intel. Intel® Core™ Ultra 7 Processor 265KF Specifications. `https://www.intel.com/content/www/us/en/products/sku/241062/intel-core-ultra-7-processor-265kf-30m-cache-up-to-5-50-ghz/specifications.html`, 2024. Intel ARK, accessed 2026-03-26.

[6] Intel. Intel® Core™ Ultra Desktop Processors (Series 2) Product Brief. `https://www.intel.com/content/www/us/en/products/docs/processors/core-ultra/core-ultra-desktop-processors-series-2-brief.html`, 2024. Intel Product Brief, accessed 2026-03-26.

[7] Intel. Intel® Core™ Ultra 9 Processor 285K Specifications. `https://www.intel.com/content/www/us/en/products/sku/241060/intel-core-ultra-9-processor-285k-36m-cache-up-to-5-70-ghz/specifications.html`, 2024. Intel ARK, accessed 2026-03-26.

[8] ASUS. PRIME Z890-P WIFI - Tech Specs. `https://www.asus.com/es/motherboards-components/motherboards/prime/prime-z890-p-wifi/techspec/`. Product specifications, accessed 2026-03-26.

[9] Alphacool. Alphacool Core Hurrican 240mm XT45 HardTube Water Cooling Set. `https://shop.alphacool.com/en/shop/water-cooling-sets/internal-sets/11992-alphacool-core-hurrican-240mm-xt45-hardtube-water-cooling-set`. Product page, accessed 2026-03-26.

[10] Alphacool. Alphacool ES Flow and Temperature Indicator HighFlow with RPM-Signal. `https://shop.alphacool.com/en/shop/controllers-and-sensors/flow-sensor/17558-alphacool-es-flow-and-temperature-indicator-highflow-with-rpm-signal`. Product page, accessed 2026-03-26.

[11] Alphacool. Alphacool Eiszapfen Temperature Sensor Plug G1/4 - Deep Black. `https://shop.alphacool.com/en/shop/controllers-and-sensors/temperature-sensor/sen-alphacool-eiszapfen-temperature-sensor-plug-g1/4-deep-black`. Product page, accessed 2026-03-26.

[12] ASUS. PRIME / ProArt / TUF GAMING Intel 800 Series BIOS Manual. `https://dlcdnets.asus.com/pub/ASUS/mb/13MANUAL/E25827_Intel_800_Series_BIOS_manual_EM_WEB.pdf?model=PRIME+Z890-P`, 2025. Official BIOS manual, accessed 2026-04-02.

[13] Intel. Intel® Extreme Tuning Utility (Intel® XTU). `https://www.intel.com/content/www/us/en/download/17881/intel-extreme-tuning-utility-intel-xtu.html`, 2026. Intel Download Center, accessed 2026-03-27.

[14] Intel. Support for Intel® Extreme Tuning Utility (Intel® XTU). `https://www.intel.com/content/www/us/en/support/products/66427/processors/processor-utilities-and-programs/intel-extreme-tuning-utility-intel-xtu.html`, 2026. Intel Support, accessed 2026-03-27.

[15] Rémi Mercier. Fan Control. `https://getfancontrol.com/`. Official website, accessed 2026-03-27.

[16] HWiNFO. HWiNFO® - System Information, Monitoring and Diagnostics. `https://www.hwinfo.com/`. Official website, accessed 2026-03-27.

[17] HWiNFO. HWiNFO® SDK. `https://www.hwinfo.com/sdk/`. Official SDK page, accessed 2026-03-27.

[18] Intel. Intel® Performance Counter Monitor (Intel® PCM). `https://github.com/intel/pcm`. Official repository, accessed 2026-03-27.

[19] InfluxData. InfluxDB. `https://www.influxdata.com/time-series-platform/`. Official website, accessed 2026-03-27.

[20] Grafana Labs. Grafana. `https://grafana.com/grafana/`. Official website, accessed 2026-03-27.

[21] GIMPS. Prime95 Download. `https://www.mersenne.org/download/`. Official download page, accessed 2026-03-27.

[22] GIMPS. Prime95 Readme. `https://www.mersenne.org/download/readme.txt`. Official documentation, accessed 2026-03-29.

[23] GIMPS. Prime95 Stress Test Notes. `https://www.mersenne.org/download/stress.txt`. Official documentation, accessed 2026-03-29.

[24] Microsoft. SetProcessAffinityMask function (winbase.h) - Win32 apps. `https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-setprocessaffinitymask`, 2022. Microsoft Learn, accessed 2026-03-26.

[25] F. J. Mesa-Martinez, E. K. Ardestani, and J. Renau. Characterizing processor thermal behavior. In *Proceedings of the Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, page 193–204, 2010.

[26] E. Rotem, J. Hermerding, A. Cohen, and H. Cain. Temperature measurement in the Intel(R) CoreTM Duo Processor. *THERMINIC 2006 hal-00171349*, pages 23–27, 2006.

[27] K. Dev, I. Paul, W. Huang, Y. Eckert, W. P. Burleson, and S. Reda. Implications of Integrated CPU-GPU Processors on Thermal and Power Management Techniques. *ArXiv*, abs/1808.09651, 2018.

[28] A. Asín-Pérez, D. Suárez-Gracia, and V. Viñals-Yúfera. A proposal to introduce power and energy notions in computer architecture laboratories. In *Proceedings of the 2007 Workshop on Computer Architecture Education*, page 52–57, New York, NY, USA, 2007. Association for Computing Machinery.

[29] S. Gutierrez, O. Benedí, D. Suárez, J. M. Marín, and V. Viñals. Forge: A Multi-purpose Platform for Measuring Energy and Temperature in Commodity PCs. In *Actas de las XX Jornadas de Paralelismo*, pages 403–408, 2009.

[30] S. Gutiérrez-Verde, O. Benedí-Sánchez, D. Suárez-Gracia, J. M. Marín-Herrero, and V. Viñals-Yúfera. Processor Energy and Temperature in Computer Architecture Courses: a hands-on approach. In *Proceedings of the 2009 Workshop on Computer Architecture Education*, 2009.

[31] S. Kim, T. Yang, N. Miljkovic, and W. P. King. Phase change material integrated cooling for transient thermal management of electronic devices. *International Journal of Heat and Mass Transfer*, 213:124263, 2023.

[32] A. Ghanbarpour, M.J. Hosseini, A.A. Ranjbar, M. Rahimi, R. Bahrampoury, and M. Ghanbarpour. Evaluation of heat sink performance using PCM and vapor chamber/heat pipe. *Renewable Energy*, 163:698–719, 2021.

Framework Paralelo para Carga y Aumento de Datos durante el Entrenamiento de Modelos de Deep Learning

Antonio De Toro-Castro[1], Marcos Lupión[2], Vicente González-Ruiz[1], Juan F. Sanjuan[1] y Pilar M. Ortigosa[1]

Resumen— **La ingesta eficiente de datos y el aumento de datos online continúan siendo desafíos en los flujos de trabajo de aprendizaje profundo, especialmente cuando se manejan conjuntos de datos con formatos no estándar o listas multidimensionales masivas que las funciones nativamente optimizadas no pueden gestionar completamente. Este trabajo presenta un framework paralelo que integra `multiprocessing` y memoria compartida global mediante una arquitectura de ring buffer, permitiendo la carga de alto rendimiento y un aumento de datos online y flexible. El framework desacopla la producción de datos del consumo, permitiendo que múltiples workers de CPU carguen y preprocesen lotes en paralelo, evitando completamente el bloqueo del intérprete global de Python y los cuellos de botella de memoria.**

La validación del enfoque se llevó a cabo sobre dos tareas representativas: reconocimiento de lengua de signos a partir de secuencias de pose humana en formato CSV, y clasificación de imágenes hiperespectrales. Frente a pipelines paralelos nativamente optimizados de TensorFlow y PyTorch, el framework obtuvo factores de aceleración de hasta $\times 8$ en la carga de datos y de hasta $\times 7$ en el entrenamiento completo en escenarios con alta demanda de memoria.

En general, el framework propuesto proporciona una solución escalable y compatible con multi-GPU para pipelines de aprendizaje profundo, mostrando un rendimiento robusto tanto en escenarios limitados por entrada/salida como por memoria en TensorFlow y PyTorch, reduciendo la fragmentación de memoria y las restricciones de asignación.

Palabras clave— **Deep Learning, Pipeline de Entrada, Aumento de Datos, Multiprocesamiento, Computación de Alto Rendimiento.**

I. Introducción

El Deep Learning (DL) emula algunos aspectos del procesamiento cerebral humano para abordar problemas complejos, como la clasificación de imágenes, la comprensión del lenguaje natural o la generación de datos sintéticos [1]. El principal elemento del DL son las Neural Networks (NNs) artificiales compuestas de unidades de procesamiento interconectadas conocidas como neuronas. Los frameworks especializados de DL, como TensorFlow (TF) [2] y PyTorch (PT) [3], forman parte de la mayoría de las soluciones modernas basadas en NNs [4].

Una de las etapas más críticas en DL es el pipeline de entrada de datos usado durante el entrenamiento del modelo [2], [3]. Típicamente, los datos se almacenan en disco y deben leerse, preprocesarse y convertirse a un formato adecuado antes de transferirse a la Graphical Processing Unit (GPU). Para conjuntos de datos pequeños, los frameworks a menudo pueden precargar los datos en memoria principal, manteniendo alta utilización y rendimiento computacional. Sin embargo, este enfoque se vuelve inviable para conjuntos de datos a gran escala [5]. En estos casos, los datos deben transmitirse y procesarse bajo demanda.

TF utiliza la Application Programming Interface (API) `tf.data`, basada en un diseño funcional que emplea `multithreading` y precarga asíncrona, mientras que PT adopta la clase `DataLoader`, que recurre al `multiprocessing` de Python para paralelizar la preparación de lotes y evitar las limitaciones del Global Interpreter Lock (GIL). Sin embargo, estas soluciones están principalmente diseñadas para formatos de datos estándar. Cuando se manejan entradas más complejas, los desarrolladores suelen recurrir a código de preprocesamiento personalizado fuera del grafo de computación, lo que puede provocar cuellos de botella y dejar la GPU infrautilizada [4].

Otra etapa crítica en DL es el Data Augmentation (DA), que mejora la generalización del modelo aplicando transformaciones como rotaciones, zoom o escalados [6], [7]. Aunque el DA offline es común, es ineficiente porque aumenta el uso de almacenamiento. Por lo tanto, los pipelines modernos dependen cada vez más del DA online [7].

En visión por computador, el DA online se implementa típicamente mediante librerías y operadores nativos del framework [8], [9]. Sin embargo, son menos adecuados cuando los aumentos requieren operaciones específicas del problema, fuera de las transformaciones estándar en el grafo, como la carga de archivos a nivel de secuencia o la manipulación de coordenadas geométricas. En estos escenarios, el DA online suele recaer en la Central Processing Unit (CPU) y se convierte en parte del pipeline de entrada, donde puede aumentar significativamente la latencia de preprocesamiento y dejar la GPU infrautilizada [4], [10].

En este trabajo, presentamos un framework modular y extensible para la ingesta de alto rendimiento y DA online en DL para formatos de datos no estándar. Para motivar nuestro enfoque, analizamos dos tareas representativas: reconocimiento de lengua de signos basado en pose [11] y clasificación de imágenes hiperespectrales [12].

Las principales contribuciones de este artículo son:

- Diseñamos un framework de pipeline de en-

[1]Dpto. de Informática, Universidad de Almería, CEINSA, e-mails: {antoniodetoro,vruiz,jsanjuan,ortigosa}@ual.es
[2]School of Computing, Ulster University, e-mail: m.lupionlorente@ulster.ac.uk

trada paralelo basado en `multiprocessing` de Python, memoria compartida Portable Operating System Interface (POSIX) y arquitectura ring buffer que desacopla los productores de datos del proceso de entrenamiento.
- El framework soporta DA online definido por el usuario directamente en elementos sin procesar, permitiendo transformaciones específicas del dominio para entradas no estándar y DA basada en GPU cuando es posible.
- El sistema propuesto expone parámetros ajustables y puede integrarse en configuraciones de entrenamiento con una o múltiples GPUs.
- El framework opera como una extensión plug-and-play de los mecanismos de pipeline de datos de TF, manteniendo la compatibilidad con `tf.data`.
- Demostramos la efectividad del enfoque propuesto en reconocimiento de lengua de signos basado en pose y clasificación de imágenes hiperespectrales.

El resto del artículo se organiza así: La Sección II analiza la literatura existente; la Sección III describe la solución propuesta; los resultados experimentales se exponen en la Sección IV; finalmente, las conclusiones se explican en la Sección V.

II. Trabajos Relacionados

Se han propuesto varios enfoques para mejorar la eficiencia de los flujos de trabajo de DL, particularmente en carga y aumento de datos. En esta sección, examinamos los principales frameworks que optimizan la ingesta de datos y revisamos estrategias de DA.

A. Pipeline de Entrada de Datos

A.1 TensorFlow y PyTorch

La carga de datos en TF y PT a menudo se convierte en un cuello de botella de rendimiento importante, principalmente debido a la disparidad entre las velocidades de procesamiento de GPU y las tareas más lentas ligadas a CPU de Input/Output (I/O) de disco y preprocesamiento de datos. El enfoque más directo para mitigar esto es cargar todo el conjunto de datos en memoria del sistema. Sin embargo, rápidamente se vuelve impráctico para conjuntos de datos a gran escala debido a errores Out-Of-Memory (OOM).

Para procesar conjuntos de datos que exceden la Random Access Memory (RAM) disponible, TF se basa en la API `tf.data` [13]. Este framework construye pipelines componibles y altamente optimizados. Para ejecutar estas operaciones concurrentemente, TF emplea `multithreading`. Si bien los hilos son eficientes en memoria, están limitados por el GIL cuando se ejecuta código Python personalizado o DA específico del dominio fuera del grafo compilado de TF.

Por el contrario, PT aborda nativamente esta limitación del GIL utilizando una arquitectura `multiprocessing` dentro de su clase `DataLoader`. Al generar procesos worker independientes en lugar de hilos, PT evita el GIL. Sin embargo, este enfoque introduce overhead de Inter-Process Communication (IPC) y cada proceso worker asigna independientemente la memoria requerida, lo que frecuentemente conduce a copias de datos redundantes y picos inesperados en el consumo total de RAM.

B. Técnicas de Aumento de Datos

Los enfoques tradicionales de DA online se basan en librerías como `imgaug` [8] y Albumentations [9], que aplican transformaciones directamente a datos en crudo en tiempo de ejecución. Sin embargo, típicamente están ligadas a CPU y operan fuera del grafo de computación.

Para superar estas limitaciones, versiones recientes de TF y PT proporcionan operadores y capas nativas de aumentos que pueden integrarse directamente en el grafo de computación y ejecutarse en la GPU. Frameworks como Kornia [14] y NVIDIA DALI [15] realizan transformaciones diferenciables por lotes directamente en memoria del dispositivo.

Sin embargo, a pesar de esta tendencia hacia preprocesamiento acelerado por GPU, el DA del lado de la CPU sigue siendo indispensable en muchos escenarios prácticos. Mientras que los operadores basados en GPU son altamente efectivos para transformaciones afines estándar en entradas de tensor regulares, son menos adecuados para modalidades irregulares o no estándar como secuencias temporales de keypoints.

III. Metodología

Esta sección presenta la metodología adoptada para construir un pipeline de entrada de datos eficiente y escalable para DL. Primero, se introduce el ring buffer paralelo propuesto. Segundo, se describe la integración del buffer con las funciones de TF. Finalmente, se explica el enfoque del framework para DA online.

A. Ring Buffer Paralelo

El framework propuesto aborda las limitaciones fundamentales de los pipelines de entrada de DL existentes implementando una arquitectura modular de alto rendimiento que desacopla la producción de datos del consumo mediante un sistema de ring buffer paralelo basado en memoria compartida. La arquitectura, ilustrada en la Figura 1, se acopla con las funciones estándar `tf.data` de TF, logrando una utilización óptima de GPU mientras soporta formatos de datos arbitrarios y estrategias de DA online.

El enfoque propuesto aprovecha las funciones del pipeline de entrada de TF, pero en lugar de gestionar el consumo de elementos a través de mecanismos de paralelismo subyacentes, esto se maneja a través de un `Parallel Ring Buffer`. Las funciones de TF instancian N objetos *reader*, cuyo único propósito es extraer datos del buffer circular. El buffer circular es poblado por procesos *worker* separados gestionados usando `multiprocessing` de Python. Cada *worker*

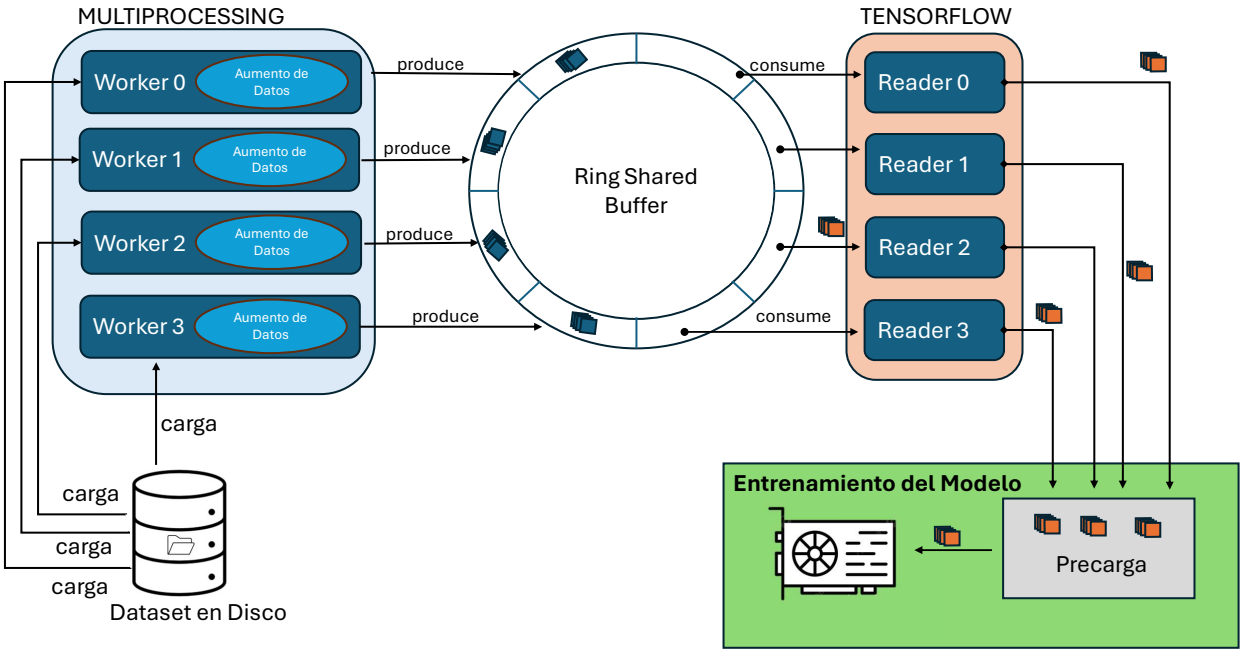


Fig. 1: Arquitectura general del sistema propuesto.

es responsable de cargar datos desde disco, aplicar DA y colocar los datos procesados en el buffer.

El componente principal de la arquitectura, el ring buffer paralelo basado en memoria compartida, se implementa en la clase Python personalizada `SharedRingPrefetched`. Gestiona un ring buffer mediante tres colas sincronizadas:

- **IN_Q**: Esta cola contiene los índices de los lotes (*BATCH IDs*) que necesitan cargarse.
- **FREE_Q**: Esta cola almacena los índices que referencian slots libres (*SLOT IDs*) en memoria compartida.
- **FULL_Q**: Esta cola contiene los *SLOT IDs* en memoria que contienen datos cargados, es decir, lotes listos para consumo.

La operación completa del buffer puede verse en detalle en la Figura 2. Primero, el proceso de entrenamiento genera lotes y los precarga en la cola *IN_Q*. En esta etapa, múltiples workers cargan los datos desde disco y realizan DA. Una vez completado este proceso, se obtiene un slot libre en memoria compartida usando la cola *FREE_Q*, almacenando el lote en el *SLOT ID* correspondiente y agregándolo a la cola de elementos procesados (*FULL_Q*). Concurrentemente, los procesos de la API `tf.data` adquieren datos de la cola *FULL_Q* siguiendo un esquema publicador/suscriptor.

A.1 Paralelismo

En este trabajo, el paralelismo se basó en la biblioteca `multiprocessing` de Python. Permite construir un pool de workers responsables de la carga y DA. A diferencia del módulo `threading`, que está limitado por el GIL, `multiprocessing` crea procesos independientes que pueden ejecutarse simultáneamente en múltiples núcleos de CPU.

Los workers actúan como productores paralelos en el pipeline de entrada. Cada worker es responsable de cargar independientemente una porción del conjunto de datos desde disco, aplicar las operaciones requeridas de preprocesamiento y DA, y luego escribir las muestras procesadas en memoria compartida. Al generar lotes a través de múltiples workers, el pipeline solapa el I/O de archivos con preprocesamiento intensivo en CPU, reduciendo así el tiempo de preparación de lotes.

A.2 Memoria Compartida

Cuando se usa el paquete `multiprocessing` de Python, se generan múltiples procesos para ejecutarse concurrentemente en los núcleos de CPU del sistema. Sin embargo, cada proceso se ejecuta como una instancia independiente del intérprete de Python, manteniendo su propio espacio de memoria aislado. Dado que los procesos de TF encargados de alimentar la GPU con nuevos datos operan en espacios de direcciones separados, transferir datos entre ellos implica serialización e IPC costosos.

Una solución a este problema es usar un bloque de RAM compartido entre los workers y los procesos de TF. Esto puede implementarse mediante el módulo `multiprocessing.shared_memory` de Python. Este módulo proporciona una interfaz de alto nivel para memoria compartida POSIX en Linux. Internamente, asigna un segmento de memoria en el sistema de archivos */dev/shm*, que es un *tmpfs* (es decir, un sistema de archivos respaldado por RAM en lugar de un disco físico). Como resultado, la región de memoria asignada reside completamente en memoria volátil del sistema, haciendo el acceso extremadamente rápido comparado con mecanismos IPC estándar.

La estructura de datos de memoria que habilita este esquema paralelo se basa en dos bloques de memoria compartida preasignados: uno dedicado a almacenar las muestras de datos preprocesadas y otro para sus etiquetas correspondientes. Cada uno de los

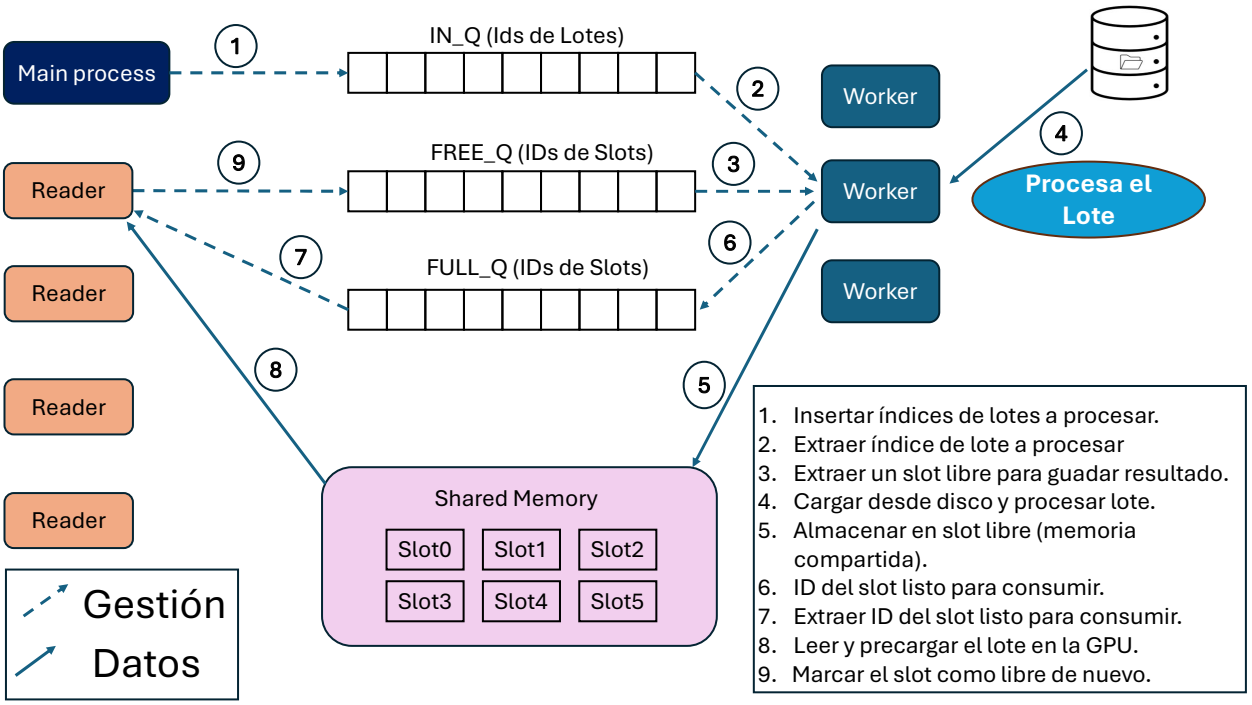


Fig. 2: Sistema multi-cola del ring buffer paralelo.

dos bloques de memoria se particiona lógicamente en *N_SLOTS* segmentos continuos, con cada slot dimensionado precisamente para contener un lote de entrenamiento completo.

Esta estrategia de preasignación estática presenta una ventaja sobre el enfoque `multiprocessing` nativo utilizado por el `DataLoader` de PT. Mientras que PT también aprovecha memoria compartida para evitar la sobrecarga de serialización, asigna y libera dinámicamente bloques de memoria para cada lote generado sobre la marcha. Por el contrario, el ring buffer propuesto reserva su memoria máxima una única vez al inicio. Durante el bucle de entrenamiento no se realiza ninguna asignación de memoria, y el IPC se reduce a únicamente la transmisión de índices enteros.

B. Integración en funciones de TensorFlow

La integración con el pipeline de datos de TF aprovecha varias funciones clave para crear un sistema de entrada eficiente y escalable que conecta sin problemas el ring buffer `multiprocessing` con el proceso de entrenamiento de GPU.

Primero, en lugar de crear solo un objeto dataset, `tf.data.Dataset` crea *NUM_READERS* procesos usando la función `range`. Luego, la función `tf.data.Dataset.interleave()` inicializa un nuevo dataset (también llamado *READER*) para cada uno de los procesos creados antes. Los datasets necesitan ser alimentados con datos. Para ello, se proporciona un generador a cada uno de ellos. Cada generador espera slots listos en la *FULL_Q* definida en el ring buffer prefetcher.

Además, el número total de épocas y lotes deben establecerse explícitamente usando las funciones `tf.data.Dataset.repeat()` y `tf.data.Dataset.take()`. Finalmente, se llama `tf.data.Dataset.prefetch(GPU_STAGE)` para almacenar *GPU_STAGE* lotes en memoria de GPU, asegurando que esta nunca tenga que esperar la transferencia de datos.

B.1 Integración Multi-GPU

Para abordar la escalabilidad de hardware dentro de un solo nodo, el framework propuesto presenta soporte robusto nativo para entrenamiento multi-GPU. Crucialmente, extender el framework a un entorno multi-GPU no requiere modificaciones estructurales al ring buffer de memoria compartida. En lugar de instanciar un pipeline de datos separado y ring buffer para cada GPU, se mantiene un solo ring buffer centralizado. La única adaptación necesaria es configurar el tamaño de slot del buffer para almacenar un *lote global*, calculado multiplicando el tamaño de lote deseado por GPU por el número total de réplicas de GPU disponibles.

La integración con el ecosistema de entrenamiento distribuido de TF se maneja sin problemas a través de la API `tf.data`. El ring buffer centralizado alimenta estos lotes globales completamente aumentados en un `tf.data.Dataset` estándar. Al aplicar `MirroredStrategy` de TF y establecer la directiva de distribución de dataset a `AutoShardPolicy.DATA`, el framework delega el particionamiento de datos completamente al backend nativo de TF.

C. Aumento de Datos Online

En este trabajo, en lugar de aplicar un conjunto fijo de transformaciones a cada muestra de datos, las especificaciones de DA se codifican e integran en el flujo de trabajo de carga de datos. El framework soporta funciones de transformación arbitrarias definidas por el usuario que pueden operar en cualquier estructura de datos.

Una dimensión de diseño importante del framework es dónde se ejecuta el DA. El framework está explícitamente diseñado para soportar estrategias basadas en CPU y GPU, y la elección depende de la naturaleza de los datos y las transformaciones requeridas.

C.1 Aumento de Datos en CPU

El DA del lado de la CPU es la única opción viable cuando las transformaciones involucran modalidades de datos irregulares o no estándar. En este caso, los workers ejecutándose en la CPU decodifican los parámetros de DA y lo realiza después de cargar los datos de entrada.

Un ejemplo representativo es el reconocimiento de lengua de signos basado en pose (Problema 1 en este trabajo), donde la entrada consiste en secuencias de coordenadas de keypoints 2D almacenadas en archivos Comma-Separated Values (CSV). La lógica de DA aplica desplazamientos espaciales controlados directamente a los valores de coordenadas en todos los frames de una secuencia, seguido del renderizado de las imágenes esqueléticas resultantes usando OpenCV. Este procedimiento de varias etapas involucra lógica condicional sobre datos estructurados, I/O a nivel de archivo y llamadas a librería que no pueden compilarse nativamente o eficientemente en operaciones de tensor de GPU sin kernels CUDA personalizados.

C.2 Aumento de Datos en GPU

El DA del lado de la GPU es preferible cuando los datos de entrada ya están en un formato de tensor regular compatible con capas nativas del framework. Las transformaciones geométricas estándar y perturbaciones espectrales pueden implementarse como `tf.keras.layers` diferenciables directamente dentro del grafo del modelo, ejecutándose en la GPU en forma de lote sin ninguna sobrecarga del lado del host.

Para el Problema 2 (clasificación de imágenes hiperespectrales), donde los datos se cargan como tensores densos `float32` de forma $48 \times 352 \times 204$, este enfoque es completamente aplicable.

Este soporte dual es clave en el framework propuesto: no impone una única estrategia de DA sino que permite a los usuarios colocar los aumentos de datos donde sean más eficientes para su modalidad de datos específica.

IV. Resultados Experimentales

Esta sección presenta una evaluación del framework paralelo propuesto. Todos los experimentos se realizan usando tareas representativas de DL que exponen las limitaciones de los enfoques existentes de pipeline de entrada.

A. Infraestructura HPC

Los experimentos se realizaron en el nodo High-Performance Computing (HPC) "Jupyter", perteneciente al Grupo de Investigación Supercomputing-Algorithms en la Universidad de Almería (España). Está equipado con una GPU NVIDIA A30, un procesador AMD EPYC 7302 con 64 núcleos, y 128 GB de RAM DDR4. La versión de Python fue 3.10, la versión de TF fue 2.15.01, y la versión de PT fue 2.8.0.

B. Formulación de Problemas

Para validar la robustez y capacidad de generalización del framework propuesto, se consideraron dos problemas diferentes: reconocimiento de lengua de signos desde secuencias de pose humana y clasificación de imágenes hiperespectrales. Estas tareas se seleccionaron porque involucran modalidades de datos distintas y requisitos de procesamiento.

B.1 Reconocimiento de Lengua de Signos

El reconocimiento de la lengua de signos basado en pose se seleccionó como el primer escenario de validación. En esta tarea, la entrada consiste típicamente en secuencias temporales de keypoints de cuerpo y mano en lugar de imágenes raw. Cuando se usan modelos convolucionales, estas secuencias se convierten típicamente en imágenes esqueléticas. El modelo usado para esta tarea sigue una arquitectura Convolutional Neural Network (CNN)-Long Short-Term Memory (LSTM) con 99.355 parámetros.

B.2 Clasificación de Imágenes Hiperespectrales

La clasificación de imágenes hiperespectrales se seleccionó como el segundo escenario de validación. Utilizamos el dataset HyperLeaf2024 disponible públicamente [16], que comprende imágenes hiperespectrales del mundo real de hojas de plantas. Las muestras usadas tienen dimensiones de $48 \times 352 \times 204$.

El desafío principal en esta tarea surge del tamaño y dimensionalidad de los datos. El DA también juega un rol importante en este dominio. Consideramos tres categorías de técnicas de DA: (i) transformaciones geométricas, (ii) inyección de ruido gaussiano, y (iii) modulación de brillo. El modelo sigue una arquitectura basada en CNN con aproximadamente 0.6M parámetros.

C. Detalles de Experimentación

El objetivo de esta sección es validar el framework propuesto en los dos problemas definidos, analizando tanto rendimiento como escalabilidad bajo condiciones experimentales controladas.

C.1 Configuración y Métricas de Evaluación

En todos los experimentos, el dataset se divide en 120 lotes por época, y el número total de épocas se fija en diez. El tiempo de ejecución de la primera época se descarta. La herramienta *TensorFlow Profiler* se empleó a lo largo de todos los experimentos.

La métrica de rendimiento principal se define como el tiempo total de procesamiento (en segundos) requerido para completar el entrenamiento de los 120 lotes. Se realizó un análisis variando diferentes parámetros principales:

- **Batch Size**: Número de muestras procesadas por paso de entrenamiento.
- **N_WORKERS**: Número de procesos paralelos generados con la librería `multiprocessing`.
- **N_READERS**: Número de procesos concurrentes generados con funciones de pipeline de entrada de TF.
- **N_SLOTS**: Número de slots asignados en el ring buffer de memoria compartida.

C.2 Líneas Base para Comparación

Para evaluar rigurosamente el framework de ring buffer de memoria compartida propuesto, su rendimiento se comparó con pipelines de carga de datos nativamente optimizados en TF y PT:

- **Optimizado TF:** Emplea los mecanismos altamente optimizados `interleave()` y `prefetch()` de la API `tf.data`, con multithreading.
- **Optimizado PT:** Aprovecha el `DataLoader` nativo de PT con múltiples workers, usando `multiprocessing`.

D. Resultados

Esta sección reporta los resultados experimentales para los dos problemas considerados.

D.1 Evaluación de la Solución Propuesta

D.1.a Aumento de Datos en CPU. La Figura **??** ilustra la aceleración (eje Y) lograda por la mejor combinación de parámetros para cada tamaño de lote (eje X), relativa a la **línea base nativamente optimizada** de cada framework, para ambos problemas. Los gráficos distinguen entre dos perfiles de ejecución considerados: el flujo de trabajo completo, incluyendo el entrenamiento del modelo (*Entrenamiento*, líneas continuas) y la configuración de red mínima diseñada para aislar el rendimiento de carga de datos (*Solo carga*, líneas discontinuas).

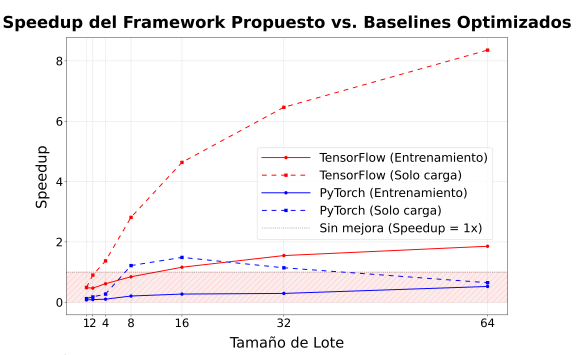


Fig. 3: Aceleración relativa a la implementación nativamente optimizada de cada framework para el Problema 1.

En el Problema 1 (Figura 3), se observa un contraste claro entre los dos frameworks al compararse con sus pipelines de datos nativamente optimizados. Para TF, nuestro framework propuesto ofrece mejoras sustanciales sobre una implementación optimizada de `tf.data`, con la ejecución exclusiva del pipeline de entrada escalando hasta una aceleración superior a ×8 con tamaño de lote 64, mientras que el perfil

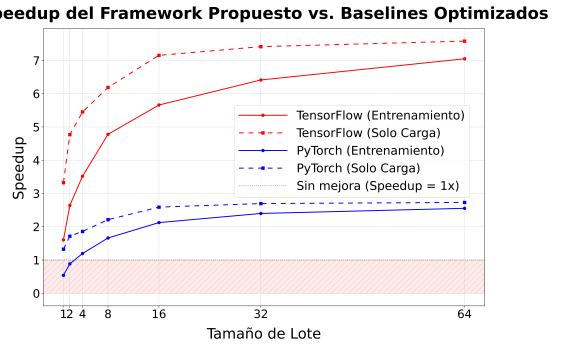


Fig. 4: Aceleración relativa a la implementación nativamente optimizada de cada framework para el Problema 2.

de entrenamiento completo se aproxima a una mejora de ×2. Esta ganancia significativa demuestra que, incluso cuando TF emplea sus mecanismos de paralelización nativos mediante multithreading, la arquitectura propuesta basada en `multiprocessing` y memoria compartida global logra superar las limitaciones impuestas por el GIL de Python y la gestión interna de memoria del framework.

Por el contrario, para PT, su `DataLoader` nativamente optimizado que aprovecha `multiprocessing` resulta altamente eficiente en este escenario dominado por operaciones de I/O sobre archivos pequeños. La implementación nativa de PT maneja de manera excepcionalmente efectiva la lectura masiva de archivos CSV microscópicos, distribuyendo la sobrecarga de apertura y cierre de archivos a través de múltiples subprocesos independientes. Consecuentemente, el enfoque propuesto generalmente no logra superar la línea base optimizada de PT, manteniéndose en su mayoría por debajo del umbral de aceleración unitaria. La ejecución de entrenamiento completo de PT permanece considerablemente por debajo de ×0,5, y aunque el perfil de solo entrada supera marginalmente la línea base en tamaños de lote de 8 y 16 (alcanzando aproximadamente ×1,5), se degrada nuevamente por debajo de 1 en tamaños de lote mayores. Esto sugiere que, para operaciones de I/O ligeras sobre archivos numerosos y pequeños, las optimizaciones nativas de carga de datos de PT están notablemente bien calibradas, haciendo que la sobrecarga de IPC introducida por nuestro framework resulte menos apropiada para este escenario particular.

En el Problema 2 (Figura 4), el escenario cambia completamente. Bajo condiciones computacionalmente pesadas y limitadas por memoria, el framework propuesto demuestra una superioridad clara sobre las líneas base nativamente optimizadas de ambos frameworks. Para TF, ambos perfiles de ejecución escalan excepcionalmente bien, logrando aceleraciones notables de aproximadamente ×7,5 para la configuración exclusiva del pipeline de entrada, y ×7 para el flujo de trabajo de entrenamiento completo en el tamaño de lote máximo. Más notablemente, a diferencia del Problema 1, nuestro framework también supera consistentemente la línea base nativamente optimizada de PT. Para tamaños de lote de 4 en adelante, ambos perfiles de PT superan la

sobrecarga inicial y logran ganancias de rendimiento sólidas entre ×2,5 y ×2,7 con tamaño de lote 64.

Crucialmente, esta brecha de rendimiento está fundamentalmente impulsada por la gestión de memoria global de nuestro framework. En implementaciones nativas basadas en `multiprocessing`, los mecanismos de carga de datos suelen asignar buffers de prefetch por worker. Cuando se manejan arrays masivos, los desarrolladores se ven obligados a limitar estrictamente el tamaño del buffer por proceso para prevenir errores de OOM, por ejemplo, permitiendo solo un elemento precargado por worker. Esta limitación estricta por proceso frecuentemente infrautiliza la memoria total disponible del sistema y falla en ocultar la latencia de I/O, dejando finalmente a la GPU sin datos. En cambio, nuestro ring buffer de memoria compartida desacopla la asignación de memoria del número de procesos worker. Al gestionar un pool global de slots de memoria, el framework puede maximizar de forma segura el uso de la capacidad de RAM disponible (por ejemplo, llenando exactamente el número máximo seguro de slots) sin vincularlo al conteo de workers. Esta ventaja arquitectónica mitiga efectivamente tanto los cuellos de botella de CPU como la severa sobrecarga de transferencia de datos que todavía sufren los dataloaders nativos al aumentar y pasar tensores multidimensionales enormes entre procesos.

D.1.b Aumento de Datos en GPU (solo Problema 2). Para evaluar la efectividad del framework cuando el DA se descarga a la GPU, se realizó un conjunto adicional de experimentos para el Problema 2. Como se detalló en la Sección IV-C.2, las operaciones pesadas de DA se eliminaron completamente de los pipelines de CPU de todos los enfoques evaluados, delegándolas a la GPU mediante métodos nativos del framework.

La Figura 5 muestra la aceleración (eje Y) lograda por la mejor combinación de parámetros para cada tamaño de lote (eje X), relativa a la línea base nativamente **optimizada** de cada framework en la configuración de DA basada en GPU. Notablemente, incluso con el DA delegado a la GPU, el framework propuesto supera sistemáticamente a los dataloaders de `multiprocessing` y `multithreading` tanto de PT como de TF en todos los tamaños de lote, evitando completamente la zona de degradación de rendimiento.

Respecto a PT, la aceleración sigue un crecimiento logarítmico claro, superando rápidamente la sobrecarga inicial del framework en tamaños de lote menores y estabilizándose entre ×3 y ×3,4 para tamaños de lote de 16 y superiores. TF presenta ganancias de rendimiento aún mayores, estableciéndose entre ×3,5 y ×4,5 en los tamaños de lote más grandes. Estos resultados confirman que, al manejar arrays multidimensionales masivos, el verdadero cuello de botella en los pipelines nativamente optimizados no es solo la computación de CPU. En cambio, la fuerte fragmentación de memoria y la elevada sobrecarga asociada a la transferencia de tensores extremadamente pesados entre procesos afectan de forma notable al rendimiento de los dataloaders nativos basados en `multiprocessing` y `multithreading`. Al utilizar el ring buffer de memoria compartida, nuestro framework evita esta limitación de ancho de banda de memoria IPC, asegurando una entrega de datos de alto rendimiento incluso cuando la CPU actúa puramente como lector y formateador de datos.

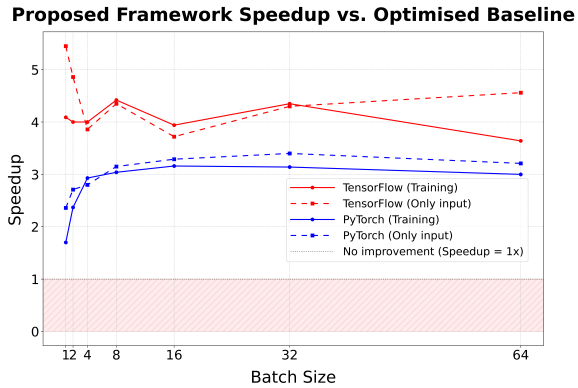


Fig. 5: Aceleración dependiendo del tamaño de lote para Problema 2, con DA ejecutado nativamente en la GPU, relativa a la implementación optimizada de cada framework.

D.2 Impacto de los Diferentes Parámetros

Esta sección analiza el impacto de cada parámetro definido en la Sección IV-C sobre el framework propuesto en la versión con aumentos en CPU. Dado que el Problema 1 sigue las mismas tendencias cualitativas que el Problema 2, aunque con tiempos absolutos menores debido a la menor carga computacional, se presentan únicamente los resultados del Problema 2, sin que ello afecte a las conclusiones generales.

D.2.a Número de readers y workers. La Figura 6 muestra cómo *N_READERS* y *N_WORKERS* afectan al tiempo de procesamiento del sistema. El eje Y representa el tiempo medio por época, mientras que el eje X está ordenado por el número de procesos readers. Para cada número de readers, diferentes números de workers se codifican mediante color. Esta disposición permite una comparación directa de cómo la variación de estos parámetros influye en el tiempo de procesamiento.

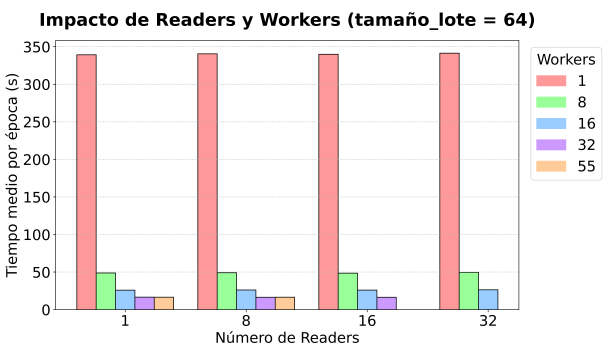


Fig. 6: Tiempo de procesamiento dependiendo de la configuración *N_READERS* y *N_WORKERS* para el Problema 2.

Los resultados muestran que el tiempo medio de época disminuye significativamente una vez que se emplea más de un worker, mientras que el número de

readers tiene un impacto marginal en el rendimiento. Con un único worker, el sistema exhibe un claro cuello de botella en la etapa de carga y preprocesamiento de datos, evidenciado por tiempos de entrenamiento que superan los 300 segundos por época. Incrementar el número de workers a 8 o 16 reduce el tiempo a aproximadamente 50 segundos por época, indicando que la capacidad de carga y preparación de datos se equilibra aproximadamente con la velocidad de entrenamiento del modelo. El hecho de que readers adicionales no mejoren sustancialmente el rendimiento indica que la etapa de precarga y alimentación de la GPU no constituye el factor limitante en este caso, y que la mayor parte del tiempo de espera surge de las operaciones de preprocesamiento en CPU e I/O.

La dependencia del número de workers es especialmente relevante en este escenario debido al elevado coste computacional asociado al procesamiento de imágenes hiperespectrales: incrementar hasta 16 o 32 workers reduce significativamente el tiempo por época, evidenciando una mejor paralelización de la etapa de preparación de datos. Como ocurre en el Problema 1, aumentar el número de readers no produce mejoras sustanciales, confirmando que la transferencia de datos a la GPU no constituye el cuello de botella. En conjunto, los resultados demuestran que la eficiencia del pipeline de entrada depende principalmente del grado de paralelización alcanzado por los workers, mientras que los readers desempeñan un papel secundario una vez que la tasa de datos permanece por debajo de la velocidad de computación del modelo.

D.2.b Número de slots. La Figura 7 muestra cómo *N_SLOTS* y *N_WORKERS* afectan al tiempo de procesamiento del sistema. El eje Y representa el tiempo medio por época, mientras que el eje X representa el número de workers agrupados por el valor de *N_SLOTS*.

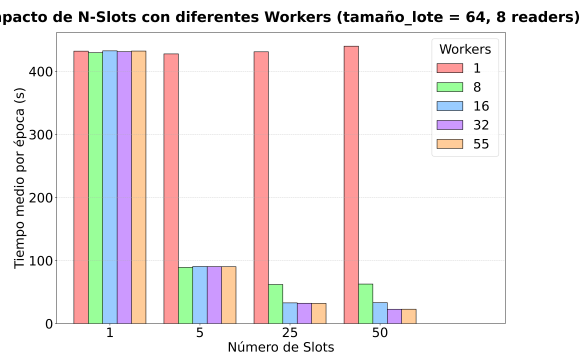


Fig. 7: Tiempo de procesamiento dependiendo de la configuración N_SLOTS para el Problema 2.

En el análisis realizado para el Problema 2, se puede apreciar que *N_SLOTS* es crítico para el rendimiento del sistema. Tener solo un slot es la configuración que presenta los tiempos de procesamiento más altos. En este caso, se produce un cuello de botella en el pipeline de entrada, ya que la memoria compartida solo contiene un elemento a la vez. Aumentarlo a 5 hace el pipeline de entrada más eficiente, aunque las configuraciones que emplean 8 o más workers también se ven limitadas por el número de slots disponibles. Se logra mayor aceleración cuando se usan 25 y 50 slots, donde hay suficientes elementos en memoria compartida a los que readers y workers pueden acceder simulatáneamente, mejorando así la eficiencia del pipeline de entrada.

Teniendo en cuenta los resultados, se puede apreciar que incrementar el número de workers cuando no existe un cuello de botella causado por el parámetro *N_SLOTS* hace el pipeline más eficiente. Sin embargo, es importante señalar que *N_SLOTS* debe establecerse cuidadosamente, ya que incrementarlo sin verificar el tamaño de RAM del sistema puede provocar un desbordamiento de memoria.

E. Limitaciones

A pesar de las mejoras de rendimiento logradas, el framework propuesto presenta ciertas limitaciones.

Primero, la arquitectura del pipeline subyacente depende en gran medida de mecanismos de memoria compartida gobernados por el estándar POSIX. Por tanto, el framework está diseñado nativamente para entornos Linux. Sin embargo, es completamente compatible con macOS. Aunque no se ejecuta nativamente en Windows, puede ejecutarse sin problemas a través de Windows Subsystem for Linux (WSL).

Segundo, respecto a escalabilidad de hardware, la versión actual del framework carece de arquitectura multi-nodo, haciendo inviable distribuir el entrenamiento a través de múltiples máquinas. En su implementación multi-GPU actual, el framework utiliza un solo ring buffer de memoria compartida global por máquina que agrupa un lote global, que posteriormente se fragmenta a través de todas las GPUs locales disponibles.

Finalmente, el sistema propuesto está altamente optimizado para mitigar cuellos de botella que surgen de operaciones de preprocesamiento complejas y la carga de muestras de datos grandes y pesadas. Para escenarios como el Problema 1, que implica operaciones I/O masivas de archivos muy pequeños, el overhead del ring buffer se vuelve contraproducente comparado con PT.

V. Conclusiones

Este trabajo presentó un framework modular, paralelo y compatible con multi-GPU para la ingesta de datos de alto rendimiento y DA online en DL. Centrado en un ring buffer de memoria compartida global, el framework se integra sin problemas con la APIs de carga de datos nativa de TF. Al desacoplar la producción de datos del consumo, la arquitectura permite que múltiples workers de CPU carguen y preprocesen lotes en paralelo, evitando de forma efectiva el GIL de Python y los cuellos de botella de memoria IPC.

Mediante dos problemas representativos (reconocimiento de lengua de signos basado en pose y clasificación de imágenes hiperespectrales), el sistema propuesto demostró una clara superioridad sobre los pipelines nativamente optimizados de TF y PT. Com-

parado con estas líneas base optimizadas, el framework logró aceleraciones de hasta ×8 en carga de datos aislada y hasta ×7 en entrenamiento completo en escenarios intensivos en memoria, con ganancias consistentes también en la configuración de DA basada en GPU.

Los estudios de ablación aclararon el papel de los parámetros clave. El número de workers paralelos (*N_WORKERS*) es el principal factor determinante para el rendimiento, mientras que el número de readers del lado del framework (*N_READERS*) tiene un efecto marginal. La capacidad de memoria compartida global (*N_SLOTS*) resultó crítica: mientras que un número reducido de slots introduce cuellos de botella en el pipeline, buffers globales óptimamente dimensionados maximizan de forma segura la RAM disponible.

En general, el enfoque propuesto es una solución escalable y plug-and-play para formatos de datos no estándar y DA online costoso, maximizando la utilización de GPU.

El trabajo futuro se centrará en desarrollar un mecanismo de autoajuste que seleccione conjuntamente los diferentes parámetros del ring buffer bajo restricciones de CPU/RAM del sistema y memoria de GPU. Además, se explorará un enfoque híbrido de preprocesamiento CPU/GPU. Finalmente, se investigará extender el framework para dar soporte al entrenamiento multi-nodo distribuido.

Agradecimientos

Este trabajo ha sido financiado por el proyecto R+D+i PID2024-155521OB-I00 de MCIN/AEI/10.13039/501100011033/ y fondos ERDF. Además, este trabajo recibió financiación del proyecto CPUENTE2024/05 de la Universidad de Almería.

Referencias

[1] N. Sharma, R. Sharma, and N. Jindal, "Machine Learning and Deep Learning Applications-A Vision," *Global Transitions Proceedings*, vol. 2, no. 1, pp. 24–28, 2021.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al., "TensorFlow: a system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.

[3] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al., "PyTorch: an imperative style, high-performance deep learning library," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019, pp. 8026–8037.

[4] Daniel Nichols, Siddharth Singh, Shu-Huai Lin, and Abhinav Bhatele, "A Survey and Empirical Evaluation of Parallel Deep Learning Frameworks," *arXiv preprint arXiv:2111.04949*, 2021.

[5] Joao Carreira and Andrew Zisserman, "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4724–4733.

[6] Connor Shorten and Taghi M Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.

[7] Luis Pérez and Jason Wang, "The Effectiveness of Data Augmentation in Image Classification using Deep Learning," *arXiv preprint arXiv1712.04621*, 2017.

[8] Alexander B. Jung, Kentaro Wada, Jonathon Crall, Satoshi Tanaka, Jacob Graving, Christoph Reinders, Sarthak Yadav, Joy Banerjee, György Vecsei, Adam Kraft, et al., "imgaug," `https://github.com/aleju/imgaug`, 2020, GitHub repository.

[9] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Alexander Druzhinin, and Alexandr A. Kalinin, "Albumentations: Fast and Flexible Image Augmentations," *Information*, vol. 11, no. 2, pp. 125, 2020.

[10] Zhe Gao, Jiali Li, Hao Wang, et al., "Deep Learning at Scale: Distributed Data Pipelines for Large Image Datasets," *Journal of Parallel and Distributed Computing*, vol. 144, pp. 55–65, 2020.

[11] Sang-Ki Ko, Chang Jo Kim, Hyedong Jung, and Choongsang Cho, "Neural sign language translation based on human keypoint estimation," *Applied Sciences*, vol. 9, no. 13, 2019.

[12] M. E. Paoletti, J. M. Haut, J. Plaza, and A. Plaza, "Deep Learning Classifiers for Hyperspectral Imaging: A Review," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 158, pp. 279–317, 2019.

[13] Derek G. Murray, Jiri Simsa, Ana Klimovic, and Ihor Indyk, "tf.data: A machine learning data processing framework," 2021.

[14] Edgar Riba, Dmytro Mishkin, Daniel Ponsa, Ethan Rublee, and Gary Bradski, "Kornia: an open source differentiable computer vision library for pytorch," in *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, 2020, pp. 3674–3683.

[15] NVIDIA, "Dali - nvidia data loading library," `https://docs.nvidia.com/deeplearning/dali/user-guide/docs/`.

[16] William Michael Laprade, Pawel Pieta, Svetlana Kutuzova, Jesper Cairo Westergaard, Mads Nielsen, Svend Christensen, and Anders Bjorholm Dahl, "Hyperleaf2024 – a hyperspectral imaging dataset for classification and regression of wheat leaves," in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2024, pp. 1234–1243.

Series temporales de temperaturas de GPU durante operaciones GEMM

Francisco Yáñez Rodríguez[1] y Carlos Fernández Sánchez[1]

Resumen— **Este trabajo presenta una herramienta de *benchmarking* desarrollada en CESGA que mide el rendimiento de operaciones GEMM a lo largo del tiempo mientras registra la temperatura de la GPU. La metodología permite visualizar los regímenes transitorios y estacionarios presentes en el rendimiento, así como su correlación con la temperatura en diferentes arquitecturas de GPU y precisiones numéricas. Además se explora brevemente como ajustar estos datos a modelos lineales sencillos de primer y segundo orden.**

Palabras clave— **Benchmarking de GPU, GEMM, dinámica térmica, series temporales, análisis de rendimiento.**

I. Introducción

LAS unidades de procesamiento gráfico (GPUs) se han consolidado como uno de los motores fundamentales del cómputo moderno gracias a su arquitectura optimizada para la ejecución masiva de multiplicaciones de matrices (GEMM). Esta capacidad es el pilar sobre el que se asientan dominios críticos como el entrenamiento e inferencia de Modelos de Lenguaje Masivos (LLMs), el análisis de datos (Big Data) y las simulaciones científicas de diversas ramas del saber. Sin embargo, la creciente densidad de transistores y las frecuencias operativas extremas desplazan el cuello de botella de estos sistemas de la capacidad de cómputo bruta hacia la eficiencia térmica del sistema.

Para poder comprender en su totalidad el comportamiento de una GPU en lo referente a su eficiencia en el tiempo de ejecución de las operaciones, la monitorización del uso de las GPUs ya no se limita a métricas que definan las características del sistema en estático. En los orígenes de las GPUs de propósito general, los estudios existentes se centraban especialmente en la caracterización de estos sistemas en distintos regímenes de trabajo, midiendo muchas variables y las correlaciones de las mismas, como la potencia consumida, su frecuencia de reloj, y adicionalmente su temperatura [1], [2], [3]. En los últimos años, si bien estas relaciones siguen midiéndose y estudiándose [4], es cada vez más frecuente encontrar estudios y análisis en los que los datos del consumo y la temperatura se obtienen estudiando ejecuciones prolongadas en el tiempo y prestando atención a las propiedades de esas series temporales [5], [6]. Esta información, además, puede ser de gran relevancia para la modelización de los comportamientos más representativos no solo de las GPUs, sino también de muchos otros sistemas electrónicos [7]. Por ello, es especialmente relevante en monitorización HPC la telemetría de alta resolución, capaz de observar la dinámica del hardware bajo condiciones de estrés sostenido.

Este interés por la modelización no es únicamente descriptivo. Cuando una serie temporal de temperatura se registra con suficiente resolución y durante un intervalo temporal adecuado, resulta natural preguntarse si su evolución puede aproximarse mediante modelos dinámicos de baja o alta complejidad. En particular, los sistemas de primer orden permiten representar de manera compacta procesos de calentamiento con una única constante de tiempo dominante, mientras que los sistemas de segundo orden ofrecen un marco ligeramente más rico para capturar respuestas con varias escalas temporales o con transitorios más complejos [8]. En el contexto de las GPUs, este tipo de aproximaciones puede ayudar a interpretar cuantitativamente la inercia térmica del dispositivo, comparar arquitecturas distintas y relacionar la respuesta observada con parámetros físicos o de diseño del sistema.

A pesar de la existencia de herramientas de gestión a nivel de centro de datos, como Grafana para la visualización de series temporales o Prometheus para la recolección de dichos datos, a menudo falta una visibilidad directa y granular que correlacione la carga de trabajo específica con la respuesta física del dispositivo. Comprender los límites de carga de estos sistemas y cómo sus propiedades térmicas influyen en el rendimiento real puede aportar información muy valiosa sobre sus regímenes operativos y casos de uso óptimos. Con este objetivo, presentamos una herramienta de *benchmarking* desarrollada en el Centro de Supercomputación de Galicia (CESGA). Esta herramienta, caracterizada por su sencillez y eficacia, permite medir simultáneamente dos series temporales críticas: el tiempo de ejecución de las operaciones GEMM y la temperatura de la GPU que las realiza. Además, permite construir visualizaciones gráficas que facilitan la inspección rápida del comportamiento observado y sientan una base adecuada para introducir análisis posteriores de ajuste y modelización matemática de dichas curvas térmicas.

Este manuscrito comenzará describiendo los métodos empleados, en los que detallaremos el esquema de trabajo que hemos seguido para desarrollar esta aplicación, como emplearla a nivel de usuario final y una breve explicación de los sistemas lineales de primer y segundo orden. Después, en los resultados, presentaremos varios ejemplos del empleo de esta herramienta con distintas opciones y máquinas, para comparar varias GPUs y verificar el uso de la herramienta. Asimismo, incorporaremos un primer análisis de

[1]CESGA (Galicia Supercomputing Center), Avda. de Vigo s/n, Campus Sur, 15705, Santiago de Compostela, España.

ajuste de las curvas de temperatura mediante sistemas dinámicos de primer y segundo orden, con el fin de explorar hasta qué punto estos modelos sencillos permiten describir la evolución térmica observada. Finalmente, cerramos con unas breves conclusiones y trabajo futuro.

II. Métodos

Para automatizar el lanzamiento de estas pruebas, o *benchmarks*, hemos diseñado un conjunto de herramientas en Bash y Python. El código está disponible en GitHub, en el siguiente repositorio: `yanezrodfrancisco/gpu-thermal-perf-evolution`.

La Figura 1 resume el flujo de trabajo completo de la aplicación. Aunque el usuario interactúa únicamente con dos scripts, bordeados de azul en la Figura 1; el procedimiento interno requiere coordinar varias piezas software para medir, almacenar y representar las series temporales de rendimiento y temperatura.

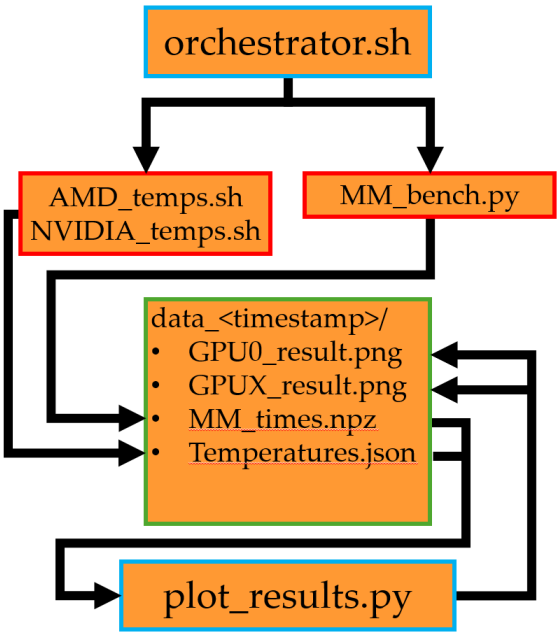


Fig. 1: Diagrama de flujo de la suite de *benchmarking*.

A. Scripts y componentes internos

El script principal de la herramienta es `orchestrator.sh`, que coordina y automatiza las dos tareas fundamentales de cada ejecución. Por un lado, lanza el script específico de la arquitectura subyacente, bien `AMD_temps.sh` o `NVIDIA_temps.sh` (dependiendo de la GPU subyacente), con el que se mide y registra la temperatura, poniendo en un formato adecuado las lecturas de la temperatura de `rocm-smi` o `nvidia-smi`. Por otro lado, ejecuta `MM_bench.py`, que es el programa encargado de realizar de forma repetida las operaciones GEMM sobre la GPU y medir el tiempo de cada una de ellas. De este modo, `orchestrator.sh` sincroniza la adquisición de temperatura con la carga computacional y centraliza toda la lógica de ejecución.

Durante cada prueba, `orchestrator.sh` crea automáticamente un directorio de salida con nombre `data_<timestamp>`, en el que se almacenan las medidas realizadas. A partir de ese directorio, el usuario puede analizar los ficheros generados directamente o emplear de manera opcional el script `plot_results.py`, cuyo objetivo es procesar las series temporales almacenadas y generar las gráficas resumen del experimento, como las que presentamos nosotros más adelante en la sección de resultados.

Esta separación entre adquisición, almacenamiento y visualización persigue dos objetivos. Por un lado, simplifica la reutilización de la herramienta en nuevos entornos hardware, puesto que los cambios necesarios suelen quedar limitados a la capa de lectura de temperatura. Por otro lado, permite repetir el análisis sobre un conjunto de datos ya adquirido sin necesidad de volver a ejecutar el *benchmark*, algo especialmente útil cuando se desea comparar varias configuraciones o ajustar la presentación final de las curvas.

B. Flujo de uso para el usuario final

Desde el punto de vista del usuario final, el flujo de trabajo es muy sencillo. En primer lugar, se ejecuta `orchestrator.sh`, que lanza la prueba de multiplicaciones de matrices y sincroniza su ejecución con la adquisición de temperatura.

Como dijimos antes, durante la ejecución, `orchestrator.sh` crea un directorio de salida con nombre `data_<timestamp>` y almacena en él los datos brutos del experimento. Las temperaturas se guardan en un fichero `.json`, lo que permite registrar de forma natural medidas procedentes de sistemas con varias GPUs. Por el contrario, los tiempos de ejecución se recogen para una única GPU y se almacenan en un fichero `.npz`, pensado para facilitar su carga y posterior análisis desde Python.

A partir de ese directorio, el usuario puede seguir dos caminos. El primero, de carácter más inmediato, consiste en ejecutar opcionalmente `plot_results.py` para generar las gráficas resumen del experimento, como las que se presentan en este trabajo. El segundo consiste en reutilizar los datos almacenados para realizar análisis adicionales, como el ajuste de las curvas térmicas a modelos dinámicos de primer y segundo orden que se estudiará más adelante en la sección de resultados.

Recalcamos que la operación evaluada es la multiplicación de dos matrices, específicamente dos matrices cuadradas $N \times N$. Con `orchestrator.sh`, el usuario puede configurar el tamaño de la matriz, la precisión numérica, el intervalo de muestreo de temperatura (en segundos) y el número total de GEMMs que se desea realizar. En consecuencia, la misma herramienta puede utilizarse tanto para estudios exploratorios de corta duración como para campañas largas orientadas a caracterizar el comportamiento térmico y el rendimiento sostenido bajo carga conti-

nuada.

Como apoyo al análisis que se introducirá en la sección de resultados, presentamos brevemente una descripción simplificada de los sistemas lineales de primer y segundo orden. Un sistema lineal de primer orden puede escribirse como

$$\tau \frac{dT(t)}{dt} + T(t) = u(t), \tag{1}$$

donde $T(t)$ representa la temperatura, $u(t)$ la entrada efectiva y τ el tiempo característico del sistema. Para una entrada $u(t)$ de valor constante, su solución analítica adopta la conocida forma exponencial

$$T(t) = T_\infty + (T_0 - T_\infty)\, e^{-t/\tau}. \tag{2}$$

siendo T_0 la temperatura en el instante inicia y T_∞ la temperatura en el equilibrio tras un tiempo transitorio. Por su parte, un sistema de segundo orden puede representarse mediante

$$\frac{d^2T(t)}{dt^2} + \mu \frac{dT(t)}{dt} + \omega^2 T(t) = u(t), \tag{3}$$

donde μ es el amortiguamiento de la temperatura en el tiempo y ω la frecuencia natural de las oscilaciones de la temperatura. A efectos prácticos, en el caso de $u(t)$ constante, lo que este sistema añade es una segunda exponencial que nos da más control sobre las propiedades de este sistema, pudiendo escribir una función solución general de la siguiente forma:

$$T(t) = T_\infty + c_1 e^{\lambda_1 t} + c_2 e^{\lambda_2 t}, \tag{4}$$

siendo c_1 y c_2 la importancia de las exponenciales con exponentes λ_1 y λ_2 (pudiendo ser estos exponentes números complejos). T_∞ mantiene el significado físico del anterior modelo.

Estas formas analíticas proporcionan un marco sencillo para estudiar si las curvas de temperatura observadas pueden interpretarse como respuestas dinámicas de baja dimensión. No es complejo hallar los valores concretos de estos parámetros con un ajuste de los datos. Nosotros ajustaremos las curvas con un script sencillo de python en el que usaremos `scipy.optimize.curvefit`. Con eso, se podrían recuperar los parámetros de las ecuaciones planteadas mediante herramientas como la transformada de Laplace.

III. Resultados

En la Figura 2 presentamos dos experimentos representativos en dos GPUs distintas: una NVIDIA A100 y una AMD Instinct MI210. En ella, se muestran las series temporales de la temperatura y el tiempo de ejecución para GEMMs en precisión `float32` con un tamaño de matriz de $N = 20000$.

La observación más destacada es la presencia de un estado transitorio de rendimiento fuertemente correlacionado (en forma y duración) con la curva de temperatura de la GPU. Además, la comparación confirma una diferencia conocida entre dispositivos: en estos experimentos, la A100 presenta menor rendimiento en `float32` que la MI210.

La Tabla I resume el rendimiento promedio, calculado con las últimas 100 multiplicaciones de cada serie temporal. De esta forma, garantizamos usar puntos tras este transitorio observado en la Figura 2. Adicionalmente, incluimos los resultados de estas pruebas en una GPU NVIDIA GB10 de una DGX SPARK, un sistema enfocado para ejecutar modelos de Inteligencia Artificial.

Precisión	A100	MI210	GB10
bfloat16	0.0765	0.1372	2.2022
float32	0.9714	0.5260	4.6821
float64	1.1124	0.7825	34.9903

Tabla I: Tiempo medio de ejecución (en segundos) para las últimas 100 operaciones GEMM.

Analizando los resultados de esta Tabla I podemos sacar información muy interesante: en primer lugar, las tarjetas gráficas modernas si no van enfocadas al cálculo y al uso HPC, como el caso de la GB10, no pueden sustituir a las tarjetas centradas en dichas operaciones como la A100 o la MI210. Adicionalmente, entre estos dos últimos dispositivos encontramos una diferencia fundamental según la precisión empleada: la NVIDIA A100 es mucho más eficiente en las operaciones de menor precisión aritmética, mientras que la AMD Instinct MI210 es mejor en precisiones mayores. Esta información es compatible con

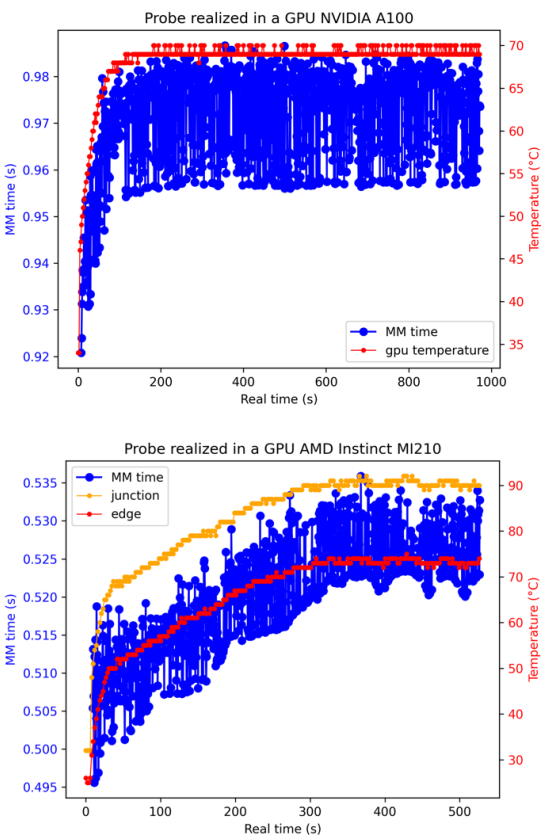


Fig. 2: Series temporales de temperatura (eje rojo) y tiempo de ejecución (eje azul) para dos experimentos representativos en GPU.

la ofrecida por las empresas distribuidoras [9], [10].

Como ejemplo de las posibles aplicaciones de esta herramienta, mostrando un pequeño caso de uso de cómo estos datos extraídos pueden ser útiles para modelizar y caracterizar las GPUs de formas más analíticas. Como adelantamos en los métodos, vamos a comprobar cómo de bien se ajustan estas curvas mostradas en la Figura 2 a modelos lineales de primer y segundo orden. Concretamente, lo haremos solo para las curvas de la temperatura, ya que la de las ejecuciones de operaciones en la GPU presentan fluctuaciones bruscas y requeriría un preprocesado antes de hacer estos análisis. Como vemos en la Figura 3, la GPU AMD Instinct MI210 presenta un transitorio algo más complejo que la GPU NVIDIA A100. Una forma de verificar esto es comprobando cómo de buenos son estos ajustes empleando el error cuadrático medio (RMSE por sus siglas en inglés). Tal y como puede verse en las leyendas de la Figura 3, para el caso de la GPU A100, los ajustes son bastante buenos, con RMSE menores que 1 grado Celsius, y con una ligera mejora al considerar un sistema de segundo orden en vez de primer orden. No es el caso para la MI210, en el que la mejoría es mucho más clara al considerar el modelo de segundo orden. Además, estos RMSE son más altos que en el caso de la A100, mostrándonos que esta gráfica se ajusta algo peor a este tipo de comportamientos. Este resultado de un peor ajuste al modelo en las gráficas de AMD en comparación con las de NVIDIA se mantiene para otros tamaños de matriz y precisiones numéricas empleadas.

Terminamos comentando los tiempos característicos de los ajustes realizados, ya que de esta forma podemos cuantificar otra propiedad que se aprecia visualmente en ambas curvas: la Instinct MI210 se calienta más despacio que la A100. La Tabla II resume los tiempos característicos más grandes (es decir, los que dominan la dinámica a medio plazo en este escenario concreto) obtenidos en los ajustes de primer y segundo orden para ambas GPUs, junto con las métricas de calidad del ajuste. En particular, se observa que la MI210 presenta un tiempo característico dominante claramente superior a la de la A100, lo que es consistente con un calentamiento más lento en el régimen transitorio. Añadimos en esta tabla el coeficiente de Pearson R^2 para ver cuán fiables son dichos ajustes de los que deducimos los valores.

GPU	Modelo	τ dominante	R^2
A100	1er orden	28.90	0.9763
A100	2º orden	34.48	0.9879
MI210	1er orden	113.80	0.9591
MI210	2º orden	154.05	0.9836

Tabla II: Tiempos característicos (en segundos) y coeficiente de Pearson para los modelos de primer y segundo orden empleados sobre las curvas de temperatura.

IV. Conclusiones

Esta herramienta de *benchmarking* desarrollada en CESGA ofrece una forma práctica de evaluar la eficiencia de GPUs monitorizando la evolución del rendimiento a lo largo del tiempo y correlacionándola con la temperatura del dispositivo. Frente a aproximaciones centradas exclusivamente en medidas estáticas o valores agregados, el enfoque adoptado en este trabajo pone el acento en la adquisición simultánea de variables temporales, permitiendo observar tanto el régimen transitorio como el estacionario de la respuesta del sistema bajo carga sostenida.

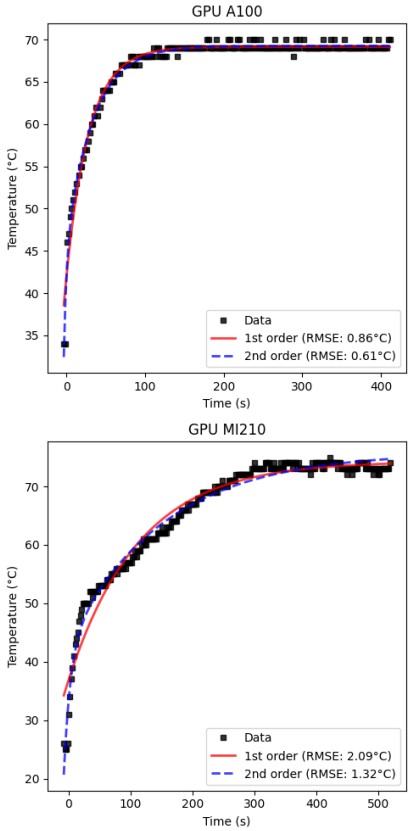


Fig. 3: Ajuste de las curvas de temperatura de la mediante modelos lineales de primer y segundo orden. En la leyenda de la figura adjuntamos el RMSE

En este sentido, la principal aportación de la herramienta no reside únicamente en automatizar la ejecución de pruebas GEMM, sino en proporcionar un marco reproducible para medir, almacenar y representar conjuntamente el tiempo de cómputo y la evolución térmica. Esta información resulta especialmente valiosa cuando se pretende comparar arquitecturas distintas o estudiar cómo la precisión aritmética y el tipo de dispositivo condicionan el comportamiento real del hardware más allá de las especificaciones nominales.

Además, los resultados obtenidos muestran que estas series temporales pueden servir también como punto de partida para análisis más interpretables desde el punto de vista matemático. En particular, el ajuste de las curvas de temperatura mediante modelos lineales de primer y segundo orden permite resumir parte de la dinámica observada en un número reducido de parámetros, facilitando la comparación entre dispositivos y proporcionando una primera aproximación a la inercia térmica de cada GPU. Aunque se trata de modelos sencillos, su capacidad

para capturar tendencias globales de calentamiento sugiere que este tipo de herramientas pueden ser útiles como puente entre la experimentación empírica y la modelización matemática del sistema.

En resumen, este software es útil tanto como mecanismo de prueba de estrés para GPUs compatibles como instrumento de análisis para estudiar la relación entre rendimiento hardware y temperatura. Esta combinación entre medidas de rendimiento y temperatura, automatización de pruebas y ajuste de modelos propone una vía sencilla y accesible para caracterizar dispositivos aceleradores en contextos HPC.

Como trabajo futuro, se plantea extender estas pruebas a escenarios con más de una GPU, de forma que puedan analizarse los efectos térmicos y de rendimiento en ejecuciones distribuidas. También resultará de interés contrastar los ajustes actuales con modelos de mayor complejidad o con formulaciones más físicamente fundamentadas, para corroborar si los patrones observados se mantienen y hasta qué punto las aproximaciones de primer y segundo orden son suficientes para describir el comportamiento térmico real de estos sistemas.

Referencias

[1] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi, "Gpuwattch: enabling energy optimizations in gpgpus," *Association for Computing Machinery*, vol. 41, no. 3, 2013.

[2] Yuki Abe, Hiroshi Sasaki, Shinpei Kato, Koji Inoue, Masato Edahiro, and Martin Peres, "Power and performance characterization and modeling of gpu-accelerated systems," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, 2014, pp. 113–122.

[3] Cong Liu, Jian Li, Wei Huang, Juan Rubio, Evan Speight, and Felix Xiaozhu Lin, "Power-efficient time-sensitive mapping in heterogeneous systems," in *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012, pp. 23–32.

[4] H. Zhang, S. Tu, and M. Li, "An empirical study of power capping and thermal dynamics on modern gpu architectures," *IEEE Micro*, vol. 42, no. 3, pp. 58–67, May–June 2022.

[5] Imran Latif, Muhammad Ali Shafique, et al., "Cooling matters: Benchmarking large language models and vision-language models on liquid-cooled versus air-cooled h100 gpu systems," *CoRR*, vol. abs/2507.16781, 2025.

[6] Seokjin Go, Joongun Park, Spandan More, et al., "Characterizing the efficiency of distributed training: A power, performance, and thermal perspective," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, 2025.

[7] David Atienza, Kai Zhu, Darong Huang, and Luis Costero, "A 20-year retrospective on power and thermal modeling and management," vol. abs/2508.05495v1, 2025.

[8] F. N. Masana, "A new approach to the dynamic thermal modelling of semiconductor packages," *Microelectronics Reliability*, vol. 41, no. 6, pp. 901–912, 2001.

[9] Advanced Micro Devices, Inc., "Amd instinct mi210 accelerator," Product Brief, online, 2022, Available: `https://www.amd.com/content/dam/amd/en/documents/instinct-business-docs/product-briefs/instinct-mi210-brochure.pdf`.

[10] NVIDIA Corporation, "Nvidia a100 tensor core gpu," Datasheet, online, 2021, Available: `https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet.pdf`.

Aumento adaptativo paralelo de imágenes sintéticas para la estimación de pose humana en el espectro térmico.

A. De Toro-Castro, M. Lupión, J.F. Sanjuan, V. González-Ruiz y P.M. Ortigosa [1]

Resumen— **En el ámbito de las imágenes térmicas de baja resolución, la estimación de la pose humana (*HPE*) continúa siendo un desafío significativo, dado que modelos avanzados y ampliamente reconocidos, como OpenPose, presentan un deterioro notable en su rendimiento. En este trabajo se propone un pipeline de aumento de datos con entrada adaptativa que aprovecha la generación iterativa de imágenes térmicas sintéticas, caracterizadas por la detección de poses humanas completas, con el objetivo de optimizar el entrenamiento de modelos Generative Adversarial Networks (*GAN*).**

Una contribución clave de este estudio es el diseño de un pipeline híbrido de ejecución, optimizado para entornos computacionales heterogéneos de alto rendimiento. Este modelo de procesamiento opera de manera paralela y asincrónica: la *GPU* se encarga del entrenamiento de la red *GAN* y de la generación sintética de imágenes mediante el modelo preentrenado, mientras que la detección de poses en dichas imágenes sintéticas se realiza utilizando los núcleos disponibles restantes de la *CPU*.

En cada iteración, se generan imágenes sintéticas a partir del modelo preentrenado en la iteración anterior, las cuales se analizan con OpenPose para identificar poses humanas completas. Paralelamente, el modelo se entrena utilizando retroalimentación basada en aquellas imágenes sintéticas generadas en iteraciones previas en las que se detectó al menos una pose completa. La cantidad de imágenes generadas en cada iteración es variable y depende del tiempo de cómputo requerido para el entrenamiento del modelo.

Este diseño paralelo e iterativo permite la integración dinámica de muestras sintéticas en el conjunto de entrenamiento sin necesidad de anotaciones manuales, traduciéndose en una mejora sustancial de la robustez y el rendimiento en tareas de estimación de pose humana en imágenes térmicas.

Palabras clave— **Estimación de la Pose Humana (*HPE*), Imágenes Térmicas de Baja Resolución, Aumento de Datos, Generación de Datos Sintéticos, Pipeline Asíncrono.**

I. Introducción

La estimación de la pose humana (*HPE*) es un problema central en la visión por computador. Su objetivo es detectar y localizar articulaciones y segmentos corporales en imágenes o secuencias de vídeo para interpretar automáticamente el movimiento humano. Esta capacidad tiene un impacto directo en múltiples aplicaciones: en salud y rehabilitación, permite el seguimiento remoto y objetivo de pacientes; en el reconocimiento de actividades humanas, facilita el análisis de acciones y comportamientos; en seguridad y vigilancia, posibilita la detección de caídas y anomalías; y en sistemas inteligentes, habilita la interacción natural con robots, interfaces gestuales y dispositivos IoT.

Los principales avances recientes han sido impulsados por imágenes RGB de alta resolución, que ofrecen gran detalle espacial para la localización precisa de puntos clave. Sin embargo, este enfoque plantea problemas de privacidad al capturar información identificable como rasgos faciales, lo cual resulta crítico en hospitales, residencias, hogares inteligentes o sistemas de vigilancia masiva. Estas preocupaciones han motivado el estudio de modalidades alternativas que preserven la privacidad sin sacrificar el rendimiento.

La imagen térmica se presenta como una alternativa prometedora. Representa a los individuos mediante su firma de calor, lo que garantiza la preservación de la privacidad y permite operar en condiciones de baja o nula iluminación. No obstante, las cámaras térmicas tienen dos limitaciones principales: la baja resolución, que reduce la precisión en la estimación de la pose humana, y la escasez de datasets anotados, que dificulta el entrenamiento de modelos de aprendizaje profundo.

A. Avances en la estimación de pose humana

Los modelos tradicionales basados en *Redes Neuronales Convolucionales (CNN)* presentan limitaciones a la hora de capturar las relaciones espaciales globales entre las distintas partes del cuerpo. En una *CNN*, cada filtro convolucional analiza únicamente una pequeña región local de la imagen, por lo que, para que la red pueda "comprender" cómo se relaciona una parte del cuerpo con otra más lejana (por ejemplo, la cabeza con los pies), es necesario apilar múltiples capas convolucionales que combinen progresivamente información de regiones cada vez más amplias. En consecuencia, las relaciones espaciales de largo alcance no se aprenden de manera explícita, sino que quedan codificadas de forma implícita en las activaciones combinadas de muchas capas, lo que dificulta capturar interacciones globales de forma eficiente. Una solución típica, aunque poco elegante, consiste en incrementar la profundidad de la red para ampliar el campo receptivo [1], [2].

Por el contrario, las arquitecturas basadas en *Transformers* han demostrado ser eficaces para superar estas limitaciones de las *CNN* en el manejo de dependencias espaciales distantes. El *Transformer*, originalmente diseñado para procesar texto, puede considerar simultáneamente todas las posiciones de una entrada gracias a su mecanismo de autoaten-

[1]Centro de Investigación en Salud (CEINSA), Universidad de Almería, e-mail: {antoniodetoro, marcoslupion,jsanjuan,vruiz,ortigosa}@ual.es

ción, lo que le permite capturar de forma eficiente dependencias globales y de largo alcance. El mecanismo de autoatención permite que cada elemento de una entrada (como un píxel o palabra) incorpore información de todos los demás. Para ello, cada elemento se representa mediante tres vectores —Query (Q), Key (K) y Value (V)—, y la atención calcula la similitud entre Q y K para asignar pesos a los V, generando así una representación contextualizada que captura dependencias globales dentro de la secuencia o imagen. Por ejemplo, el modelo *TransPose* permite representar de manera explícita las relaciones espaciales globales necesarias para la localización precisa de puntos clave (*keypoints*) en la imagen [3]. La arquitectura *Transformer* ofrece además una ventaja natural para modelar interacciones por pares o de orden superior, ya que una sola capa de autoatención puede capturar la relación entre cualquier par de ubicaciones de la imagen.

Una alternativa prometedora son los *Vision Transformers (ViT)*, cuya estructura es más simple y permite un mayor paralelismo computacional, alcanzando un nuevo equilibrio entre velocidad de inferencia y rendimiento. En los *ViT*, una imagen se trata como una secuencia de pequeños parches, de manera análoga a cómo una oración se representa como una secuencia de palabras, utilizando el mecanismo de autoatención para aprender las relaciones entre dichos parches. Esta simplicidad estructural elimina la necesidad de incorporar conocimientos específicos de dominio en el diseño del codificador *Transformer*. Por ejemplo, el modelo *ViTPose* [4] se ha consolidado como una referencia en los *benchmarks* de gran escala entrenados con el conjunto de datos *COCO* (Common Objects in Context) [5], superando a las arquitecturas convolucionales tradicionales. A diferencia de otros enfoques híbridos que emplean una *CNN* como *backbone* y un *Transformer* como módulo de refinamiento, modelos como *ViTPose* utilizan directamente un *Transformer* de visión para la extracción de características, tanto en paradigmas *top-down* como *bottom-up*. Esto pone de manifiesto su capacidad para obtener representaciones robustas y modelar de manera efectiva las interacciones espaciales de largo alcance, una de las principales debilidades de las *CNN*.

B. Datos sintéticos para HPE térmico

La escasez de conjunto de imágenes térmicas con anotaciones de poses humanas ha impulsado la investigación en generación sintética. Los primeros trabajos utilizaron anotaciones de poses 3D y captura de movimiento (*MoCap*) para generar diferentes poses sin necesidad de anotaciones manuales [6]. Sobre esta base, el aumento de datos guiado por *MoCap* amplía artificialmente el conjunto de datos reales incorporando una gran cantidad de imágenes sintéticas fotorrealistas que representan una mayor diversidad de poses humanas, incluyendo sus anotaciones de poses 3D. Para ello, combina imágenes reales con anotaciones de poses 2D y una biblioteca de datos *MoCap* 3D. El motor de síntesis genera nuevas imágenes seleccionando una pose 3D de la biblioteca y buscando en el conjunto real aquellas imágenes cuyas poses 2D coincidan localmente con la proyección bidimensional de esa pose 3D. A partir de esta correspondencia, se ensambla la nueva imagen mediante la técnica de copy-paste de parches provenientes de las imágenes reales y se aplica un proceso de fusión consciente de la pose para eliminar artefactos. El resultado son imágenes sintéticas cinemáticamente coherentes y visualmente realistas, que enriquecen el entrenamiento de modelos de estimación de pose 3D. Sin embargo, genera artefactos significativos en los límites entre las regiones de imagen unidas. La necesidad de suavizar estos artefactos mediante un algoritmo de fusión es crucial para evitar que la red los interprete como características discriminativas relacionadas con la pose. Además, la estimación de pose falla cuando la pose observada no pertenece a la base de datos de entrenamiento de *MoCap*.

Más recientemente, se han propuesto enfoques avanzados basados en renderizado físico y modelos generativos. Un ejemplo destacado es *ThermalSynth*, que genera alrededor de 80.000 imágenes sintéticas empleando sombreadores térmicos en entornos 3D [7]. El sistema combina imágenes del conjunto LTD (Long-term Thermal Drift dataset), utilizadas como fondos térmicos reales, con personajes sintéticos en formato RGB animados mediante el motor de juego *Unity*. El sombreador térmico aplica la Ley de Stefan-Boltzmann para calcular la radiación de cuerpo negro de cada personaje, transformando su representación RGB original en una imagen térmica simulada. Este proceso aprovecha la información de color y textura de los modelos 3D y la traduce en una firma de calor coherente con los principios de la física térmica, generando resultados visualmente similares a los capturados por una cámara infrarroja. Sin embargo, *ThermalSynth* no está orientado a la detección de poses humanas, sino que entrena una red *CNN* para la clasificación de eventos de caída, limitando su propósito a distinguir entre las categorías "caída" y "no caída".

Otras líneas de investigación se centran en transformar imágenes del espectro visible en imágenes térmicas, mediante la traducción de imágenes RGB a infrarrojas que respeten las leyes físicas. Entre estos enfoques destacan *PETIT-GAN* y *PID*, los cuales incorporan restricciones radiométricas para mejorar el realismo y la coherencia en el dominio térmico [8], [9]. El modelo *PETIT-GAN* es una red generativa adversaria físicamente informada que genera imágenes térmicas sintéticas a partir de imágenes reales, traduciendo una imagen térmica existente a otra que simula cómo se vería la misma escena bajo una modalidad térmica diferente (por ejemplo, monocromática). Por su parte, *PID* es un modelo de difusión informado por principios físicos que introduce restricciones derivadas de las leyes del infrarrojo durante el entrenamiento, haciendo uso de la Ley de Planck y la Ley de Desplazamiento de Wien de forma comple-

mentaria. Esto garantiza una consistencia física en la generación de imágenes infrarrojas a partir de entradas RGB. No obstante, estas metodologías basadas en la transformación del espectro visible al térmico dependen de imágenes visibles en las que se detecten poses humanas, y dicha transformación no garantiza que las poses sigan siendo detectables en las correspondientes imágenes térmicas generadas.

C. Rol de la CPU en la paralelización de GAN

En el entrenamiento distribuido de las *GAN*, los aceleradores como las *GPUs* o *TPUs* concentran la mayor parte de la carga computacional debido al elevado coste de operaciones como las convoluciones y la retropropagación, que implican el cálculo de gradientes y la actualización de millones de parámetros. El proceso principal de entrenamiento, que abarca tanto la propagación hacia adelante como la retropropagación de gradientes y la posterior actualización de los parámetros del generador y del discriminador, se ejecuta íntegramente en las *GPUs*, ya que estas son especialmente eficientes en la ejecución paralela de operaciones matriciales y convolucionales de gran escala. Sin embargo, la *CPU* desempeña un papel crucial en la paralelización, actuando como nodo anfitrión que coordina el flujo de datos, gestiona la entrada/salida (I/O), realiza preprocesamiento y supervisa la sincronización entre los diferentes nodos de cálculo. Este rol es fundamental para mantener la eficiencia del entrenamiento distribuido, garantizar la correcta computación de métricas de evaluación, como el *Fréchet Inception Distance (FID)*, y evitar cuellos de botella en la transferencia de datos entre memoria principal y aceleradores. Así, la *CPU* se convierte en un componente central para la escalabilidad y estabilidad del entrenamiento de GAN a gran escala. Sin embargo, en la práctica, las actuales CPU multinúcleo rara vez operan a plena capacidad durante el entrenamiento, ya que solo un subconjunto de núcleos se utiliza de manera efectiva para la gestión de datos y sincronización, dejando el resto de los recursos ociosos durante la mayor parte del proceso.

Las arquitecturas basadas en *GAN* contienen millones de parámetros y requieren grandes conjuntos de datos, lo que hace que su entrenamiento sea altamente costoso en términos computacionales y temporales. En particular, el entrenamiento de *GAN* mediante *StyleGAN2*, implementado originalmente sobre *TensorFlow*, limita el paralelismo de datos a un único nodo con múltiples GPUs [10]. Para superar esta restricción, se propuso una versión distribuida de *StyleGAN2* que integra *Horovod*, permitiendo el entrenamiento en clústeres multinodo y habilitando tanto el escalado fuerte como el débil de la arquitectura [11].

Los modelos como *BigGAN*, intensivos en cómputo, aceleran su entrenamiento mediante *GPUs* o *TPUs* [12], pero dependen de la *CPU* como nodo anfitrión para coordinar el flujo de datos, realizar preprocesamiento y supervisar el entrenamiento distribuido. De manera similar, *ParaGAN*, un marco escalable para entrenamiento distribuido de *GAN*, utiliza la *CPU* para tareas de gestión, preprocesamiento y operaciones de I/O hacia el almacenamiento, permitiendo un entrenamiento eficiente y escalable de arquitecturas *GAN* a gran escala [13].

D. Nuestra propuesta: Aumentación Adaptativa del Discriminador de Poses (ADAP)

En este trabajo proponemos una *pipeline* paralela y adaptativa para la generación sintética de datos en la estimación de poses humanas térmicas (Thermal Human Pose Estimation, HPE). El objetivo principal es optimizar el uso de los recursos computacionales disponibles, especialmente los núcleos ociosos de la *CPU*, para acelerar y enriquecer el proceso de entrenamiento de las redes generativas.

La *pipeline* aprovecha el tiempo inactivo de la *CPU* para generar imágenes sintéticas térmicas a partir del modelo *GAN* preentrenado en el instante actual. Estas imágenes se analizan mediante *OpenPose* con el fin de detectar y validar poses humanas completas, garantizando que las imágenes generadas sean anatómicamente coherentes y útiles para el entrenamiento [14]. Posteriormente, las imágenes sintéticas validadas se incorporan dinámicamente al conjunto de datos de entrenamiento en el siguiente intervalo de actualización, ampliando de manera continua y controlada la diversidad del dataset.

Para gestionar este proceso, se introduce un mecanismo de *Aumentación Adaptativa del Discriminador de Poses (ADAP)*, inspirado en el principio del *Adaptive Discriminator Augmentation (ADA)* de *StyleGAN3*. *ADAP* ajusta de manera adaptativa la incorporación de imágenes sintéticas basándose en la calidad de las poses detectadas y el rendimiento actual del discriminador, asegurando un equilibrio entre diversidad sintética y estabilidad de entrenamiento.

A continuación, se detallan las principales contribuciones:

- Se propone una pipeline paralela adaptativa que aprovecha los recursos ociosos de la *CPU* para generar y validar imágenes sintéticas durante el entrenamiento, incrementando la eficiencia computacional sin interrumpir el ciclo de entrenamiento adversarial de la *GAN*.
- Se introduce el mecanismo *ADAP*, que extiende el concepto de *ADA* al dominio de la estimación de poses térmicas, integrando la validación estructural de las poses como criterio de aumento adaptativo.
- Se presenta una estrategia de aumento de datos térmicos orientada a la preservación de la privacidad, capaz de enriquecer el conjunto de entrenamiento sin necesidad de recopilar nuevas imágenes reales ni anotaciones manuales adicionales.
- Se demuestra que el uso coordinado de *CPU* y *GPU* mejora la eficiencia del entrenamiento distribuido y la diversidad de las representaciones

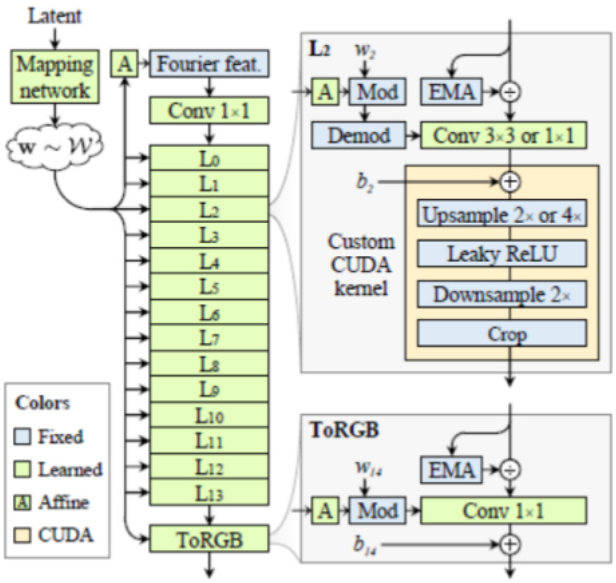


Fig. 1: Arquitectura del generador de *StyleGAN3*, basada en una red GAN que permite un control preciso del proceso de síntesis de imágenes.

de pose, favoreciendo la robustez del modelo generativo final.

II. Metodología

Esta sección describe en detalle la arquitectura y el flujo de trabajo propuesto para la generación y evaluación de imágenes térmicas humanas sintéticas mediante *GAN*. El enfoque se basa en el uso de *StyleGAN3* como arquitectura base [15], adaptada mediante la integración de un módulo de retroalimentación estructural de poses humanas que se propone en este artículo y hemos denominado *StyleGAN3-Pose*.

A. Arquitectura generativa de StyleGAN3

StyleGAN3 es capaz de generar imágenes con continuidad espacial y temporal de alta fidelidad, siendo especialmente efectivo en tareas como síntesis de texturas, generación de rostros o entornos complejos, incluso en aplicaciones de vídeo [16], [17], [18].

La figura 1 muestra la arquitectura interna del generador de *StyleGAN3*, un modelo basado en *GAN* que incorpora mejoras significativas en la estabilidad del entrenamiento y en la calidad espacial y temporal de las imágenes sintetizadas. Esta arquitectura ha sido diseñada específicamente para evitar aliasing, un problema común en versiones anteriores [19], [20], [21], mediante técnicas de modulación espectral y convoluciones cuidadosamente programadas.

En el esquema, el proceso se inicia con un vector latente z, que es transformado por una *Mapping Network* en un vector de estilo w. Este vector de estilo actúa como modulador afín en cada capa de la red de síntesis, lo que permite controlar de manera precisa el contenido visual generado.

El cuerpo principal del generador se compone de 14 bloques convolucionales (de L_0 a L_{13}), que progresivamente refinan los detalles de la imagen. Cada bloque incluye un módulo *ToRGB* que convierte las representaciones intermedias en salidas RGB a distintas escalas. En paralelo, el modelo aplica operaciones como modulación/demodulación, convoluciones 3×3 ó 1×1, normalización y activaciones no lineales (*Leaky ReLU*), muchas de las cuales están optimizadas con *filtros convolucionales* personalizados en CUDA. También se aplica un promedio móvil exponencial (*EMA*) a los pesos del generador, lo que estabiliza el entrenamiento y mejora la calidad final de la imagen generada.

B. Entrenamiento con aumentación adaptativa de datos

El entrenamiento adversarial en *GAN* presenta una alta sensibilidad al sobreajuste del discriminador, especialmente cuando se dispone de conjuntos de datos pequeños o con poca diversidad. En tales escenarios, el discriminador tiende a memorizar las muestras reales, impidiendo que el generador aprenda una distribución generalizable. Para mitigar este problema, *StyleGAN3* incorpora *ADA*, una extensión introducida originalmente en *StyleGAN2-ADA*.

La figura 2 representa el proceso de entrenamiento con la arquitectura original de *StyleGAN3*. Este proceso comienza con la combinación del vector de estilo w y una señal de ruido, que sirven como entrada al generador preentrenado (G), el cual produce imágenes térmicas sintéticas de 128×128 píxeles. Paralelamente, se cargan imágenes reales con resolución de 160×120 píxeles que sirven como referencia para el aprendizaje.

Tanto las imágenes reales como las generadas se someten a un módulo *ADA*, encargado de aplicar aumentos aleatorios para evitar el sobreajuste del discriminador (D). Posteriormente, ambas imágenes son procesadas por la red discriminadora, cuya tarea es clasificar si cada imagen proviene del conjunto real o ha sido generada. La señal de error del discriminador se utiliza para ajustar los pesos del generador, guiando su evolución hacia una mayor capacidad de imitación realista.

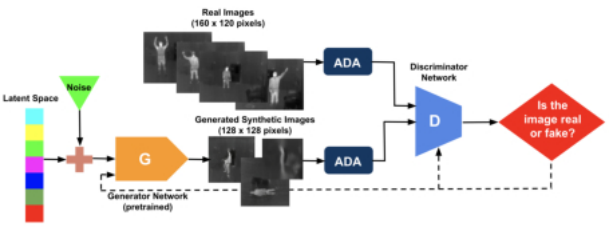


Fig. 2: Esquema del proceso de entrenamiento con la arquitectura original de StyleGAN3.

ADA aplica aumentaciones de datos estocásticas (rotaciones, traslaciones, recortes, cambios de luminancia, entre otras) tanto a las imágenes reales como a las generadas, pero lo hace con una probabilidad dinámica p que se ajusta automáticamente durante el entrenamiento. Este valor de probabilidad se adapta en función del grado de sobreajuste detectado en el discriminador mediante una métrica heurística r_t, que representa la proporción de predicciones "reales" que el discriminador asigna erróneamente (E) a imágenes sintéticas:

$$r_t = E\left[D(G(z)) > 0\right], \tag{1}$$

donde $D(G(z))$ indica la salida del discriminador ante una imagen generada por el generador G a partir de un vector de ruido z.

El objetivo de *ADA* es mantener r_t cerca de un valor objetivo r_{target} (típicamente 0.6). Para ello, se actualiza la probabilidad de aumento p en cada iteración según una regla de control proporcional:

$$p_{t+1} = p_t + k \cdot (r_t - r_{target}), \tag{2}$$

donde k es un factor de ajuste que controla la velocidad de adaptación. Si el discriminador muestra signos de sobreajuste (por ejemplo, $r_t < r_{target}$), se incrementa la cantidad de aumentaciones; si, por el contrario, no se observan indicios de sobreajuste, la probabilidad se reduce.

Este mecanismo adaptativo permite mantener el equilibrio entre el generador y el discriminador a lo largo del entrenamiento, evitando el colapso del modelo y mejorando la calidad y estabilidad de las imágenes sintetizadas. En consecuencia, *StyleGAN3* logra un entrenamiento más robusto, capaz de producir imágenes de alta fidelidad espacial incluso en contextos con conjuntos de datos limitados.

Por otro lado, se debe remarcar que el progreso del entrenamiento se mide en *kimg*, que representa mil imágenes procesadas por el generador. Si B es el tamaño de lote (*batch size*) e I el número de iteraciones completas, el número total de *kimg* se calcula como:

$$kimg = \frac{B \cdot I}{1000}. \tag{3}$$

El entrenamiento de StyleGAN3 se ejecuta de forma intensiva en la GPU; sin embargo, el sistema delega la gestión de los puntos de control (snapshots) a la CPU. Siguiendo las recomendaciones para datasets pequeños (menos de 10,000 imágenes a 128x128 px), se establece un volumen total de entrenamiento de entre 5,000 y 10,000 kimg. Por defecto, cada 50 ticks (equivalentes a 200 kimg procesados), la CPU almacena un archivo de pesos (.pkl) del modelo entrenado hasta ese momento y una cuadrícula de imágenes sintéticas de muestra. Estos intervalos permiten monitorizar la evolución del modelo mediante la métrica FID (Fréchet Inception Distance). Al comparar la distribución de características latentes entre imágenes reales y generadas, se puede identificar el punto de overfitting (sobreajuste) y determinar el momento óptimo para la finalización del proceso.

C. Integración de StyleGAN3-Pose

La Aumentación Adaptativa del Discriminador de Poses (*ADAP*), propuesta en este artículo, extiende el mecanismo original de StyleGAN3 mediante un flujo de trabajo asíncrono. Mientras la *GPU* ejecuta el entrenamiento principal, la *CPU* aprovecha los snapshots (puntos de control) generados periódicamente para sintetizar nuevas imágenes y analizar su estructura mediante el algoritmo de detección de poses humanas. El sistema filtra y selecciona únicamente aquellas imágenes en las que se detecta una anatomía completa y coherente, reintegrándolas al dataset de entrenamiento. Esta ejecución en paralelo optimiza los recursos de cómputo, permitiendo que la regularización semántica de *ADAP* mejore la estabilidad del modelo sin interrumpir el ciclo de aprendizaje en la *GPU*.

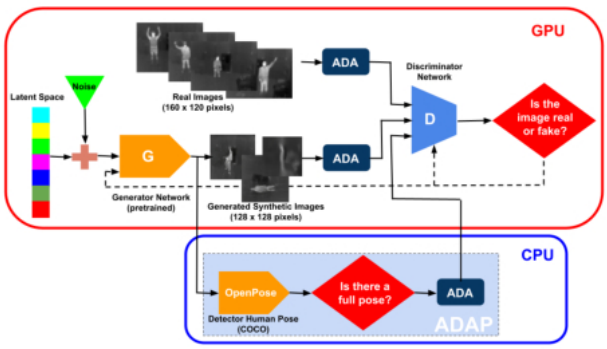


Fig. 3: Esquema de entrenamiento con *StyleGAN3-Pose*, que combina generación adversaria y HPE.

La figura 3 presenta el flujo de entrenamiento extendido con *StyleGAN3-Pose*, donde el generador (G) sigue produciendo imágenes térmicas sintéticas, pero ahora se incorpora un sistema paralelo ejecutado en la *CPU* para analizar la calidad anatómica de estas imágenes.

Las imágenes generadas, junto con las reales, se procesan mediante *ADA* y se introducen en el discriminador (D). En paralelo, las imágenes sintéticas se transfieren a la *CPU*, donde se aplican técnicas de detección de pose humana completa usando *OpenPose*. Este análisis es gestionado por el módulo *ADAP*, que descarta aquellas imágenes en las que no se detecta ninguna pose completa, usando las imágenes sintéticas con poses completas para reentrenar el modelo. Esta retroalimentación permite ajustar dinámicamente el entrenamiento, favoreciendo la generación de imágenes con mayor coherencia anatómica.

Con esta propuesta, el entrenamiento de *StyleGAN3* se apoya en un marco de ejecución paralela altamente optimizado que distribuye las tareas entre la *CPU* y la *GPU* con el objetivo de maximizar la eficiencia computacional y la escalabilidad. El framework de paralelización en *StyleGAN3* está diseñado para aprovechar arquitecturas heterogéneas y configuraciones multi-GPU, utilizando bibliotecas de comunicación optimizadas como *NCCL* (NVIDIA Collective Communications Library) para la agregación eficiente de gradientes y la sincronización de parámetros entre dispositivos. Este enfoque permite escalar el entrenamiento de manera casi lineal con el número de *GPU*, manteniendo la estabilidad numérica del modelo y minimizando la sobrecarga de comunicación. La *CPU*, aunque no participa directamente en el cálculo de los gradientes, desempeña un papel esencial en la alimentación continua del pipeline de entrenamiento, asegurando que las *GPU* operen sin interrupciones. Además, coordina tareas como el cálculo de métricas de evaluación (por ejemplo,

FID y la escritura de snapshots del modelo, funciones críticas para la monitorización y reproducibilidad del entrenamiento.

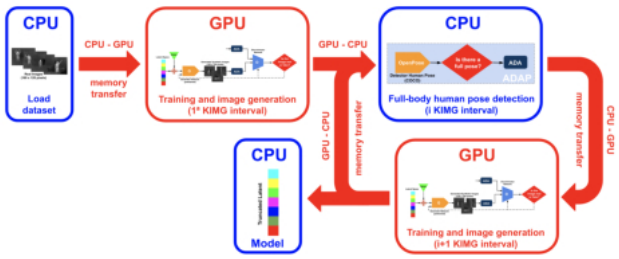


Fig. 4: Flujo de entrenamiento de *StyleGAN3-Pose* con ejecución paralela en CPU/GPU.

La figura 4 detalla la arquitectura operativa paralela de *StyleGAN3-Pose*. En este enfoque, las tareas se dividen entre la *GPU* y la *CPU*. La *GPU* se encarga del entrenamiento del modelo y la síntesis de imágenes en cada intervalo, mientras que la *CPU* realiza tareas auxiliares críticas como la carga del conjunto de datos, guardar copia del modelo preentrenado y la detección de poses completas. Tras cada ciclo de entrenamiento, las imágenes generadas se transfieren a la *CPU*, donde se evalúan estructuralmente. Esta evaluación determina si las imágenes cumplen con los criterios anatómicos establecidos. La retroalimentación resultante se utiliza para influir en el siguiente ciclo de entrenamiento, cerrando así un bucle de mejora continua.

D. Deteccion de postura con OpenPose en ADAP

Tras generar las imágenes sintéticas mediante *StyleGAN3-Pose*, estas se transfieren a la *CPU* para su análisis con *OpenPose*, empleando el modelo preentrenado con el conjunto de datos *COCO*. Como se ilustra en la figura 5, el sistema identifica hasta 18 puntos clave (keypoints) del cuerpo humano; para los fines de este estudio, se define una pose completa cuando se detectan con éxito al menos 15 de ellos.

Posteriormente, el módulo *ADAP* recibe como entrada las imágenes anotadas y aplica una regla de decisión basada en la presencia o ausencia de una pose humana completa. Las imágenes que no cumplen este criterio son descartadas del conjunto de imágenes sintéticas útiles y no se consideran para la retroalimentación estructural del modelo. La figura 6 muestra tres ejemplos de imagenes térmicas diferentes en las que se ha detectado una pose completa, parcial y no se ha detectando ninguna pose.

Este pipeline no solo cuantifica la proporción de imágenes anatómicamente válidas, sino que establece un mecanismo de retroalimentación estructural que mejora de forma adaptativa la calidad del generador. Al penalizar implícitamente las configuraciones latentes que producen figuras incompletas o incoherentes, el sistema refuerza la generación de contenido sintético con mayor realismo estructural. La figura 7 ejemplifica esta evolución mediante imágenes sintéticas en las que se detecta una pose completa, obtenidas con modelos entrenados tras procesar 100, 500 y 1000 kimg.

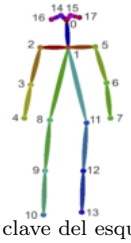


Fig. 5: Puntos clave del esqueleto humano.

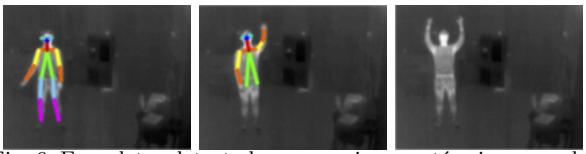


Fig. 6: Esqueletos detectados en una imagen térmica: completo, parcial o no detectado.



Fig. 7: Imágenes sintéticas generadas con modelos entrenados con 100 kimg (izquierda), 500 kimg (centro) y 1000 kimg (derecha).

III. Experimentacion

En esta sección se detalla el marco experimental diseñado para evaluar la capacidad de generación de imágenes térmicas y la detección de poses mediante las arquitecturas *StyleGAN3* y *StyleGAN3-Pose*. El proceso abarca desde la configuración técnica de los hiperparámetros del espacio latente hasta el tratamiento del conjunto de datos *UAL-Smarthome*. A continuación, se describen los ajustes específicos de la red de mapeo, los criterios de truncamiento y las fases de preprocesamiento aplicadas para garantizar la calidad y coherencia de las representaciones sintéticas obtenidas.

A. Configuración inicial de parámetros

Durante el entrenamiento del modelo *StyleGAN3*, se utilizaron las siguientes configuraciones del espacio latente:

- Profundidad del mapeo: 8 capas totalmente conectadas en la red de mapeo para capturar relaciones complejas entre atributos visuales.
- Estrategia de truncamiento: Se aplicó truncamiento mediante interpolación entre w y la media de w, con un coeficiente $\psi = 0{,}7$ durante la generación de imágenes sintéticas para asegurar diversidad manteniendo la coherencia estructural.
- Inicialización de semillas: Se usaron semillas determinísticas en intervalos controlados para evaluar la evolución del modelo y generar conjuntos comparables a lo largo del entrenamiento.

En *StyleGAN3-Pose* se emplea la misma configuración del espacio latente utilizada en el *StyleGAN3*

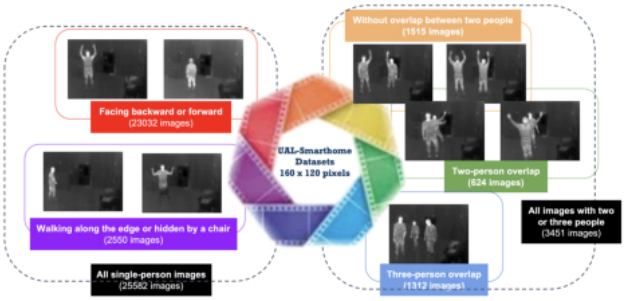


Fig. 8: Conjuntos de datos de imágenes térmicas de baja resolución de *UAL-Smarthome*.

original, con el fin de garantizar la comparabilidad de resultados y la validez de las métricas aplicadas. La única diferencia es que, mientras *StyleGAN3* define un número total de iteraciones *kimg*, en *StyleGAN3-Pose* se especifica un intervalo de evaluación mediante el parámetro $kimg_{interval}$, por ejemplo, 4, 8, 16 u 32 kimg. Para cada intervalo, se generó un lote de 1000 imágenes sintéticas a partir de 1000 semillas aleatorias distintas en el espacio $\mathcal{Z}$, garantizando una exploración diversa del espacio latente. Posteriormente, estas imágenes sintéticas creadas se utilizaron para la detección de poses humanas mediante *OpenPose*.

B. Conjunto de datos y preprocesamiento

El conjunto de datos utilizado en este estudio corresponde a *UAL-Smarthome*, una colección de imágenes térmicas adquiridas en un entorno interior, donde se representan distintos escenarios de interacción humana con una o más personas en escena [22], [23]. Las imágenes presentan una resolución original de 160×120 píxeles, y fueron capturadas mediante sensores térmicos de bajo coste instalados en la vivienda inteligente experimental. La figura 9 muestra la cámara PureThermal 2 Smart I/O Module + FLIR Lepton 3.5.

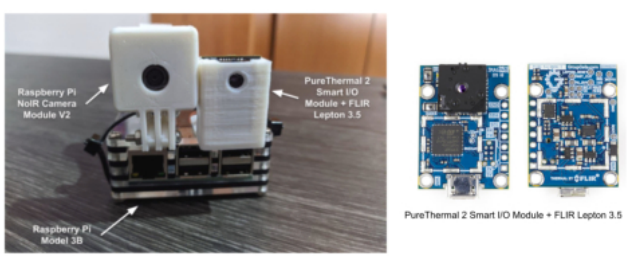


Fig. 9: Cámaras.

Para su uso en el entrenamiento con *StyleGAN3* y *StyleGAN3-Pose*, se realizó una fase de preprocesamiento que incluyó los siguientes pasos:

- Filtrado de imágenes por contenido: Se eliminaron aquellas imágenes que no contenían personas o que mostraban condiciones térmicas excesivamente degradadas (baja emisión, superposición extrema o ruido por reflejos).
- Normalización de resolución: Las imágenes fueron redimensionadas a una resolución cuadrada de 128×128 píxeles mediante interpolación bilineal. Esta resolución fue elegida por ser compatible con la arquitectura *StyleGAN3* y representar un equilibrio entre nivel de detalle y coste computacional.
- Organización en subconjuntos: El dataset fue dividido en subconjuntos temáticos según la configuración de la escena y la interacción de los individuos: (1) una persona de espaldas o de frente, (2) una persona parcialmente oculta o en los bordes de la imagen, (3) dos personas sin solapamiento, (4) dos personas superpuestas, (5) tres personas superpuestas y (6) un conjunto total con dos o más personas.
- Aumento de datos: Durante el entrenamiento se aplicaron aumentos dinámicos mediante el módulo *ADA*, incluyendo rotaciones, traslaciones, escalado, inversiones horizontales y transformaciones de contraste. Esto permitió mejorar la generalización del modelo, especialmente en subconjuntos reducidos.

Los subconjuntos de imágenes seleccionadas fueron utilizados tanto para la fase de entrenamiento usando *StyleGAN3*, como para la evaluación con *OpenPose* usando *StyleGAN3-Pose*.

C. Resultados

Se entrenaron distintos modelos utilizando cada uno de los conjuntos de datos *UAL-Smarthome* tanto con la implementación original de *StyleGAN3*, como con *StyleGAN3-Pose*. El hardware utilizado para el entrenamiento incluía un procesador *Intel(R) Core(TM) i5-14400* con 16 núcleos, 64 GB de memoria RAM y una GPU NVIDIA RTX 4060 Ti con 16 GB de memoria, ejecutando el sistema operativo Ubuntu 24.04.1 LTS.

La figura 10 muestra el porcentaje de uso promedio de la *CPU* y de la *GPU* durante el entrenamiento del modelo utilizando la arquitectura original de *StyleGAN3*, correspondiente a los primeros 32 kimg. A lo largo del proceso se identifican claramente cinco fases diferentes:

1. Carga inicial del snapshot del modelo.
2. Entrenamiento continuo del generador y discriminador, caracterizado por una alta demanda computacional en la *GPU*.
3. Generación de imágenes sintéticas.
4. Evaluación estructural mediante detección de poses humanas completas con *OpenPose*, lo cual incrementa significativamente la carga sobre la *CPU*.
5. Guardado del snapshot del modelo, tras finalizar el intervalo.

Durante la mayor parte del entrenamiento, la *GPU* mantiene una utilización estable cercana al 100 %, lo que refleja su papel predominante en la computación de la red generativa. En cambio, la utilización de la *CPU* es más variable, con picos de actividad asociados a las fases que implican procesamiento fuera del flujo adversarial, como la evaluación estructural (fase 4) y las operaciones de entrada/salida (fases 1 y 5). Esta diferenciación en el uso de recursos ilustra

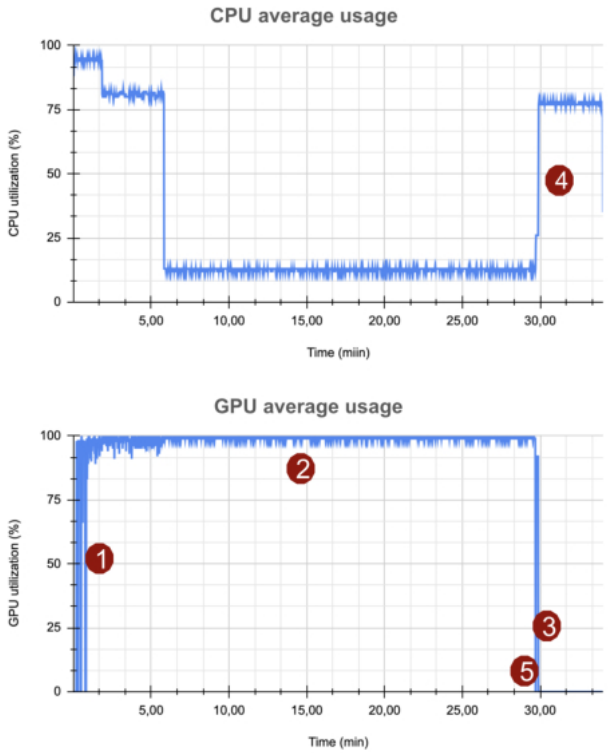


Fig. 10: Uso promedio de *CPU* (arriba) y uso de *GPU* (abajo) durante el entrenamiento del modelo utilizando la arquitectura original de *StyleGAN3*, correspondiente a los primeros 32 kimg.

claramente el reparto de tareas entre *CPU* y *GPU* durante el entrenamiento con *StyleGAN3*.

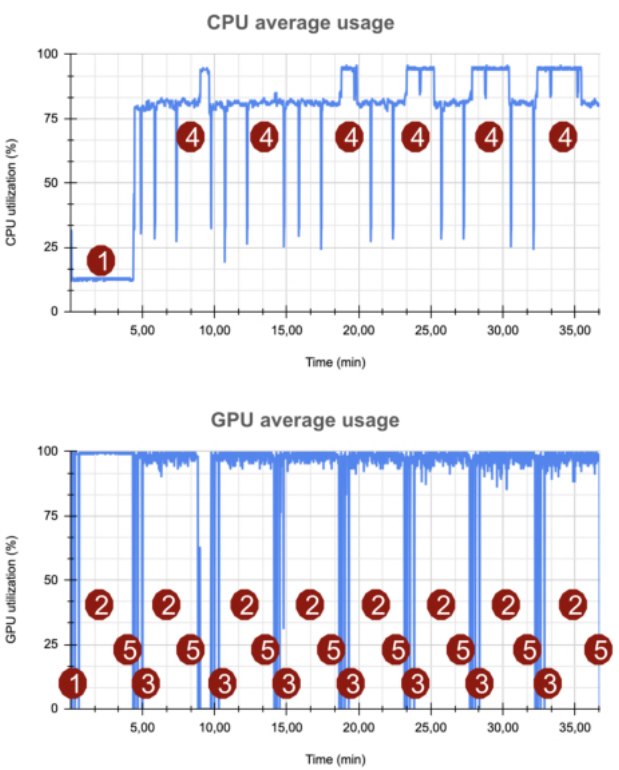


Fig. 11: Utilización promedio de *CPU* (arriba) y *GPU* (abajo) durante el entrenamiento en paralelo del modelo *StyleGAN3-Pose*, a lo largo de los primeros 32 kimg con pasos de intervalo de 4 kimg.

La figura 11 muestra el porcentaje de uso promedio

Tabla I: Estadísticas de entrenamiento sobre el conjunto completo de imágenes reales de los datasets de *UAL-Smarthome*, utilizando el *StyleGAN3* original entrenado durante 500 kimg.

UAL-Smarthome Datasets	N. personas	N. imágenes reales 160×120	% Pose completa 500 kimg
De frente o de espalda	1	23032	42,40 %
Andando por el borde o oculto por una silla	1	2550	28,51 %
Imágenes con una sola persona	1	25582	41,01 %
Sin solapamiento entre personas	2	1515	39,87 %
Solapamiento de dos personas	2	624	14,42 %
Solapamiento de tres personas	3	1312	12,80 %
Imágenes con dos o tres personas	2-3	3451	24,98 %

de la *CPU* y de la *GPU* durante el entrenamiento del modelo *StyleGAN3-Pose* ejecutado en modo paralelo, a lo largo de los primeros 32 kimg con intervalos de evaluación cada 4 kimg. Este enfoque combina el entrenamiento sobre *GPU* con un módulo adicional de evaluación estructural ejecutado en la *CPU*, encargado de detectar poses humanas completas en las imágenes sintéticas generadas, utilizando *OpenPose* y el módulo *ADAP*.

A diferencia del comportamiento observado en el entrenamiento con la arquitectura original de *StyleGAN3*, aquí se aprecia una actividad más constante y elevada en la *CPU*, como consecuencia del proceso de análisis estructural paralelo, que se activa de forma cíclica tras cada intervalo de entrenamiento. Esta carga se traduce en picos de utilización asociados a la evaluación de las poses generadas, mientras que el resto del tiempo la *CPU* mantiene un nivel de uso intermedio constante.

Por su parte, la *GPU* presenta una utilización sostenida cercana al 100 % durante las fases de entrenamiento adversarial, interrumpida únicamente por breves descensos asociados a la generación de imágenes sintéticas y el almacenamiento del snapshot del modelo preentrenado. En conjunto, esta distribución de carga evidencia la sincronización eficiente y el reparto funcional de tareas entre *CPU* y *GPU* que caracteriza al enfoque paralelo de *StyleGAN3-Pose*, permitiendo aprovechar mejor los recursos computacionales disponibles y mejorar la coherencia estructural de las imágenes generadas.

El entrenamiento de los distintos modelos con cada uno de los conjuntos de datos *UAL-Smarthome* usando *StyleGAN3* se realizaron hasta completar 500 kimg. Posteriormente, se generaron 1000 imágenes sintéticas con cada modelo. La tabla I muestra, para cada conjunto de datos, el número total de imágenes térmicas reales con una resolución de 160×120 píxeles y el porcentaje de imágenes sintéticas creadas con el respectivo modelo entrenado hasta completar 500 kimg con al menos una pose completa detectada.

La detección de poses humanas completas en las imágenes creadas por el modelo reveló que aproximadamente un 40 % de las imágenes sintéticas contenían poses completas en tres conjuntos de datos, alrededor de un 25 % en dos, y menos del 15 % en los dos restantes. Estos resultados demuestran que ninguno de los modelos entrenados logró generar imágenes sintéticas en las que más del 50 % de las imágenes contuvieran poses humanas completas.

La tabla II muestra el porcentaje de imágenes sintéticas creadas con los modelos entrenados has-

Tabla II: Estadísticas de entrenamiento utilizando únicamente las imágenes reales de los datasets de *UAL-Smarthome* en las que se ha detectado una postura completa, con el *StyleGAN3* original entrenado durante 500 y 1000 kimg, respectivamente.

UAL-Smarthome Datasets	N. imágenes con poses 128×128 pixels	% Pose completa 500 kimg	% Pose completa 1000 kimg
De frente o de espalda	8165	53,40 %	56,20 %
Andando por el borde o oculto por una silla	565	34,90 %	29,10 %
Imágenes con una sola persona	8730	70,70 %	59,40 %
Sin solapamiento entre personas	259	46,70 %	61,20 %
Solapamiento de dos personas	114	50,00 %	64,10 %
Solapamiento de tres personas	144	7,60 %	10,60 %
Imágenes con dos o tres personas	517	42,50 %	55,40 %

Tabla III: Análisis comparativo de las métricas de entrenamiento por conjunto de datos entrenado con *StyleGAN3-Pose*, considerando diferentes snapshots.

Modelos UAL-Smarthome	Intervalo del snapshot $KIMG_{intervalo}$	Promedio # imág. sint.	% Pose completa
De frente o de espalda	4	386	**80,71 %**
	8	530	**80,38 %**
	16	743	70,25 %
	32	1036	55,50 %
Andando por el borde o oculto por una silla	4	386	**75,21 %**
	8	526	**91,25 %**
	16	740	**79,00 %**
	32	1050	**79,50 %**
Imágenes con una sola persona	4	385	**80,57 %**
	8	528	**88,88 %**
	16	740	72,00 %
	32	1050	51,00 %
Sin solapamiento entre personas	4	385	**83,36 %**
	8	526	**75,63 %**
	16	736	**86,00 %**
	32	1050	67,00 %
Solapamiento de dos personas	4	383	**91,14 %**
	8	526	**95,50 %**
	16	737	**91,25 %**
	32	1043	**82,00 %**
Solapamiento de tres personas	4	381	**77,36 %**
	8	522	**83,63 %**
	16	738	70,00 %
	32	1043	54,00 %
Imágenes con dos o tres personas	4	381	**87,36 %**
	8	525	63,25 %
	16	737	**81,00 %**
	32	1043	39,00 %

ta 500 kimg y 1000 kimg, respectivamente, que contenían al menos una pose humana completa detectada. En este caso, los modelos se entrenaron exclusivamente con las imagenes que realmente tenían al menos una pose completa detectada y redimensionadas a 128×128 píxeles. Durante el entrenamiento, se realiza un snapshot del modelo en cada intervalo de kimg especificado; por defecto, este valor es 50, lo que corresponde a 200 kimg. Se debe destacar, que el tiempo de entrenamiento usando *StyleGAN3* por cada mil imágenes es de 2 min/kimg independientemente del dataset utilizado en el entrenamiento.

Posteriormente se entrenaron otros modelos sobre cada uno de los conjuntos de datos *UAL-Smarthome*, utilizando la versión paralela de *ADAP* integrada en *StyleGAN3-Pose* y se generaron 1000 imágenes sintéticas con cada modelo. En este caso, cada vez que se realiza el snapshot del modelo, se generan imágenes sintéticas utilizando el modelo entrenado hasta ese punto, además de guardar una copia del modelo preentrenado en disco. De forma paralela, se ejecuta *ADAP*, que detecta las imágenes generadas y analiza las poses humanas completas mediante *OpenPose*, eliminando aquellas imágenes sintéticas en las que no se haya detectado ninguna pose completa, y reutilizando el resto de imágenes sintéticas para el entrenamiento del modelo en el siguiente intervalo.

La tabla III muestra una comparativa del entrenamiento de los modelos con distintos intervalos de snapshot (4, 8, 16 y 32 kimg). Para cada conjunto de datos, se incluye el número promedio de imágenes sintéticas generadas por intervalo (Avg. # synth. images), y el porcentaje de imágenes con poses completas detectadas (% Full pose) en las 1000 imágenes generadas con el modelo final entrenado, resaltando en negrita las que superen el 75 %.

En este caso, se observa que el porcentaje de imágenes sintéticas con poses humanas completas supera considerablemente al obtenido usando *StyleGAN3*. Además, conforme se aumenta el intervalo snapshot, se aumenta el número de imágenes sintéticas generadas en cada intervalo debido a que el tiempo de entrenamiento para cada intervalo es mayor, sin embargo el número de imágenes sintéticas acumuladas es inferior ya que se realizan menos snapshots durante el entrenamiento.

IV. Conclusiones

El uso de imágenes térmicas, junto con técnicas de aumento adaptativo, se presenta como una alternativa interesante y más respetuosa con la privacidad frente a las cámaras RGB tradicionales, ya que permite analizar la pose humana sin revelar la identidad de las personas. Sin embargo, al trabajar con imágenes térmicas de baja resolución (128×128), aparecen limitaciones importantes en *StyleGAN3*, especialmente cuando se intenta detectar poses en las imágenes generadas.

Aun así, se ha comprobado que eliminar del conjunto de entrenamiento aquellas imágenes en las que no se detectan poses, y aumentar el número de iteraciones (kimg), puede mejorar ligeramente los resultados. No obstante, este aumento no siempre es beneficioso: en algunos casos aparece sobreentrenamiento (*overfitting*). Por ejemplo, en la tabla II, el modelo entrenado con todas las imágenes de una persona empeora al pasar de 500 a 1000 kimg (del 70.7 % al 59.4 %). Esto indica que entrenar durante más tiempo no garantiza mejores resultados, ya que el modelo puede empezar a generar imágenes con errores que *OpenPose* no es capaz de interpretar correctamente.

Para abordar estas limitaciones, se ha propuesto *StyleGAN3-Pose*. Esta mejora aprovecha el tiempo en el que la *CPU* está inactiva mientras la *GPU* entrena el modelo, generando nuevas imágenes a partir de los *snapshots*. Las imágenes sintéticas en las que se detectan poses completas se incorporan al conjunto de entrenamiento en la siguiente fase. Gracias a este enfoque paralelo, se alcanzan porcentajes superiores al 75 % de imágenes con poses completas en todos los subconjuntos del dataset *UAL-Smarthome*, especialmente cuando se utilizan intervalos de *snapshot* pequeños (4 kimg u 8 kimg).

Además, esta arquitectura permite aprovechar mejor los recursos disponibles. Al delegar la detección de poses en la *CPU*, se puede trabajar en paralelo y mantener la *GPU* prácticamente al 100 % de uso durante el entrenamiento.

Los resultados más bajos se obtienen en el caso de imágenes con tres personas, donde el rendimiento se sitúa entre el 7.6 % y el 10.6 %. Esto muestra que el problema se vuelve mucho más complejo a medida que aumenta el número de personas en la escena.

En particular, al modelo le cuesta mantener la coherencia cuando hay varios cuerpos interactuando, con oclusiones y solapamientos.

En definitiva, aunque *StyleGAN3-Pose* mejora los resultados, generar escenas realistas con varias personas sigue siendo un desafío abierto. Actualmente, el sistema solo es capaz de detectar una pose completa por imagen, por lo que una posible línea futura sería extender este enfoque para identificar múltiples poses simultáneamente.

Agradecimientos

Este trabajo ha sido financiado por el proyecto de I+D+i PID2024-155521OB-I00 del MCIN/AEI/10.13039/501100011033 y por fondos ERDF. Además, este trabajo recibió financiación del proyecto CPUENTE2024/05 de la Universidad de Almería.

Referencias

[1] Tomas Pfister, James Charles, and Andrew Zisserman, "Flowing convnets for human pose estimation in videos," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015.

[2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever, "Language models are unsupervised multitask learners," Tech. Rep., OpenAI, 2019.

[3] Sen Yang, Zhibin Quan, Meng Nie, and Wankou Yang, "Transpose: Keypoint localization via transformer," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[4] Yufei Xu, Jing Zhang, Qiang Zhang, and Dacheng Tao, "Vitpose++: Vision transformer for generic body pose estimation," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[5] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár, "Microsoft coco: Common objects in context," in *European Conference on Computer Vision (ECCV)*, 2015.

[6] Grégory Rogez and Cordelia Schmid, "Mocap-guided data augmentation for 3d pose estimation in the wild," in *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[7] Nikhil Madan, Prashant Garg, Sarthak Kaul, and Ankit Mahajan, "Thermalsynth: A novel approach for generating synthetic thermal human scenarios," in *IEEE/CVF Winter Conference on Applications of Computer Vision Workshops (WACVW)*, 2023, pp. 130–139.

[8] Oded Berman, Noam Oz, Dan Mendlovic, Nir Sochen, Yuval Cohen, and Itamar Klapp, "Petit-gan: Physically enhanced thermal image-translating gan," in *IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2024, pp. 1607–1616.

[9] Fei Mao, Jiaqi Mei, Sheng Lu, Fei Liu, Liang Chen, Feng Zhao, and Yao Hu, "Pid: Physics-informed diffusion model for infrared image generation," *arXiv preprint*, vol. arXiv:2407.09299, 2024.

[10] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila, "Analyzing and improving the image quality of stylegan," *arXiv preprint*, vol. arXiv:1912.04958, 2020.

[11] Nicola A. Loppi and Tuomas Kynkäänniemi, "Multi-node training for stylegan2," in *Pattern Recognition. ICPR International Workshops and Challenges*. 2021, vol. 12661 of *Lecture Notes in Computer Science*, Springer.

[12] Andrew Brock, Jeff Donahue, and Karen Simonyan, "Large scale gan training for high fidelity natural image synthesis," *arXiv preprint*, vol. arXiv:1809.11096, 2019.

[13] Zhiwei Shi, Jing Li, and Yang You, "Paragan: A scalable distributed training framework for generative adversarial networks," in *ACM Symposium on Cloud Computing (SoCC)*. 2024, pp. 1032–1044, Association for Computing Machinery.

[14] Tatsuya Ino, Masaki Samukawa, Takumi Ishida, Naoya Wada, Yuta Koshino, Shingo Kasahara, and Hiroshi Tohyama, "Validity and reliability of openpose-based motion analysis in measuring knee valgus during drop vertical jump test," *Journal of Sports Science and Medicine*, vol. 23, no. 1, pp. 515–525, 2024.

[15] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönnen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila, "Alias-free generative adversarial networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[16] Yuval Alaluf, Or Patashnik, Zhizhong Wu, Amir Zamir, Eli Shechtman, Daniel Lischinski, and Daniel Cohen-Or, "Third time's the charm? image and video editing with stylegan3," *arXiv preprint*, vol. arXiv:2201.13433, 2022.

[17] A. I. Humayun, Randall Balestriero, and Richard Baraniuk, "Polarity sampling: Quality and diversity control of pre-trained generative networks via singular values," *arXiv preprint*, vol. arXiv:2203.01993, 2022.

[18] Sourav Das and Pankaj Walia, "Enhancing early diabetic retinopathy detection through synthetic dr1 image generation: A stylegan3 approach," *arXiv preprint*, vol. arXiv:2501.00954, 2025.

[19] Tero Karras, Samuli Laine, and Timo Aila, "A style-based generator architecture for generative adversarial networks," *arXiv preprint*, vol. arXiv:1812.04948, 2019.

[20] Ricard Durall, Margret Keuper, and Janis Keuper, "Watch your up-convolution: Cnn based generative deep neural networks are failing to reproduce spectral distributions," *arXiv preprint*, vol. arXiv:2003.01826, 2020.

[21] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila, "Training generative adversarial networks with limited data," *arXiv preprint*, vol. arXiv:2006.06676, 2020.

[22] Marcos Lupión, Vicente González-Ruiz, Juan F. Sanjuan, and Pilar M. Ortigosa, "Privacy-aware fall detection and alert management in smart environments using multimodal devices," *Internet of Things*, vol. 30, 2025.

[23] Marcos Lupión, Aurora Polo-Rodríguez, Javier Medina-Quero, Juan F. Sanjuan, and Pilar M. Ortigosa, "3d human pose estimation from multi-view thermal vision sensors," *Information Fusion*, vol. 104, 2024.

Planificación de tareas moldeables en GPUs dinámicamente reconfigurables mediante Aprendizaje por Refuerzo Profundo

Jorge Villarrubia[1], Luis Costero[1], Francisco D. Igual[1] y Katzalin Olcoz[1]

Resumen— **Los avances en particionado dinámico de GPUs, como la tecnología Multi-Instance GPU (MIG) de NVIDIA, mejoran la utilización de recursos en centros de datos al permitir la co-ejecución de tareas en instancias aisladas. Sin embargo, los planificadores MIG actuales suelen basarse en métodos estáticos o heurísticas agnósticas de las características de las tareas, sacrificando optimalidad por tratabilidad. Este artículo presenta un planificador para MIG entrenado mediante Aprendizaje por Refuerzo Profundo (DRL) que minimiza el tiempo de finalización (*makespan*) de una cola de tareas en una GPU MIG. A diferencia de enfoques previos, nuestra propuesta aborda de forma holística el moldeado de tareas según los recursos de la GPU, la reconfiguración dinámica del dispositivo y el orden de ejecución. Para gestionar el enorme espacio de soluciones, introducimos optimizaciones como la representación canónica del estado, el enmascaramiento de acciones inviables y un mecanismo para fomentar la exploración de reconfiguraciones complejas. Estas técnicas hacen factible el enfoque y ofrecen ideas valiosas para problemas similares. El modelo se evalúa con *benchmarks* de las *suites* Rodinia y Altis, y con cargas sintéticas que emulan situaciones realistas. Los resultados muestran que el agente DRL mejora consistentemente al estado del arte, especialmente en cargas mixtas donde las simplificaciones previas fallan, logrando planificaciones cuya diferencia respecto al óptimo es inferior al 20 % (en promedio).**

Palabras clave— ***Multi-Instance GPU* (MIG), Gestión de Recursos Moldeables, Aprendizaje por Refuerzo Profundo, Planificación de Tareas**

I. Introducción

El crecimiento exponencial de la Inteligencia Artificial (IA), en particular de los Grandes Modelos de Lenguaje (LLMs), ha disparado la demanda de capacidad de cómputo paralelo. Las GPUs modernas para centros de datos, con miles de núcleos y cientos de gigabytes de memoria, ofrecen un rendimiento de varios PetaFLOPS. Sin embargo, una proporción significativa de las cargas de trabajo, como las provenientes del ámbito de la computación de altas prestaciones (HPC) [1,2] o ciertas tareas de *Deep Learning* (DL) [3], no logra utilizar estos recursos de manera eficiente. Esta infrautilización no solo alarga los tiempos de ejecución de las tareas, sino que también conlleva un desperdicio energético considerable debido al alto consumo de estos dispositivos.

Una solución natural para mejorar la utilización de la GPU es la ejecución concurrente de aplicaciones independientes. Para ello, NVIDIA ofrece tecnologías como *Multi-Process Service* (MPS) [4] y, más recientemente, *Multi-Instance GPU* (MIG) [5]. Mientras que MPS comparte algunos recursos como el ancho de banda de memoria entre las aplicaciones concurrentes, provocando en algunos casos contención y problemas de calidad de servicio [1, 6], MIG adopta un enfoque basado en el aislamiento, particionando la GPU en instancias virtuales independientes. Cada instancia dispone de recursos dedicados (*Streaming Multiprocessors*, caché L2, memoria DRAM y canales de memoria), garantizando un rendimiento predecible y un aislamiento total [6, 7]. Una característica crucial de MIG es su soporte para la reconfiguración dinámica, permitiendo modificar una instancia virtual durante la ejecución de otras sin interferir con ellas, a diferencia de tecnologías de particionado estático como AMD MxGPU [8].

Para los proveedores de servicios en la nube (CSPs), MIG es una herramienta prometedora para entornos multiinquilino. Sin embargo, su uso se ha limitado tradicionalmente a un particionado estático y ajeno a las características de las tareas entrantes. Aunque sistemas de gestión de trabajos como SLURM y HTCondor, u orquestadores como Kubernetes, ya soportan MIG, sus políticas de planificación no integran estrategias que combinen la reconfiguración dinámica con una planificación consciente de las tareas. Reconfigurar una partición MIG conlleva un pequeño overhead temporal [2], pero este coste suele ser rentable, ya que permite adaptar la GPU a las necesidades cambiantes de las tareas pendientes.

Esto da lugar a un problema de planificación de tareas en un sistema reconfigurable, que requiere determinar tanto el orden de ejecución de las tareas, como la partición de la GPU y las instancias concretas de la partición que se asignarán a cada tarea. En la literatura, este paradigma se conoce como planificación de tareas moldeables (*moldable scheduling*), ya que el planificador decide la cantidad de recursos que asigna a cada tarea, en lugar de recibir esta asignación como un dato fijo (*rigid scheduling*). Se trata de un problema de alta complejidad computacional, pues incluso sin considerar la reconfiguración, la planificación de tareas es NP-dura [9]. No obstante, incorporar la reconfiguración dinámica, si bien incrementa aún más la dificultad, abre la puerta a mejoras significativas en la calidad de las soluciones. Esto ha sido escasamente explotado hasta la fecha y constituye el núcleo de este trabajo.

En un trabajo previo [2], demostramos el potencial de un enfoque de planificación moldeable para MIG, introduciendo un algoritmo de aproximación

[1]Dpto. de Arquitectura de Computadores y Automática. Universidad Complutense de Madrid, e-mail: {jorvil01,lcostero,figual,katzalin}@ucm.es

que mejoraba el tiempo de finalización frente al uso de MIG estático o a un planificador dinámico del estado del arte. Sin embargo, nuestra solución anterior, al igual que otras [10], se basaba en heurísticas y simplificaciones importantes debido a la complejidad del problema, lo que dejaba margen de mejora. Enfoques basados en métodos exactos, como la programación lineal [11], también suelen simplificar el problema para eliminar su naturaleza NP-dura.

Este trabajo da un paso más allá, abordando el problema sin depender de simplificaciones excesivas o formas de solución predefinidas, utilizando para ello Aprendizaje por Refuerzo Profundo (DRL). El DRL, que combina el Aprendizaje por Refuerzo con redes neuronales, ha demostrado una gran eficacia en problemas de control y gestión de recursos de alta dimensionalidad [12] como el que surge al modelar la planificación de tareas moldeables en GPUs dinámicamente reconfigurables con MIG.

Nuestras principales contribuciones se resumen a continuación:

- Formulación del problema: Definimos formalmente el problema de planificación moldeable con reconfiguración dinámica MIG, incluyendo sus restricciones específicas.
- Planificador DRL para MIG: Proponemos una solución para el problema basada en Aprendizaje por Refuerzo Profundo, evitando asunciones y simplificaciones previas.
- Evaluación exhaustiva: Validamos nuestro planificador DRL en una amplia gama de cargas de trabajo, logrando mejoras en el *makespan* frente al uso estático de MIG, y frente a los planificadores recientes como MISO [7] y FAR [2].
- Código abierto y herramientas: Publicamos el código de entrenamiento del agente, una herramienta de planificación para ejecutar tareas en GPUs MIG siguiendo las decisiones del agente, y una interfaz gráfica para visualizar el estado de la planificación [13].

El resto del artículo se organiza como sigue. La sección II ofrece los conceptos fundamentales de MIG y el trabajo relacionado. La sección III formaliza el problema de planificación. La sección IV especifica el modelado del problema con Aprendizaje por Refuerzo. La sección V detalla la metodología experimental. La sección VI presenta los resultados de evaluación. Finalmente, la sección VII expone las conclusiones.

II. Antecedentes y trabajo relacionado

A. Conceptos y restricciones de la tecnología MIG

NVIDIA incorpora MIG en sus GPUs para centro de datos más recientes con arquitecturas Ampere, Hopper y Blackwell (modelos A30, A100, H100, B100, B200). Una GPU con MIG se puede dividir en varias GPUs virtuales, llamadas *instancias*. Cada instancia está compuesta por una o varias unidades de recursos contiguas físicamente, denominadas *slices* (S_i, $1 \leq i \leq K$), donde K indica el número total de *slices* de la GPU (4 para la A30, 7 para el resto).

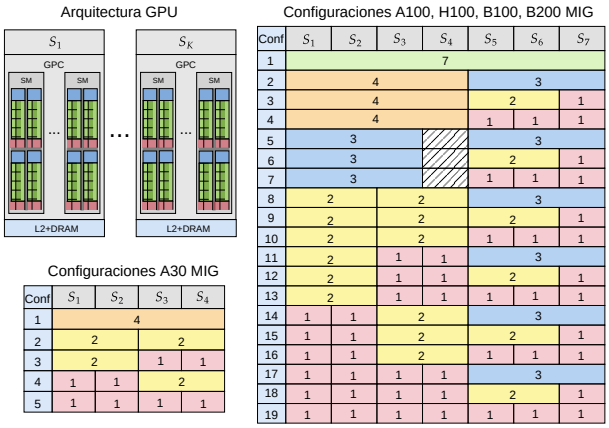


Fig. 1: Configuraciones de *slices* por instancia en GPUs con capacidad de particionado MIG.

Tabla I: Tiempo promedio en segundos para crear/destruir una instancia MIG según su tamaño en diferentes GPUs.

	A30		A100		H100	
Slices	Crear	Destruir	Crear	Destruir	Crear	Destruir
1	0.11	0.10	0.16	0.20	0.16	0.21
2	0.12	0.10	0.17	0.20	0.21	0.23
3	–	–	0.20	0.21	0.33	0.25
4	0.13	0.10	0.21	0.21	0.38	0.26
7	–	–	0.24	0.22	0.42	0.26

Cada *slice* se corresponde con un *GPU Processing Cluster* (GPC), que incluye sus propios *Streaming Multiprocessors* (SMs), una porción de caché L2 y memoria DRAM, y ancho de banda de memoria dedicado. El *tamaño* de una instancia viene determinado por el número de *slices* que la forman. El conjunto de instancias en que se divide la GPU se denomina *configuración* o *partición*.

La Fig. 1 ilustra las configuraciones posibles. No todas las combinaciones de tamaños de instancia son válidas. Por ejemplo, en una GPU de 7 *slices*, no es posible crear una instancia de 5 o 6 *slices*. Además, la configuración que incluye una instancia de 3 *slices* en las posiciones $\{S_1, S_2, S_3\}$ (configuraciones 5, 6 y 7 en la Fig. 1) deshabilita el slice S_4, utilizando su memoria pero dejando sus recursos de cómputo inactivos. Nuestras pruebas indican que estas configuraciones son siempre peores que aquellas que utilizan todos los *slices* de forma productiva (por ejemplo, una instancia de 4 *slices* que incluye S_4), por lo que podremos eliminarlas del espacio de búsqueda de soluciones sin perjuicio.

Las configuraciones se pueden ajustar dinámicamente, destruyendo instancias existentes que deben estar inactivas (no hay mecanismos para expropiar una tarea en ejecución) y creando nuevas instancias en su lugar, sin afectar a las tareas que se ejecutan en otras instancias no involucradas en el cambio. La creación o destrucción de una instancia conlleva un tiempo bastante estable que solo depende de su tamaño (número de *slices*) y se reporta en la tabla I para diferentes GPUs. La desviación de estas latencias de reconfiguración resultó ínfima entre ejecuciones, lo que justifica su uso como datos de entrada constantes para el problema de planificación de tareas que se planteará.

Finalmente, se ha verificado experimentalmente la

efectividad del aislamiento que proporciona MIG, comprobando que las variaciones en los tiempos de ejecución de diversas tareas, al co-ejecutarse con otras en distintas combinaciones, son inferiores al 2 % [2]. Esta elevada predictibilidad, unida a la existencia de predictores de tiempo de ejecución altamente precisos para diferentes tamaños de instancia (error <2 %) [6, 7], valida la posibilidad de emplear estimaciones robustas y fiables del tiempo de ejecución de las tareas en función del tamaño de instancia asignado para la planificación.

B. Trabajo relacionado

Planificación en MIG. Los trabajos previos se basan fundamentalmente en heurísticas [2, 7, 10, 14, 15] o programación lineal [11], simplificando el problema a resolver para reducir su complejidad. Una simplificación habitual es desacoplar la decisión de moldeado de la tarea (el tamaño de instancia) de la planificación final de la co-ejecución y la asignación [2,10,11,14]. Esto se suele hacer asignando a cada tarea el tamaño de instancia que maximiza su eficiencia paralela ($\text{eff}_n = \text{tiempo}_1 / (n \cdot \text{tiempo}_n)$). Sin embargo, esta simplificación no es adecuada en todos los escenarios. Por ejemplo, la tabla II presenta tres tareas cuya eficiencia óptima se produce con instancias de tamaño 2, 4 y 4, pero, como ilustra la Fig. 2, dicha asignación de recursos supone una duración más de un 40 % superior que la asignación de 2 *slices* a todas las tareas. Otra simplificación común consiste en asignar inmediatamente recursos libres a las tareas pendientes, sin considerar la posibilidad de mantenerlos estratégicamente ociosos para reconfigurarlos de una forma más beneficiosa en el futuro.

Particularmente, nuestras líneas base de comparación serán el planificador MISO [7], que utiliza la configuración que maximiza la suma de los *speedups* individuales para las siguientes tareas en orden FIFO, y FAR [2], un algoritmo de aproximación en tres fases que planifica por lotes y refina la solución con heurísticas. Herramientas como Volcano [16] o YuniKorn [17] para Kubernetes utilizan métodos aún más simples y obtienen peores resultados.

Planificación de tareas con RL. El Aprendizaje por Refuerzo (RL) se ha aplicado con éxito a otros problemas de planificación que abarcan desde desafíos clásicos como *Job Shop Schedule* [18] hasta la planificación en diversos clústeres y sistemas heterogéneos [19, 20]. Sin embargo, en esos casos, las tareas no son moldeables (sus requisitos de recursos son fijos), y el sistema no es dinámicamente reconfigurable como las GPUs con capacidad MIG. Frente a estos trabajos, nuestra propuesta maneja aspectos de moldeabilidad y reconfiguración que son clave para tecnologías como MIG.

Reconfiguración dinámica y particionado. En el campo de las FPGAs, la Reconfiguración Parcial Dinámica (DPR) [21] es un concepto similar a MIG, que se ha optimizado con métodos similares a los discutidos [22], con las mismas limitaciones. Por otro lado, la reciente aplicación de DRL en dominios con características afines, como el particionado de caché [23] o la reconfiguración dinámica de redes [24], sugiere la idoneidad de nuestra propuesta.

Tabla II: Duración de la tarea para cada tamaño de instancia (en negrita, para el que se obtiene la eficiencia óptima).

	Tarea 1		Tarea 2 y Tarea 3	
Slices	Tiempo	Eficiencia	Tiempo	Eficiencia
1	25	1	12	1
2	**10**	**1.25**	5	1,2
4	10	0,625	**2**	**1.5**

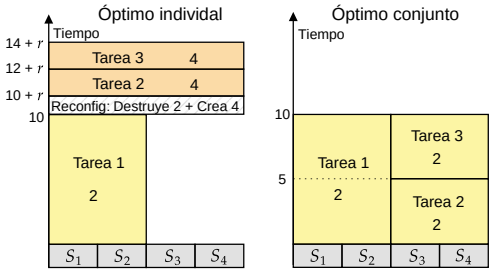


Fig. 2: Planificación de tareas de la Tabla II moldeadas según su eficiencia individual vs. la decisión óptima conjunta.

III. Planteamiento del problema

Nos centramos en optimizar la planificación de un conjunto de tareas pendientes en una GPU con MIG. Tal y como ilustra el esquema de la Fig. 3, el problema se puede abordar de dos formas que se describen a continuación:

- **Enfoque estático multi-lote**: Las tareas se organizan por orden de llegada en lotes de tamaño limitado N. El planificador procesa cada lote, generando un plan completo para todas sus tareas antes de considerar tareas del siguiente lote. Las tareas que llegan durante la planificación o ejecución de un lote se reservan para los siguientes lotes (cola en orden de llegada).
- **Enfoque dinámico**: El planificador gestiona un lote dinámico de tareas. Las nuevas tareas se incorporan al lote en cuanto llegan y son consideradas para las próximas decisiones. Si el planificador necesita una observación de tamaño fijo, se toma el lote formado por las primeras N tareas de la cola. Cuando una tarea pasa del lote a ejecución, es reemplazada por la siguiente en la cola (no se tienen que planificar todas las tareas del lote, sino que este se va modificando dinámicamente conforme hay espacio libre).

El enfoque estático es más simple y evita la inanición de tareas, pero el dinámico puede encontrar mejores patrones de co-ejecución al poder combinar tareas recién llegadas con las que ya estaban pendientes, aunque es más complejo de entrenar debido a su naturaleza estocástica (la transición del lote no solo dependerá de las decisiones de planificación sino de la llegada de nuevas tareas). En este sentido, se evaluarán ambos métodos, ya que DRL puede desarrollar políticas robustas para entornos estocásticos, aunque suele requerir mucho más entrenamiento.

Entrada del problema. El planificador recibe

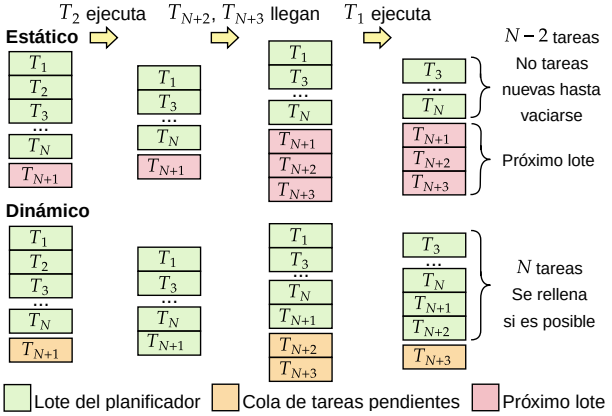


Fig. 3: Evolución de los enfoques estático vs. dinámico con la ejecución o llegada de tareas para un lote limitado a N tareas.

como entrada:

- Para cada tarea T_i, una función $t_i(s)$ que estima su tiempo de ejecución para cada tamaño de instancia $s \in S_G$ (donde S_G es el conjunto de tamaños de instancia posibles para la GPU G).
- Las funciones $t_{\text{crear}}(s)$ y $t_{\text{destruir}}(s)$ con la duración de las operaciones de reconfiguración para un tamaño de instancia s (tiempos de la tabla I).

Salida del problema. El planificador debe decidir la siguiente acción basándose en el lote actual de tareas y el estado de la GPU. Las acciones posibles son: (1) reconfigurar la GPU a una nueva configuración válida, o (2) asignar una tarea del lote a una instancia libre de la configuración actual. En el enfoque dinámico, el planificador también puede tomar decisiones cuando una nueva tarea se une al lote y hay recursos disponibles.

Objetivo. Minimizar el *makespan*, es decir, el tiempo necesario para completar todas las tareas recibidas. El *makespan* es la métrica más frecuentemente usada en la literatura de planificación [25], y está directamente relacionada con una mejor utilización conjunta de los recursos, así como con otras métricas de interés como el consumo energético.

IV. Modelado del entorno

El modelado del entorno para el desarrollo del agente RL se ilustra en el diagrama de alto nivel de la Fig. 4. El entorno comprende dos componentes principales: el estado de la GPU y el lote de tareas pendientes. Ambos se codifican mediante las transformaciones que se detallarán en la siguiente sección, dando lugar a la representación de estado que maneja el agente en cada instante de tiempo: la tupla $(\text{GPU}_t, \text{Lote}_t)$. El agente, implementado como una red neuronal perceptrón multicapa (MLP), procesa este estado y genera una distribución de probabilidad sobre el conjunto de acciones posibles (representadas en la figura por círculos rojos cuya intensidad cromática indica la probabilidad asignada). Sobre esta distribución se aplica una máscara que reescala las probabilidades anulando las acciones no factibles en el estado actual (su uso se motivará en la sección IV-B). Sobre las probabilidades reescaladas se elige aleatoriamente la acción a realizar, la cual modifica el entorno, proporciona una recompensa al agente y genera una nueva observación.

A. Definición de los estados

El estado en el instante t se define formalmente como $(\text{GPU}_t, \text{Batch}_t)$, donde:

- GPU_t representa el estado actual de la GPU. Concretamente codifica la configuración MIG actual y la disponibilidad de recursos de cada uno de sus *slices*. Se modela como:

$$\text{GPU}_t = \left(\text{Conf}_t, S_1^t.\text{time} \ldots, S_K^t.\text{time}\right)$$

  donde Conf_t es el identificador entero de la configuración actual (numeración de la Fig. 1), S_i^t.time es el tiempo restante hasta que el *slice* S_i esté libre (0 si está inactivo). La dimensión de este vector es fija para una arquitectura de GPU dada ($K = 7$ para GPU A100, H100, etc.).

- Batch_t incluye las estimaciones de duración de las tareas a ejecutar en la GPU. Se modela como una matriz de dimensión fija $N \times |S_G|$:

$$\text{Batch}_t = \left(\vec{t_1}, \vec{t_2}, \ldots, \vec{t_N}\right),$$

  donde $\vec{t_i} = \left(t_i\left(1\right), t_i\left(2\right), \ldots, t_i\left(\text{máx}\, S_G\right)\right)$ es el vector de tiempos de ejecución de la i-ésima tarea del lote para cada tamaño de instancia, ordenados de menor a mayor tamaño. Si hay menos de N tareas disponibles, los vectores restantes se rellenan con ceros.

Dada la enorme cantidad de posibilidades en el espacio de estados, fueron necesarias diversas transformaciones y optimizaciones sobre esa representación para que el aprendizaje fuese efectivo. Se describen a continuación las principales:

Discretización de los datos de tiempo: Aunque DRL puede manejar valores continuos, la experiencia muestra que la discretización simplifica y acelera el aprendizaje [10]. Los tiempos de ejecución $t_i(s)$ y los tiempos de reconfiguración $t_{\text{crear/destruir}}(s)$ se discretizaron en M niveles equiespaciados entre 0 y el tiempo máximo de las tareas.

Representación canónica del lote: Un lote de tareas es inherentemente un conjunto, por lo que el orden de las tareas en el vector de entrada no debería influir en la decisión del agente. No obstante, la posición donde aparezcan estos datos en la entrada de la red neuronal con la política del agente influirá en su aprendizaje y decisiones. Para evitarlo, los vectores de las tareas discretizados $\vec{t_i} \in \mathbb{N}_M^{|S_G|}$ se ordenan según su valor numérico en base M, representando así siempre de la misma forma el mismo conjunto (forma canónica). Además, para compactar aún más la representación, se agrupan las tareas por clase (las que tenga exactamente el mismo vector de tiempos), y el estado del lote se representa como una lista de pares $\vec{u_j}, n_j$, donde $\vec{u_j}$ es el vector de tiempos de la clase y n_j el número de tareas de esa clase, manteniendo la ordenación y el

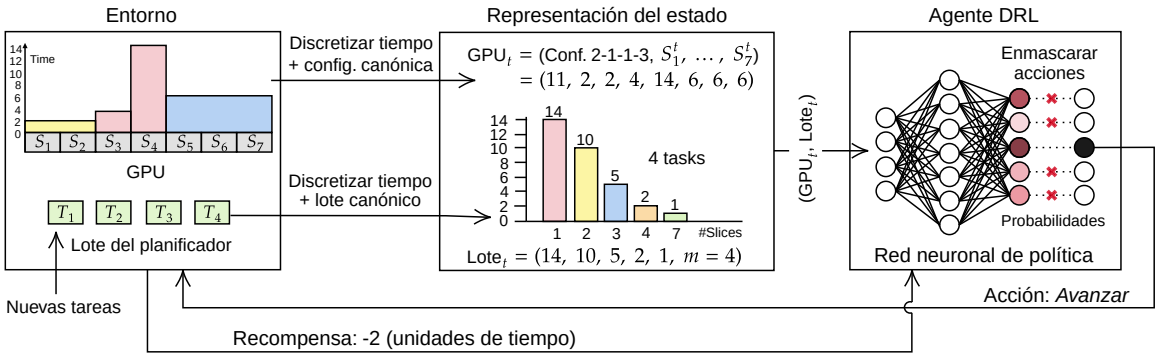


Fig. 4: Diagrama de alto nivel del planificador para MIG basado en DRL.

límite de N tareas totales.

Eliminación de configuraciones inútiles: Como se explicó en la sección II-A, las configuraciones que contienen una instancia de 3 *slices* que deja un *slice* inactivo (configuraciones 5, 6 y 7 de la Fig. 1) son siempre peores en términos de rendimiento que las que utilizan los cuatro *slices* de forma productiva. Por tanto, dichas tres configuraciones se eliminaron del espacio de exploración, reduciendo el número de posibles configuraciones MIG de 19 a 16 para las GPUs de 7 *slices*.

Configuraciones equivalentes: Dos configuraciones pueden representar escenarios de planificación idénticos fruto de simetría sin importancia para la planificación. Por ejemplo, las configuraciones 2-1-1-3 y 1-1-2-3 (válidas en GPUs de 7 *slices*) son equivalentes: existe una permutación de los *slices* que transforma una en la otra. Sin embargo, no toda reordenación de instancias preserva la equivalencia. Consideremos 2-2-1-1-1 y 1-1-2-2-1: aunque una permutación las relaciona, no son equivalentes porque no se mantienen las mismas posibilidades de reconfiguración. Como se puede comprobar en la Fig. 1, en la primera configuración es posible fusionar las dos instancias de tamaño 2 para crear una de tamaño 4 (4-1-1-1), pero al aplicar la permutación inversa a esta nueva configuración se obtiene 1-1-4-1, que no es válida según las reglas de MIG.

Formalmente, dos configuraciones son equivalentes si existe una permutación de *slices* que las relacione y preserve la validez de todas las configuraciones MIG posibles. Esta condición garantiza que cualquier decisión de planificación en una tenga una traducción válida en la otra. Según esta definición, las GPUs de 7 *slices* tienen 3 pares de configuraciones equivalentes ({2-1-1-3, 1-1-2-3}, {2-1-1-2-1, 1-1-2-2-1} y {2-1-1-1-1, 1-1-2-1-1-1}), mientras que las GPUs de 4 *slices* tienen un par ({2-1-1, 1-1-2}).

Para simplificar el aprendizaje, seleccionamos una representación canónica para cada clase de equivalencia: la de mayor valor lexicográfico. Cuando la GPU está en una configuración no canónica, el agente trabaja con la correspondiente versión canónica realizando la permutación σ que permite transicionar a ella (p. ej., la permutación σ para pasar de 1-1-2-3 a 2-1-1-3 es la que intercambia los *slices* S_1 y S_2 por S_3 y S_4). El cambio de base debe tenerse en cuenta

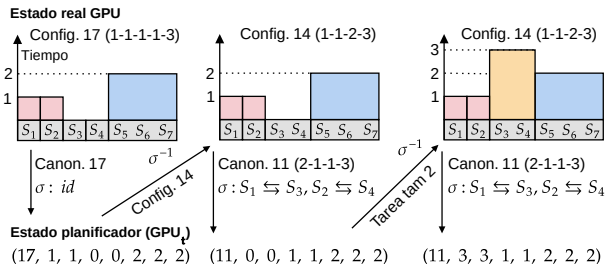


Fig. 5: Conversión entre el estado real de la GPU y la representación canónica del agente cambiando de base con σ y σ^{-1}.

al tomar decisiones sobre el entorno ya que, aunque el agente lo perciba en la configuración canónica, en realidad puede estar en otra configuración de la clase. En este sentido, como se ilustra en la Fig. 5, en cada paso de tiempo se usa σ para traducir la observación de la GPU al estado interno del agente y σ^{-1} para traducir de vuelta la acción del agente al entorno.

B. Definición de las acciones

Las acciones principales que se pueden realizar sobre la GPU operando con MIG son: reconfigurar el dispositivo, o introducir una nueva tarea (del lote) en una instancia libre. Estas acciones solo pueden realizarse sobre instancias libres, de forma que los instantes de decisión para el agente son precisamente aquellos en que esto puede suceder: inmediatamente después de que una tarea termina o una reconfiguración se completa. En este sentido, el agente se modela con una acción más (*Avanzar*) que equivale a dejar pasar el tiempo hasta el siguiente instante de decisión (y particularmente permite dejar *slices* ociosos durante cierto periodo, lo que podría ser beneficioso para fusionarlos en instancias más grandes). Las posibles acciones son:

- **Avanzar:** Adelanta el tiempo de la GPU hasta que la siguiente instancia se libere. Es la única acción que hace progresar el tiempo y permite al modelo mantener *slices* inactivos de forma estratégica para, por ejemplo, acumular suficientes tareas y fusionarlas en una instancia mayor. La precondición para que pueda utilizarse es que la GPU no esté completamente libre.
- **Reconfigurar a C:** Cambia la configuración actual a la configuración objetivo C. La precondición original requiere que todos los *slices* que van a ser modificados (destruidos o creados)

estén libres. En una versión mejorada de nuestro modelo, esta precondición se elimina (como se motivará y explicará en la sección VI-E), permitiendo la reconfiguración directa y ejecutando implícitamente los pasos de *Avanzar* necesarios.

- **Ejecutar la tarea $\tilde{t}_i$ en la instancia I_m:** Asigna una tarea del lote a una instancia libre de la configuración actual. La precondición es que la tarea exista (no sea de relleno) y que la instancia I_m esté completamente libre.

Teniendo en cuenta que a lo sumo hay N tareas diferentes en el lote y K instancias en las que ejecutarlas, el número total de acciones es: $1 + \#\text{Configuraciones} + N \cdot K$. Para una A100 con las reducciones en el número de configuraciones explicadas anteriormente habría $1 + 13 + 14 \cdot 7 = 112$ acciones potenciales para un tamaño de lote $N = 14$ (el que se utilizará finalmente). Sin embargo, muchas de ellas no son viables en todos los estados (deben cumplirse las precondiciones indicadas en cada acción). Por ejemplo, no es posible ejecutar una tarea en una instancia que aún no ha terminado de ejecutar otra tarea ($I_m.\text{time} > 0$) o con un índice superior al número de instancias en la configuración actual (p. ej., ejecutar en I_m con $m > 2$ para una configuración 4-3, aunque sí sea posible en otros estados).

Para manejar esto, se probó el método clásico basado en penalizar las acciones inviables durante el entrenamiento con una recompensa extremadamente negativa. Sin embargo, dado el tamaño del espacio de exploración, tras más de 200 millones de pasos de entrenamiento, nuestros modelos seguían tomando acciones inválidas (el muestreo no es denso en este espacio). Seguidamente, se probó una técnica de la literatura para el enmascaramiento de acciones [26], que garantiza que el agente nunca tomará una decisión imposible (la librería *StableBaselines3* tiene una implementación de los principales algoritmos de entrenamiento consciente de la aplicación de estas máscaras). Como veremos en la Sección VI, con esta técnica sí se obtuvieron resultados satisfactorios en una ventana de entrenamiento asumible.

C. Diseño de la recompensa

El objetivo del agente es maximizar la recompensa acumulada descontada a lo largo de un episodio: $\max_\pi \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t \mid \pi\right]$, donde $\gamma \in (0, 1]$ denota el factor de descuento, r_t la recompensa obtenida en el instante t, π la política del agente y T el último instante de actuación. Para que dicho objetivo fuera equivalente a minimizar el *makespan* de la planificación se empleó la siguiente función de recompensa:

$$r_t = \begin{cases} -sig_tiempo_t & \text{si } a_t = Avanzar, \\ 0 & \text{en otro caso,} \end{cases}$$

donde sig_tiempo_t es el tiempo transcurrido fruto de la acción de *Avanzar* hasta que se libere alguna instancia. El episodio de entrenamiento termina cuando no hay tareas en el lote y todos los *slices* de la GPU están libres ($S_i^T.\text{time} = 0\ \forall i$ y Batch_t está vacío). De esta forma, la suma de las recompensas al final del episodio es exactamente el tiempo transcurrido hasta finalizar la última tarea (*makespan*) con signo negativo (para que el agente lo minimice).

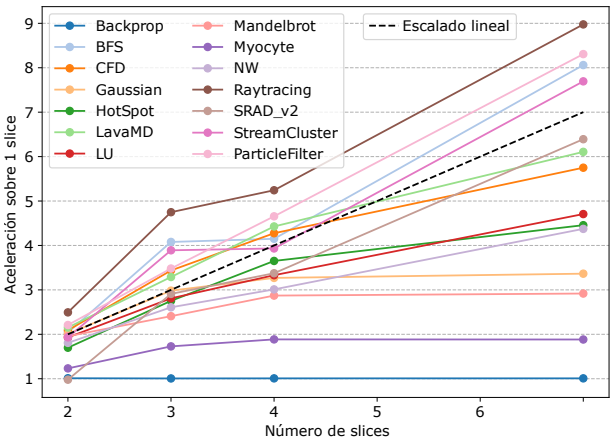


Fig. 6: Escalado de *benchmarks* elegidos con número de *slices*.

V. Metodología experimental

A. Conjuntos de cargas de trabajo

Por una parte, la evaluación se realizará con cargas de trabajo reales, pero el uso de técnicas de aprendizaje automático requiere una enorme cantidad de datos que se asemejen a los reales, por lo que estos se generaron de forma sintética.

***Benchmarks* reales como tareas:** Se seleccionaron 18 *kernels* de las *suites* de *benchmarking* para GPU Rodinia [27] y Altis [28], buscando gran heterogeneidad en los patrones de escalabilidad con MIG (aceleración según el número de *slices* asignados). Dicha aceleración se ilustra en la Fig. 6 para 14 de esos *benchmarks*, observándose: tareas que apenas escalan (como *Backprop*), tareas que escalan hasta un cierto límite (p. ej. *Gaussian* hasta 4 *slices*), que escalan linealmente para todos los tamaños (como *SRAD*) o que incluso superescalan por ser *memory-bound* con pocos *slices* (como *Raytracing*). Se generan 100 subconjuntos de prueba diferentes con estos 18 *kernels* que contienen al menos N tareas y otros 100 con menos de N tareas, para evaluar diferentes escenarios.

Generador sintético de tareas: Inspirado en los comportamientos reales observados para diversas tareas reales, en [2] se presentó un generador parametrizable de duraciones de tarea en MIG según el número de *slices*. Los parámetros clave del generador son: los porcentajes p_s de tareas que escalan bien hasta cada tamaño de instancia s (con speedup casi lineal para $s' \leq s$ y sub-lineal para $s' > s$), y el porcentaje p_{sup} de tareas que exhiben super-linealidad al pasar de 1 a 2 *slices* (común en tareas limitadas por memoria). Los tiempos se generan con cierta aleatoriedad para simular variabilidad dentro de cada configuración de escalado.

Cargas de trabajo sintéticas: Para el entrenamiento y la evaluación, se generaron 1000 conjuntos

de 100 tareas para cada una de las configuraciones del generador que se describen a continuación:

- **PoorScaling:** Tareas que escalan poco al incrementar el número de *slices* de la instancia asignada: $(p_1, p_2, p_3, p_4, p_7) = (50, 50, 0, 0, 0)\,\%$.
- **GoodScaling:** Tareas que escalan bien al incrementar el número de *slices* de la instancia asignada: $(p_1, p_2, p_3, p_4, p_7) = (0, 0, 0, 50, 50)\,\%$.
- **MixScalingUniform:** Capacidades de escalado uniformemente distribuidas: $(p_1, p_2, p_3, p_4, p_7) = (20, 20, 20, 20, 20)\,\%$.
- **MixScalingExtreme:** Capacidades de escalado polarizadas (muy malas junto a muy buenas): $(p_1, p_2, p_3, p_4, p_7) = (45, 5, 0, 5, 45)\,\%$.

B. Métricas de evaluación

Para evaluar el modelo, lo comparamos contra otras soluciones del estado del arte usando una métrica que refleja una cota superior de la distancia a la solución óptima. Para calcular esta métrica se obtiene una cota inferior bastante ajustada del *makespan* óptimo al planificar un lote de n tareas, infraestimando el área hasta la finalización de la última tarea en una representación 2D *slices* × *tiempo* (como la ilustrada en la Fig. 2). Dicha infraestimación se basa en suponer que esta área se reparte uniformemente entre los K *slices* de la GPU. Puesto que dicho área es al menos la suma del área que ocupan las tareas (A_{tareas}) y las reconfiguraciones de la GPU ($A_{\text{reconfig.}}$), se acotan inferiormente estos términos en base al conjunto S de los tamaños de instancia utilizados por las tareas ($L_{\text{tareas}}(S)$ y $L_{\text{reconfig.}}(S)$), minimizando todos los posibles subconjuntos:

$$l_cota = \frac{\min\limits_{S \neq \varnothing \subseteq S_G} \left(L_{\text{tareas}}(S) + L_{\text{reconfig.}}(S)\right)}{K}$$
$$\leq \frac{A_{\text{tareas}} + A_{\text{reconfig.}}}{K} \leq \frac{A_{\text{total}}}{K} \leq \text{opt. } makespan,$$

donde S es el subconjunto de tamaños de instancia usados por las tareas del lote.

Para acotar el área de las tareas, se suma un cota inferior para el área individual de cada una, minimizando, para todos los posibles tamaños de instancia $s \in S$, la suma del área ocupada por la tarea ($s \cdot t_i(s)$) y el área ociosa ($\text{Idle(s, S)} \cdot \text{t}_\text{i}\text{(s)}$):

$$L_{\text{tasks}}(S) = \sum_{i=1}^{n} \min_{s \in S} \left(s \cdot t_i(s) + \text{Idle(s, S)} \cdot \text{t}_\text{i}\text{(s)}\right).$$

Idle(s, S) calcula el número de *slices* que necesariamente deben permanecer ociosos mientras la tarea ejecuta en una instancia de tamaño s, siendo S el subconjunto de los tamaños de instancia utilizados por todas las tareas del lote. Para ello, se evalúan las posibles configuraciones C que incluyan una instancia de tamaño s, sumando los tamaños de instancia s' que estén necesariamente desocupados al no estar en S, y se considera el mínimo de los resultados:

$$\text{Idle(s, S)} = \min_{\substack{C \text{ de MIG} \\ s \in C}} \sum_{\text{s}' \in \text{C, s}' \notin \text{S}} \text{s}'.$$

Finalmente, para calcular una cota inferior del área ocupada por reconfiguraciones, simplemente se suma el área asociada con la creación y destrucción de los diferentes tamaños de instancia utilizados ($s \in S$):

$$L_{\text{reconfig.}}(S) = \sum_{s \in S} s \cdot \left(t_{\text{crear}}(s) + t_{\text{destruir}}(s)\right).$$

La cota inferior del *makespan* que se calcula siguiendo este método se usa como referencia frente a la que comparar el *makespan* producido por el planificador, obteniendo así un porcentaje p_{opt} que sobreestima la distancia al óptimo real:

$$p_{\text{opt}}(\,\%) = \left(\frac{makespan \text{ del planificador}}{l_cota} - 1\right) \cdot 100.$$

Nótese que aunque las estimaciones de tiempo son discretizadas en el estado del modelo, esta métrica se calcula con el tiempo original, de forma que también incluye el error provocado por la cuantización.

Para el enfoque estático de gestión del lote, se calculará la métrica p_{opt} sobre cada uno de los sucesivos lotes empleados y se reportará la media de estos porcentajes. Sin embargo, esto no es adecuado para el enfoque dinámico porque se reemplazan las tareas del lote según el orden en que el modelo las planifique. Lo más justo sería comparar el *makespan* obtenido frente a la menor de todas las cotas inferiores *l_cota* de entre los posibles subconjuntos de tamaño N (tamaño del lote); sin embargo, esto no es factible computacionalmente debido al número exponencial de posibilidades. En su lugar, para evaluar el enfoque dinámico solamente se comparará el *makespan* obtenido con el de otro planificador P (sin tener, por tanto, noción de cuán lejos está del óptimo):

$$p^{P}_{\text{dinámico}} = \left(\frac{makespan \text{ de } P}{makespan \text{ agente dinámico}} - 1\right) \cdot 100.$$

C. Líneas base de referencia

A continuación se listan y describen las propuestas de planificación para MIG contra las que se enfrentaron los modelos RL desarrollados:

- **FAR**: Planificador heurístico *offline* de nuestra autoría [2], que constituye una línea base muy reciente y competitiva.
- **MISO**: Planificador MIG del estado del arte [7], que procesa las tareas en orden de llegada (FIFO) y elige la configuración MIG que maximiza la suma de los *speedups* de las siguientes tareas (estrategia de optimización local, no global).
- **FixBest**: Uso de MIG sin reconfiguración dinámica. Se emplea una configuración fija para ejecutar todas las tareas, pero se considera el escenario ideal donde esa configuración es la que menos *makespan* arroje entre todas las posibles.
- **NoMIG**: Ejecución de las tareas de forma secuencial en la GPU completa, sin particionar el dispositivo ni co-ejecutar las tareas.

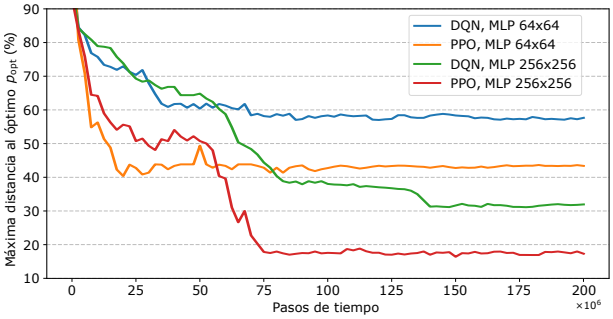


Fig. 7: Valor p_{opt} durante el entrenamiento con diferentes algoritmos y arquitecturas de red.

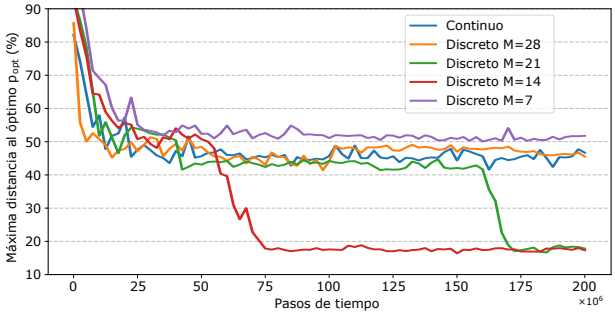


Fig. 8: Valor p_{opt} durante el entrenamiento con diferentes opciones de discretización de los tiempos.

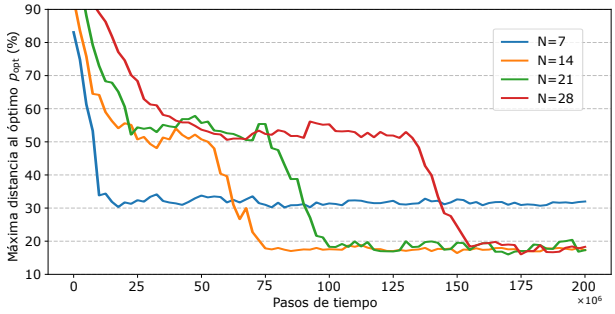


Fig. 9: Valor p_{opt} durante el entrenamiento con diferentes tamaños de lote.

VI. Evaluación y refinamiento

Esta sección presenta la evaluación exhaustiva de nuestra propuesta, donde diversos hiperparámetros requieren un ajuste fino. Para ilustrar el impacto de algunos de ellos, incluimos gráficas que muestran la evolución de la métrica p_{opt} durante el entrenamiento (porcentaje de desviación respecto al óptimo, calculado como se explicó en la sección V-B). Además, nos referiremos a los resultados de la tabla III, que muestran los valores de p_{opt} tras 200 millones de pasos de entrenamiento para diferentes modelos y cargas de trabajo. En esta tabla, las filas corresponden a los tipos de carga, mientras que las columnas comparan las líneas base con las versiones refinadas explicadas a continuación. Dado que la métrica p_{opt} es más adecuada para el enfoque estático, adoptamos este método en los experimentos hasta la sección VI-F, donde evaluamos también el enfoque dinámico.

A. Estrategia de entrenamiento y arquitectura

Para entrenar al agente RL en el entorno descrito en la sección IV, se evaluaron varios *frameworks*, algoritmos y arquitecturas de red. Tras las pruebas iniciales, optamos por StableBaselines3 [29] frente a RLlib [30], al rendir bastante mejor en nuestro escenario. En cuanto a los algoritmos, comparamos *Deep Q-Learning* (DQN) [31] y *Proximal Policy Optimization* (PPO) [32] como principales representantes de los paradigmas basados en política y valor del DRL respectivamente. Como se muestra en la Fig. 7 para cargas GoodScaling, PPO superó a DQN en velocidad de convergencia y recompensa final, por lo que se escogió para las evaluaciones posteriores.

Respecto a la arquitectura de la red, se probaron diversos perceptrones multicapa (MLPs), partiendo del que usa por defecto el *framework* (64×64), sobre el que se añadieron más neuronas por capa o más capas. La Fig. 7 muestra los valores de p_{opt} para varias arquitecturas durante 200 millones de pasos de tiempo. Los mejores resultados se obtienen con la arquitectura 256×256, que fue finalmente seleccionada.

B. Discretización del tiempo

En cuanto a la discretización de las duraciones de las tareas, realizamos varios experimentos comparando el uso de valores de tiempo continuos frente a representaciones discretizadas de diferente granularidad, dividiendo el tiempo máximo representable en M niveles uniformemente espaciados. La Fig. 8 muestra la evolución de p_{opt} durante 200 millones de pasos de entrenamiento para varias de estas opciones. Se observa que, incluso con una cantidad considerable de entrenamiento, el aprendizaje con valores de tiempo continuos no alcanza los resultados de p_{opt} que sí logran las versiones discretizadas con $M = 14$ y $M = 21$ (comenzar con $M = 7$ y modificar en múltiplos de 7 tiene que ver con los 7 *slices* de estas GPUs). Además, $M = 14$ logra valores de p_{opt} similares a $M = 21$, pero requiere considerablemente menos pasos de entrenamiento y tiempo (el entrenamiento es más lento con M mayor). Por lo tanto, seleccionamos $M = 14$ para las evaluaciones subsiguientes. Esta elección se validará al superar a las líneas base, cumpliendo uno de nuestros objetivos principales.

C. Tamaño del lote

De manera similar, buscamos un valor adecuado para el hiperparámetro N que define el tamaño del lote. Dado que una GPU con 7 *slices* puede ejecutar ese número de tareas en paralelo, comenzamos con $N = 7$ y probamos varios múltiplos. La Fig. 9 muestra el p_{opt} obtenido durante el entrenamiento para cargas GoodScaling. Se observa que $N = 7$ converge a un p_{opt} superior al de valores mayores de N, lo que sugiere que un lote de 7 tareas es insuficiente para una buena planificación. Los valores $N \in \{14, 21, 28\}$ convergen a resultados similares, pero $N = 14$ lo hace significativamente más rápido, por lo que fue elegido.

D. Codificación del tiempo discreto

Evaluamos también dos posibles codificaciones para los valores de tiempos discretizados a M niveles. Por una parte una codificación One-Hot para ca-

Tabla III: Promedio de p_{opt} (%) para diferentes cargas de trabajo ($M = N = 14$ y 200 millones de pasos de entrenamiento).

	Líneas base				Versión del agente RL			
Tipo de cargas	FAR	MISO	FixBest	NoMIG	One-Hot	Float	Entropy	Direct Reconfig
Benchs	22.71	26.72	75.37	101.52	89.53	89.36	70.22	21.39
PoorScaling	18.49	19.21	19.21	173.58	15.19	12.64	12.65	12.65
GoodScaling	18.68	20.39	23.38	25.17	17.41	14.68	14.94	14.89
MixScalingUniform	21.95	27.17	94.27	140.11	86.24	85.49	68.47	20.03
MixScalingExtreme	23.09	25.60	92.73	112.35	78.86	78.41	20.95	19.56

da uno de ellos (vector de M componentes con un 1 en el nivel codificado y 0 en todas las demás). La alternativa, que denominamos Float, es utilizar directamente el valor real cuantizado más cercano. Por ejemplo, el intervalo $[0, 1]$ se divide en $M = 4$ con valores reales cuantizados $\{0{,}25, 0{,}5, 0{,}75, 1\}$ y 0,6 se representaría como 0,5, que es el más cercano. Aunque One-Hot puede facilitar el aprendizaje (indica los niveles explícitamente), su mayor dimensionalidad (M entradas por cada tiempo frente a una sola con Float) puede ralentizar el entrenamiento. Al comparar ambas opciones (columnas One-Hot y Float en la tabla III), observamos que Float funciona ligeramente mejor en todos los casos y es más rápida, por lo que la adoptamos.

E. Problema de la falta de reconfiguración

En las versiones One-Hot y Float del agente, se aprecian resultados considerablemente pobres para las cargas Benchs, MixScalingUniform y MixScalingExtreme, con valores de p_{opt} superiores al 80 % (marcados en rojo en la tabla III). La diferencia clave entre estas cargas y aquellas donde el RL funciona bien (PoorScaling y GoodScaling) radica en la diversidad de las capacidades de escalado de las tareas. En escenarios mixtos, la reconfiguración dinámica es crítica. Al analizar ejemplos concretos, descubrimos que el modelo tendía a realizar muy pocas acciones de reconfiguración con curvas de aprendizaje que mostraban una convergencia muy rápida hacia mínimos locales subóptimos. Para solventar este problema, se desarrollaron dos mejoras sobre la propuesta base que se indican a continuación:

Mejora 1: Fomentar la exploración. Se favoreció significativamente la exploración de la política sobre su explotación. En PPO, el coeficiente de entropía juega un papel clave en este sentido. Lo aumentamos de 0 a 0.01, y aplicamos una reducción progresiva con el transcurso del entrenamiento: 0.01 en el primer cuarto de los pasos, 0.0075 en el segundo, 0.005 en el tercero y 0 en el último cuarto. Como se refleja en la columna Entropy de la tabla III, hubo una mejora notable para MixScalingExtreme, pero el rendimiento siguió siendo pobre para el resto de cargas mixtas.

Mejora 2: Reconfiguración directa. Para simplificar la decisión del agente, eliminamos la precondición que requiere que los *slices* a reconfigurar estén libres. Modificamos la acción para que, al seleccionar una reconfiguración, el agente avance implícitamente el tiempo hasta que los recursos necesarios estén disponibles, ejecutando la reconfiguración. De esta forma, es más fácil que el agente muestree este tipo de acciones, infrautilizadas hasta el momento, especialmente durante las primeras etapas del entrenamiento. Como se observa en la columna Direct Reconfig de la tabla III, en los casos donde el agente no reconfiguraba eficazmente, los resultados mejoran sustancialmente, reduciendo el *makespan* obtenido a tan solo un 20 % superior al óptimo (p_{opt}). Estos resultados superan consistentemente a todas las líneas base, con una ligera mejora sobre FAR (1-4 %), una ganancia más notable sobre MISO (5-7 %) y ventajas muy significativas (más de un 100 % en muchos casos) sobre métodos sin reconfiguración dinámica (FixBest) o sin co-ejecución (NoMIG).

Tabla IV: Porcentaje de mejora $p^{P}_{\text{dinámico}}$ (%) frente a otros enfoques (500 millones de pasos de entrenamiento).

	Línea base		Agente RL estático
Tipo de cargas	FAR	MISO	Direct Reconfig
Benchs	2.45	5.80	1.35
PoorScaling	4.21	4.85	-0.92
GoodScaling	2.55	4.02	-0.73
MixScalingUniform	2.78	7.18	1.16
MixScalingExtreme	3.28	5.39	0.32
WideTimes	2.31	7.01	1.25

F. Enfoque dinámico

Finalmente, evaluamos los resultados del enfoque dinámico, cuyas ventajas y desventajas se discutieron en la Sección III. La tabla IV muestra el porcentaje de mejora $p_{\text{dinámico}}$ de la versión refinada Direct Reconfig del agente dinámico, comparado con las líneas base y con la misma versión del agente estático. Este agente se entrenó durante más pasos (500 millones) debido a su naturaleza estocástica, que ralentiza el aprendizaje pero aporta mayor variedad de opciones. Para una comparación justa, el agente estático también se entrenó durante 500 millones de pasos, sin ganancia apreciable respecto a los 200 millones previamente empleados.

La tabla IV muestra que el enfoque dinámico mejora ligeramente el *makespan* del método estático (aproximadamente un 1 %) para cargas variadas o con duraciones heterogéneas (Benchs, MixScalingUniform, etc.), que es precisamente donde el estático está más lejos del óptimo. Para cargas más homogéneas, el dinámico fue ligeramente peor (alrededor de un 1 %). En conjunto, las mejoras frente a las líneas base oscilan entre el 2 % y el 7 %, con una latencia media de inferencia de apenas 3.04 ms (percentil 99 de 3.09 ms) en un servidor con CPU Intel Xeon Silver 4314 (empleando el modelo 256×256

con $N = M = 14$). De esta forma, no solo se obtienen mejores resultados que con los enfoques previos, sino que el rendimiento es suficiente para emplear la propuesta en un entorno *online* real.

VII. Conclusiones

En este trabajo presentamos un planificador basado en Aprendizaje por Refuerzo Profundo (DRL) para la gestión de tareas moldeables en GPUs con particionado MIG dinámico. A diferencia de los enfoques previos, que dependen de heurísticas y simplificaciones que desacoplan las decisiones de planificación, el enfoque de nuestro agente es holístico, sin asumir ninguna forma de solución prefijada.

Mediante un conjunto de optimizaciones clave, como la discretización y representación canónica del estado, el enmascaramiento de acciones inválidas o el fomento de la exploración en el entrenamiento, hemos conseguido que el agente converja de manera efectiva en un espacio de soluciones extremadamente grande. Los resultados experimentales, obtenidos tanto con *benchmarks* reales (procedentes de Rodinia y Altis) como con cargas sintéticas representativas, demuestran que el modelo final (Direct Reconfig) mejora consistentemente al estado del arte: supera a FAR en un 2-4 %, a MISO en un 5-7 % y a las versiones estáticas (FixBest y NoMIG) en más del 100 % en muchos escenarios, con un exceso medio sobre el óptimo inferior al 20 %.

Además de los resultados, ponemos a disposición de la comunidad el código de entrenamiento, la herramienta de planificación y ejecución en GPUs MIG reales, y un visualizador gráfico del estado del agente [13]. Este trabajo no solo resuelve de forma efectiva el problema de planificación en MIG, sino que ofrece lecciones prácticas y transferibles para el diseño de soluciones DRL en otros entornos similares, con recursos dinámicamente reconfigurables o moldeables.

Agradecimientos

Este trabajo ha sido financiado por PID2024-158311NB-I00 (MCIN/AEI/10.13039501100011033), PCI2024-162578-3 (MICIU/AEI/10.13039/501100011033 /) y por la Unión Europea NextGenerationEU/PRTR y ERDF A way of making Europe.

Referencias

[1] Theodora Adufu et al., "Exploring the Diversity of Multiple Job Deployments over GPUs for Efficient Resource Sharing," in *International Conference on Information Networking*. 2024, pp. 777–782, IEEE Computer Society.

[2] Jorge Villarrubia et al., "Leveraging Multi-Instance GPUs through moldable task scheduling," *Journal of Parallel and Distributed Computing*, vol. 204, pp. 105128, 2025.

[3] Connor Espenshade et al., "Characterizing Training Performance and Energy for Foundation Models and Image Classifiers on Multi-Instance GPUs," in *4th Workshop on Machine Learning and Systems*, 4 2024, pp. 47–55.

[4] NVIDIA, "Multi-Process Service," `https://docs.nvidia.com/deploy/mps/`.

[5] NVIDIA, "Multi-Instance GPU User Guide," `https://docs.nvidia.com/datacenter/tesla/mig-user-guide/`.

[6] Bowen Zhang et al., "MIGER: Integrating multi-instance gpu and multi-process service for deep learning clusters," in *53rd International Conference on Parallel Processing*, 2024, p. 504–513.

[7] Baolin Li et al., "MISO: Exploiting Multi-Instance GPU Capability on Multi-Tenant GPU Clusters," in *13th Symposium on Cloud Computing*, 2022.

[8] AMD, "MxGPU Deployment Guide," `https://drivers.amd.com/relnotes/amd_mxgpu_deploymentguide_vmware.pdf`.

[9] Jianzhong Du et al., "Complexity of Scheduling Parallel Task Systems," *SIAM Journal on Discrete Mathematics*, vol. 2, no. 4, pp. 473–487, 1989.

[10] Jiaxing Qi et al., "ElasticBatch: A Learning-Augmented Elastic Scheduling System for Batch Inference on MIG," *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[11] Bekir Turkkan et al., "Optimal workload placement on multi-instance gpus," 2024.

[12] Yan Gu et al., "Deep reinforcement learning for job scheduling and resource management in cloud computing: An algorithm-level review," 2025.

[13] Jorge Villarrubia, "Repositorio de código: RL MIG scheduler," `https://github.com/artecs-group/RL_MIG_scheduler`, 2025.

[14] Seungbeom Choi et al., "Serving Heterogeneous Machine Learning Models on Multi-GPU Servers with Spatio-Temporal Sharing," in *USENIX ATC*, 2022.

[15] Xinpeng Wei et al., "Optimizing GPU Sharing for Container-Based DNN Serving with Multi-Instance GPUs," in *17th ACM International Systems and Storage Conference*, 9 2024, pp. 68–82.

[16] Volcano Project, "Volcano documentation of dynamic-MIG feature," `https://github.com/volcano-sh/volcano/blob/master/docs/design/dynamic-mig.md`.

[17] Software Foundation Apache, "Yunikorn user guide for NVIDIA GPUs," `https://yunikorn.apache.org/docs/user_guide/workloads/run_nvidia/`.

[18] Dongping Qiao et al., "Optimization of job shop scheduling problem based on deep reinforcement learning," *Evolutionary Intelligence*, vol. 17, 2024.

[19] Jaime Fomperosa et al., "Task scheduler for heterogeneous data centres based on deep reinforcement learning," in *Parallel Processing and Applied Mathematics*, 2023, pp. 237–248.

[20] Boyang Li et al., "MRSch: Multi-Resource Scheduling for HPC," in *IEEE International Conference on Cluster Computing*, 2022.

[21] Wang Lie et al., "Dynamic partial reconfiguration in fpgas," in *Third International Symposium on Intelligent Information Technology Application*, 2009, vol. 2, pp. 445–448.

[22] Ashutosh Dhar et al., "DML: Dynamic Partial Reconfiguration with Scalable Task Scheduling for Multi-Applications on FPGAs," *IEEE Transactions on Computers*, vol. 71, 2022.

[23] Ayoub Ben-Ameur et al., "Cache allocation in multi-tenant edge computing: an online model-based reinforcement learning approach," *IEEE Transactions on Cloud Computing*, pp. 1–14, 2025.

[24] Ruiqi Si et al., "A multi-agent reinforcement learning method for distribution system restoration considering dynamic network reconfiguration," *Applied Energy*, vol. 372, pp. 123625, 2024.

[25] Maziyar Khadivi et al., "Deep reinforcement learning for machine scheduling: Methodology, the state-of-the-art, and future directions," *Computers & Industrial Engineering*, vol. 200, pp. 110856, 2025.

[26] Shengyi Huang et al., "A closer look at invalid action masking in policy gradient algorithms," *The Intl. FLAIRS Conference Proceedings*, vol. 35, 2022.

[27] Shuai Che et al., "Rodinia: A benchmark suite for heterogeneous computing," in *IEEE International Symposium on Workload Characterization*, 2009, pp. 44–54.

[28] Bodun Hu et al., "Altis: Modernizing gpgpu benchmarks," in *IEEE International Symposium on Performance Analysis of Systems and Software*, 2020, pp. 1–11.

[29] Antonin Raffin et al., "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[30] Eric Liang et al., "RLlib: Abstractions for distributed reinforcement learning," in *International Conference on Machine Learning*, 2018, vol. 80, pp. 3053–3062.

[31] Volodymyr Mnih et al., "Playing atari with deep reinforcement learning," 2013.

[32] John Schulman et al., "Proximal policy optimization algorithms," 2017.

Generación de Código RISC-V Optimizado con MLIR y xDSL

Jie Lei[1], Adrián Castelló[1]y Héctor Martínez[2]

Resumen— **La creciente adopción de dispositivos RISC-V en la computación de altas prestaciones y la amplia variedad de sus características han puesto de relieve la necesidad de generar código portable y óptimo para la extensión RISC-V Vector (RVV). Sin embargo, las distribuciones actuales de MLIR carecen de soporte para mapear las abstracciones de alto nivel a las instrucciones vectoriales RVV, lo que limita su capacidad para generar núcleos de RISC-V escritos en C. Este artículo presenta un enfoque de compilación que combina MLIR con xDSL para completar las etapas necesarias para la generación de código óptimo en RVV. Utilizando representaciones intermedias personalizadas y transformaciones implementadas en xDSL, transformamos sistemáticamente operaciones definidas en alto nivel en código C especializado y consciente del hardware que incluye instrucciones RVV. Los códigos resultantes se emiten como funciones C portables que pueden integrarse directamente en aplicaciones existentes, lo que permite su adopción sin modificar el software circundante. Al integrar el código generado en el micro-kernel de la multiplicación de matrices, el enfoque propuesto supera consistentemente a OpenBLAS, alcanzando hasta 12,2 GFLOPS en comparación con los 5,1 GFLOPS del modelo base y proporcionando mejoras de rendimiento de entre un 10 y un 35 % en las cargas de trabajo evaluadas.**

Palabras clave— **Extensión RISC-V Vector, Infraestructuras de compiladores, Generación de código, MLIR, xDSL, HPC, ML.**

I. Introducción

La creciente adopción de RISC-V como ISA (del inglés *Instruction Set Architecture*) común en dominios que van desde dispositivos del IoT (del inglés *Internet of Things*) hasta sistemas de HPC (del inglés *High Performance Computing*) ha introducido nuevos desafíos para la portabilidad y la optimización del software. En el contexto de HPC, los procesadores RISC-V modernos implementan cada vez más la extensión RISC-V Vector (RVV) en diversas configuraciones, incluyendo unidades de SIMD de anchos variables y arquitecturas de registros vectoriales de gran tamaño. Aunque estas plataformas comparten una ISA común, su diversidad microarquitectónica dificulta el desarrollo de software portátil y de alto rendimiento.

Lograr un alto rendimiento en RVV requiere una gestión explícita de la programación agnóstica de la longitud del vector, la selección de intrínsecos y el dimensionamiento de los bloques. A medida que el número y la heterogeneidad de las implementaciones de RISC-V continúan creciendo, mantener códigos optimizados a mano se vuelve cada vez más impracticable, incluso para cargas de trabajo de HPC habituales, como el álgebra lineal densa.

Las infraestructuras de compiladores como Halide [1], MLIR [2], [3], Apache TVM [4] o Exo [5] abordan este desafío generando código optimizado y consciente del hardware a partir de representaciones de alto nivel. Entre ellas, MLIR proporciona una potente IR (del inglés *Intermediate Representation*) de múltiples niveles y un marco de trabajo que permite personalizar las transformaciones. Sin embargo, las distribuciones actuales de MLIR carecen de flujos completos de reducción para objetivos RVV, en particular para mapear abstracciones vectoriales de alto nivel a instrucciones RVV y a código C consciente del hardware. Esta situación limita la aplicabilidad de MLIR para la generación de códigos optimizados para RISC-V.

En este trabajo, cerramos esta brecha combinando MLIR con xDSL [6], [7], un conjunto de herramientas nativas de Python para diseñar compiladores específicos de dominio, lo que nos permite generar sistemáticamente código C basado en instrucciones RVV. Utilizando xDSL, implementamos las etapas de transformación faltantes en MLIR e introducimos IRs que modelan explícitamente la semántica de RVV y las transformaciones conscientes de la longitud del vector. Los códigos resultantes se emiten como funciones C independientes que pueden integrarse directamente en aplicaciones existentes.

Aprovechando la generación automática de código, también nos beneficiamos de la creación de una colección completa de micro-kernels de distintos tamaños para la GEMM (del inglés *General Matrix Multiplication*). Las combinaciones generadas se utilizan posteriormente para elegir el mejor micro-kernel para cada escenario y, en particular, para los modelos de DL (del inglés *deep learning*), el mejor tamaño de micro-kernel para cada capa y arquitectura objetivo.

Concretamente, este artículo realiza las siguientes contribuciones:

- Analizamos las limitaciones de MLIR para la generación de código RVV e identificamos las etapas de reducción faltantes.
- Diseñamos e implementamos un flujo de compilación híbrido MLIR–xDSL que cierra estas brechas mediante dialectos y transformaciones personalizados.
- Habilitamos la generación de micro-kernels de C y RVV especializados y conscientes del hardware para GEMM.
- Evaluamos el código generado y lo comparamos con la implementación altamente optimizada de

[1]Universitat Politècnica de València, València, España, e-mail: {jlei,adcastel}@disca.upv.es.
[2]Universidad de Córdoba, Córdoba, España, e-mail: el2mapeh@uco.es.

```
for (jc=0; jc<n; jc+=nc)           // L1 loop
 for (pc=0; pc<k; pc+=kc) {        // L2
  Bc := B(pc:pc+kc-1,jc:jc+nc-1);  // Pack B
  for (ic=0; ic<m; ic+=mc) {       // L3
   Ac := A(ic:ic+mc-1,pc:pc+kc-1); // Pack A
   for (jr=0; jr<nc; jr+=nr)       // L4
    for (ir=0; ir<mc; ir+=mr)      // L5
     // Loop L6
     C(ic+ir:ic+ir+mr-1, jc+jr:jc+jr+nr-1)
        += Ac(ir:ir+mr-1,pr) *
           Bc(pr,jr:jr+nr-1);
  }}
```

Fig. 1: Pseudo-código del algoritmo GEMM de BLIS basado en Goto.

GEMM en OpenBLAS utilizando diferentes modelos de DL.

El resto del artículo se organiza de la siguiente manera. La Sección II resume un algoritmo de alto rendimiento para GEMM e introduce los marcos de trabajo MLIR y xDSL; la Sección III enumera otros enfoques para la generación automática de código y revisa algunos trabajos existentes; la Sección IV presenta el flujo de trabajo completo de nuestra solución propuesta; la Sección V evalúa y compara el código C autogenerado en operaciones GEMM; la Sección VI cierra el artículo con algunas observaciones finales.

II. Contexto

A. GEMM *en bibliotecas de alto rendimiento*

Considere la GEMM $C \leftarrow C + AB$, donde $A \in \mathbb{R}^{m \times k}$, $B \in \mathbb{R}^{k \times n}$, y $C \in \mathbb{R}^{m \times n}$. Las implementaciones de alto rendimiento de GEMM en las bibliotecas BLAS modernas siguen la estructura algorítmica popularizada por GotoBLAS [8], con la computación organizada como una jerarquía de memoria de bucles y rutinas de empaquetamiento diseñadas para explotar la jerarquía de memoria y las unidades SIMD, tal como se muestra en la Figura 1.

En particular, en el nivel más externo, tres bucles anidados iteran sobre las dimensiones n, k y m utilizando parámetros de bloque conscientes de la caché n_c, k_c y m_c. Estos bucles dividen las matrices en paneles que encajan en distintos niveles de caché. Las submatrices de A y B se reorganizan y empaquetan en búferes contiguos ($A_c \in \mathbb{R}^{m_c \times k_c}$ y $B_c \in \mathbb{R}^{k_c \times n_c}$) para garantizar un comportamiento predecible de la caché.

La aritmética real y crítica se realiza dentro del micro-kernel, un componente que normalmente se escribe en ensamblador o en C utilizando instrucciones vectoriales. El micro-kernel actualiza un pequeño micro-tile de la matriz C de tamaño $m_r \times n_r$ procesando el resultado a través de la dimensión k_c, realizando efectivamente una secuencia de operaciones de producto punto a punto. Los micro-paneles empaquetados $A_r \in \mathbb{R}^{m_r \times k_c}$ y $B_r \in \mathbb{R}^{k_c \times n_r}$, que forman parte de A_c y B_c, se organizan para maximizar la reutilización de registros, garantizar el acceso a la memoria de forma contigua y mejorar la eficiencia SIMD.

Adaptar esta estructura a RISC-V con RVV requiere un manejo cuidadoso de la ejecución agnóstica de la longitud del vector, la selección de instrucciones vectoriales y los tamaños de bloque, lo que motiva los enfoques sistemáticos de generación de código.

B. MLIR y xDSL

MLIR es una infraestructura de compiladores desarrollada dentro del ecosistema de LLVM para abordar la creciente complejidad de las pilas de software modernas, en particular en dominios como HPC y ML. A diferencia de las IRs de compiladores tradicionales que operan a un único nivel de abstracción, MLIR admite múltiples niveles de abstracción dentro de un marco unificado. Esto se logra a través de un sistema flexible de dialectos, cada uno de los cuales puede modelar operaciones, tipos y transformaciones específicos del dominio. Este diseño permite una progresión gradual hacia código orientado al hardware de bajo nivel mientras se preserva la estructura semántica a lo largo de todo el flujo de compilación.

Una fortaleza clave de MLIR reside en su extensibilidad y composibilidad. Los desarrolladores pueden definir dialectos personalizados, introducir nuevos pases de transformación e interoperar con los backends de LLVM existentes. MLIR también proporciona la infraestructura para la reescritura de patrones, la canonización y el análisis, lo que facilita la optimización sistemática y la generación de código. En el contexto de ML y el álgebra lineal, MLIR permite representaciones estructuradas de operaciones de tensores, referencias de memoria y expresiones aritméticas, que pueden reducirse incrementalmente a dialectos específicos del objetivo o a marcos de emisión de código externos.

xDSL es un kit de herramientas de compilación ligero y nativo de Python diseñado para reflejar y complementar la filosofía de MLIR, enfatizando el prototipado rápido y la accesibilidad. Proporciona un entorno ligero para definir dialectos, construir IRs e implementar fases de reducción sin la sobrecarga de ingeniería de las grandes infraestructuras basadas en C++.

Una ventaja principal de xDSL es su flexibilidad para la experimentación y la automatización. Su base en Python permite una integración fluida con la creación de scripts, marcos de trabajo de prueba y herramientas externas, lo que la hace particularmente adecuada para el desarrollo de compiladores impulsado por la investigación. Con xDSL, los desarrolladores pueden prototipar rápidamente nuevas transformaciones, generar IR e iterar sobre estrategias de reducción con un mínimo de código repetitivo. Como resultado, xDSL sirve como un entorno eficaz para explorar flujos de compilación novedosos y cerrar brechas en las infraestructuras existentes.

III. Trabajo Relacionado

La rápida proliferación de plataformas de cómputo ha creado un desafío significativo para la portabilidad de rendimiento: lograr una ejecución eficiente en arquitecturas con anchos de vector, jerarquías de memoria, conjuntos de instrucciones y características

arquitectónicas fundamentalmente diferentes.

En este contexto, la generación automática de código optimizado se ha convertido en un tema central de investigación. Por lo tanto, en esta sección revisamos brevemente algunos ejemplos aplicados a la generación de GEMM y micro-núcleos. Los autores de [9], [10] utilizan TVM para optimizar la operación completa de GEMM en diferentes arquitecturas de hardware, y los autores de [11] emplean MLIR para describir las primeras experiencias también con el algoritmo GEMM completo.

En el caso de los micro-kernels optimizados, el trabajo en [12] se centra en generar soluciones para Arm NEON. Además, el trabajo en [13] utiliza el marco de trabajo Exo para implementar generadores de código especializados para CPUs RISC-V.

Desde la perspectiva del uso de xDSL para la generación de código, nuestro trabajo se inspira en [14], que también utiliza xDSL y MLIR para la extensión RISC-V. Su enfoque evita explícitamente los compiladores de propósito general, optando en su lugar por construir un backend autónomo optimizado para su microarquitectura interna, concretamente el acelerador Snitch, para generar código ensamblador nativo altamente eficiente.

En contraste, nuestro trabajo prioriza la amplia portabilidad y el desarrollo ágil en el fragmentado ecosistema RISC-V, dirigiéndose específicamente a las instrucciones RVV. Al emitir código C estándar, nuestros micro-kernels de tamaño dinámico actúan como un ensamblador universal. Esto garantiza una mayor aplicabilidad y una integración fluida en las cadenas de herramientas existentes, sin bloquear al usuario en un único backend de hardware personalizado.

IV. Construcción del generador de código

Este trabajo tiene como objetivo aprovechar MLIR para facilitar la generación automática de código para diseños de GEMM de alto rendimiento en CPUs RISC-V. Por lo tanto, el objetivo final es proponer un flujo de diseño que permita a los usuarios especificar las dimensiones del micro-kernel ($m_r \times n_r$), el tipo de datos y la longitud de los registros vectoriales utilizando dialectos MLIR de alto nivel.

El flujo de trabajo está diseñado para generar micro-kernels de código C portátil enriquecidos con instrucciones vectoriales RVV 1.0. El código C estándar fue elegido estratégicamente como el formato de salida final, proporcionando una solución altamente compatible. Gracias a su portabilidad, puede compilarse y ejecutarse en diversas plataformas RISC-V.

A. El algoritmo de reducción de MLIR incompleto

MLIR facilita la generación de código C a través de su dialecto `emitc`, creado para cerrar la brecha entre las abstracciones de LLVM y los sistemas novedosos o empotrados donde el acceso a la infraestructura completa de LLVM es limitado. Una vez que el código está completamente escrito en `emitc`, la herramienta `mlir-translate` lo convierte directamente en código C.

En MLIR, los usuarios pueden formular sus diseños utilizando abstracciones de alto nivel en una variedad de dialectos. Por ejemplo, pueden expresar la multiplicación de coma flotante utilizando el dialecto `arith` o la asignación dinámica de memoria utilizando el dialecto `memref`. Ambas operaciones son críticas para una operación GEMM de alto rendimiento.

Por lo tanto, comenzamos utilizando MLIR (LLVM 22.0) para realizar la reducción de estas funciones de alto nivel a `emitc`. Si bien esta versión completó con éxito la reducción del dialecto `arith`, no pudo manejar el de `memref`. Este problema ocurre porque el pase de reducción de `memref` a `emitc` para tipos de memoria dinámicos está incompleto en la versión upstream, lo que impide la generación de punteros C estándar. También comprobamos que MLIR era capaz de realizar la reducción de código escrito en `emitc` puro a C. Sin embargo, escribir en `emitc` es cercano a hacerlo en ensamblador, algo que este trabajo busca evitar.

B. Flujo propuesto de generación automática de código híbrido MLIR–xDSL

Al intentar implementar nuestros pases de reducción personalizados directamente dentro del proyecto oficial de MLIR, encontramos un alto coste de desarrollo y la rigidez de la infraestructura de MLIR, que presenta una curva de aprendizaje pronunciada. Centrándonos en nuestro objetivo principal, que busca desarrollar un proceso de reducción que convierta los dialectos MLIR de alto nivel en `emitc` para generar código C a través de la herramienta `mlir-translate`, concluimos que necesitábamos un marco alternativo para simplificar el diseño de la reducción.

Para resolver esto, mezclamos xDSL y MLIR en el flujo híbrido mostrado en la Figura 2. Utilizamos xDSL para generar micro-kernels con una configuración especificada por el usuario en dialectos de MLIR. xDSL emplea su propia API para generar un diseño de IR mezclado con varios dialectos de MLIR. Una vez generada la IR, pasa por nuestro algoritmo de reducción para generar el código `emitc`. Luego, regresamos a la herramienta oficial `mlir-translate` para completar el flujo de traducción a C.

Dentro de este entorno unificado, aprovechamos Python para generar automáticamente los encabezados de proyecto C necesarios y los programas de prueba principales, desplegar los archivos fuente en una placa RISC-V, compilarlos de forma nativa y evaluar el rendimiento de cada código generado. Esta integración fluida garantiza que el ciclo de desarrollo de extremo a extremo, desde la especificación de alto nivel del micro-núcleo hasta la generación de código y la evaluación comparativa, pueda ejecutarse en cuestión de minutos.

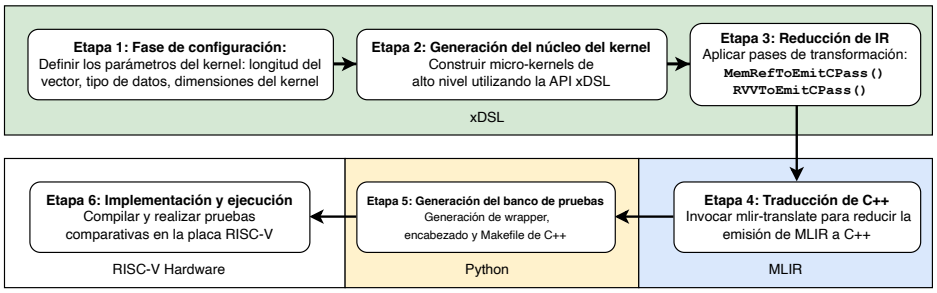


Fig. 2: Representación del flujo de generación de código híbrido MLIR-xDSL.

C. *El flujo del generador de micro-núcleos de extremo a extremo*

Para ilustrar las capacidades de nuestro flujo de generación de código híbrido MLIR-xDSL, seguimos la creación de un micro-kernel RISC-V Vector (RVV) de 8×4. Nos centramos específicamente en el componente más crítico: el bucle L6 del micro-kernel (Figura 1), que realiza las operaciones FMA.

Etapa 1: Configuración del micro-kernel (xDSL). El flujo está diseñado para ser flexible en la construcción de código, tanto para tipos de datos escalares como para vectoriales. El proceso comienza en el entorno xDSL basado en Python, donde el usuario especifica las dimensiones del micro-kernel (p. ej., $m_r = 8, n_r = 4$), el tipo de datos (p. ej., FP32) y la longitud del vector objetivo expresada en bits (p. ej., `vlen=256`). Nuestras abstracciones xDSL personalizadas construyen dinámicamente la representación de MLIR de alto nivel para todos los micro-kernels, en todas las combinaciones posibles de dimensiones, desde 1×1 hasta $m_r \times n_r$. Este conjunto de micro-kernels generado automáticamente cubre todos los casos críticos posibles en la operación GEMM.

Etapa 2: Generación del cuerpo del micro-kernel (xDSL). Nuestro `IRBuilder` de Python dirige la construcción del micro-kernel, incluyendo la carga del micro-tile de la matriz C en los registros C_r, el bucle L6 y el almacenamiento de resultados de los registros C_r en la matriz C. El bucle L6 incluye la carga de datos desde el búfer A_c a los registros vectoriales A_r y n_r operaciones `AXPI`. Nótese que el micro-kernel carga los valores del búfer B_c directamente en lugar de usar registros B_r debido a la pérdida de rendimiento señalada en [13]. El código para la generación del bucle L6 se muestra en la Figura 3.

La línea 3 emite la carga vectorial de los valores de A_c en los registros vectoriales A_r. En el caso del micro-núcleo de 8×4, solo se usa un registro porque los 8 elementos FP32 caben dentro de un único registro de 256 bits; las líneas 8–14 insertan n_r (4 en este caso) instrucciones FMA utilizando la abstracción `lb.vfmacc` que forma parte de nuestro dialecto `rvv` propuesto; las líneas 19–21 añaden la lógica para reutilizar los acumuladores de los registros C_r (`scf.YieldOp`); y las líneas 25–26 construyen el bucle for L6.

El código de la Figura 3 produce la IR de MLIR de

```
1  # 1. Emitir la instruction vle32.v
2  # para cargar el vector Ar de los datos de Ac
3  Ar.append(lb.vle32(Ac, k_mul_lda, vl))
4
5  # 2. Emitir nr instrucciones vfmacc para
6  # multiplicar el escalar de Bc por el vector
7  # Ar, acumulando en Cr
8  for i in range(nr):
9    c_new[(i, mr)] = lb.vfmacc(
10     Cr[(i, mr)],   # Acumulador SSA entrante
11     Bc, b_offset,  # Bc memory scalar
12     Ar[mr],        # Vector A cargado
13     vl             # Longitud del vector
14   )
15
16 # 3. Entrega de los nuevos registros
17 # vectoriales generados a la siguiente
18 # iteracion
19 yield_vals =[c_new[(i,j)] for n in range(nr)
20                           for j in range(mr)]
21 loop_body.add_op(scf.YieldOp(*yield_vals))
22
23 # 4. Construir la representacion del bucle
24 # scf.for L6
25 for_op = scf.ForOp(start, stop, step,
26              iter_inits, Region([loop_body]))
```

Fig. 3: Representación en xDSL (Python) del bucle L6 para el micro-núcleo de 8×4.

```
1  %37,...,%40 =scf.for %41 = %10 to %0 step %11
2              iter_args(%42 = %19,...) -> (...)
3  {
4    ...
5    // Emitir FMA para Columna 0
6    // (tomando el acumulador entrante %42)
7    %61 = "rvv.vfmacc_vf_f32m10p"(%42, %4, %53,
8                                  %52, %51) :
9    (!rvv.vfloat32m1, memref<-1xf32>, index,
10    !rvv.vfloat32m1, index) -> !rvv.vfloat32m1
11   ...
12   // Entrega SSA explicita de nuevos
13   // acumuladores (%61) a la siguiente
14   //iteracion del loop
15   scf.yield %61, %62, %63, %64
16 }
```

Fig. 4: IR de MLIR con la representación `scf.for` del L6 loop y enlaces FMA.

la Figura 4, donde: las líneas 7–10 muestran la primera operación FMA de RVV del bucle L6. Omitimos las otras 3 instrucciones de FMA por brevedad; la línea 15 gestiona la reutilización de los registros C_r; y las líneas 1–16 conforman el cuerpo del bucle L6 (`scf.for`). El método de nomenclatura de variables utilizado en el código MLIR emplea nombres secuenciales como `%33` o `v68` porque la herramienta MLIR mapea determinísticamente la numeración de registros subyacente de MLIR.

Para entender cómo el flujo de xDSL orquesta la generación de FMA, analicemos los 5 argumentos pasados a la instrucción `rvv.vfmacc_vf_f32m10p` de MLIR generada: `%42 (!rvv.vfloat32m1)` es el

```
// 1. Generacion inicial xDSL/MLIR
// (Algebra vectorial abstracta)
func.func @xdsl_api_kernel_(%0: index,
    %1: memref<-1xf32>,
    %2: memref<-1xf32>, ...) { ... }

// 2. IR de EmitC transformada
// (Punteros de memoria)
"func.func"()<{sym_name = "xdsl_api_kernel_",
    function_type = (index, !emitc.ptr<f32>,
    !emitc.ptr<f32>, ...) -> ()}>

// 3. Emision final de C++ ejecutable
void xdsl_api_kernel_(void*, int v1,
    float* v2, float* v3, ...) { ... }
```

Fig. 5: Transformación de la función del micro-kernel a través de las etapas de generación.

```
@irdl_op_definition
class vfmacc_vf_f32m1Op(IRDLOperation):
    name = "rvv.vfmacc_vf_f32m1Op"
    # Definir estáticamente y aplicar
    # restricciones  tipo de datos

    # Registro acumulador vectorial
    vd     = operand_def(RVVFloat32M1Type)
    # Puntero de memoria abstracto
    memref = operand_def(MemRefType)
    # Desplazamiento para extraer escalar
    offset = operand_def(IndexType)
    # Registro vectorial obtenido
    vs     = operand_def(RVVFloat32M1Type)
    # Limite entero de longitud del vector
    avl    = operand_def(IndexType)
    # Acumulador devuelto
    result = result_def(RVVFloat32M1Type)
```

Fig. 6: Definición en xDSL de la operación `vfmacc_vf_f32m1Op` en el dialecto `rvv`.

registro acumulador actual C_r para esta columna de C, obligatorio por los `iter_args` de SSA; `%4 (memref<...>)` es el puntero de memoria al empaquetado de B_c; `%53 (index)` es el desplazamiento entero requerido para extraer el escalar de B_c; `%52 (!rvv.vfloat32m1)` es el registro vectorial A_r previamente obtenido de A_c; y `%51 (index)` es la longitud del vector que controla los límites del hardware de la operación.

El espacio de nombres `rvv.*`, que incluye tanto la instrucción FMA como el tipo de datos `!rvv.vfloat32m1`, es nuestro dialecto RVV-MLIR personalizado y propuesto.

Etapa 3: Reducción de la IR a `emitc` (xDSL). La estructura actual de la IR, que mezcla los dialectos `scf`, `memref`, `arith` y el personalizado `rvv`, no se puede compilar directamente. Por lo tanto, esta etapa del flujo aplica pases de transformación específicos uno por uno para realizar la reducción de estos dialectos de alto nivel a `emitc`. Concretamente, como resultado de esta etapa, nuestra IR de MLIR generada por xDSL converge en un código de dialecto `emitc` puro de MLIR. Este resultado se alcanza de la siguiente manera:

Primero, xDSL aborda las abstracciones de memoria a través del pase `MemRefToEmitCPass`. El tipo `memref` proporciona un mecanismo para representar arreglos multidimensionales con strides y diseños complejos. Sin embargo, los compiladores de C operan nativamente sobre bloques de memoria planos gestionados por punteros C estándar. Este pase identifica las ocurrencias de `memref` (como las estructuras Ac, Bc, C) y las traduce a abstracciones de punteros C estándar. Podemos seguir esta transformación específica a través de la firma de la función del núcleo en las etapas del flujo resumidas en la Figura 5. La líneas 3–5 definen la función xDSL con los objetos `memref`; las líneas 9–11 muestran las estructuras `emitc.ptr`; y las líneas 14–15 los punteros C.

A continuación, el pase `SCFToEmitCPass` resuelve las restricciones del flujo de control impuestas por la forma SSA de MLIR. Descompone el bloque `scf.for` en un bucle for imperativo de `emitc`, reemplazando los parámetros transportados por el bucle SSA con variables C (`emitc.variable`) y operaciones de asignación directa (`emitc.assign`).

Continuando con la reducción, el pase `RVVToEmitCPass` traduce las operaciones vectoriales a RVV para C. Para lograr esto, nuestro diseño define un dialecto personalizado `rvv` de MLIR dentro de xDSL mediante la subclase de la clase `IRDLOperation` para cada instrucción de hardware; ver Figura 6. Por ejemplo, la instrucción FMA `vfmacc_vf_f32m1Op` (línea 3) se construye declarando sus entradas requeridas usando `operand_def` (líneas 7–16) y su salida usando `result_def` (línea 18).

Durante esta etapa de reducción, el pase `RVVToEmitCPass` utiliza la infraestructura `RewritePattern` para mapear estas construcciones en nodos de `emitc`; ver Figura 7. En su código se encuentra la función `match_and_rewrite` (líneas 4–29), que intercepta instancias de las operaciones `rvv`s durante el recorrido del árbol. Cuando el rewriter encuentra un `vfmacc_vf_f32m1Op`, extrae los operandos SSA, como el acumulador, las entradas de memoria y la longitud del vector (líneas 14–24). Luego, el rewriter construye la representación estructural equivalente, inyecta nodos `EmitCSubscriptOp` para calcular los punteros de memoria y construye una operación `emitc.CallOpaqueOp` (líneas 12–25). Esto instruye al traductor a emitir el literal de cadena pura de la instrucción vectorial en C: `"__riscv_vfmacc_vf_f32m1"` (línea 13).

La operación FMA completamente transformada aparece en el código `emitc`, como se muestra en la Figura 8. Las llamadas opacas de `emitc` encapsulan tanto las cadenas C traducidas como las restricciones de tipo de datos. Podemos desglosar los cuatro argumentos pasados a esta llamada de hardware: `%71 (!emitc.opaque<"vfloat32m1_t">)` es el registro acumulador C_r y el paso directo de SSA (`%42`) se transforma en una variable de estado (`%71`); `%73 (f32)` es el escalar extraído del vector B_c; `%62 (!emitc.opaque<"vfloat32m1_t">)` es el registro A_r, preservando el tipo de estructura de hardware; y `%59 (index)` es la longitud del vector que controla la operación.

Etapa 4: Traducción a C++. Con la IR comple-

```
class ConvertRVV_vfmacc_vf_f32m1_ToEmitC(
  RewritePattern):
  @op_type_rewrite_pattern
  def match_and_rewrite(op:vfmacc_vf_f32m10p,
                  rewriter:PatternRewriter):
    # 1. Definir la estructura C resultante
    vector_type = emitc.EmitC_OpaqueType(
                StringAttr("vfloat32m1_t"))

    # 2. Construir la llamada RVV de C con
    # los operandos de MLIR
    call_op = emitc.EmitC_CallOpaqueOp(
      callee="__riscv_vfmacc_vf_f32m1",
        call_args=[
          # Vector acumulador Cr
          op.vd,
          # Escalar Bc cargado
          load_op.result,
          # Vector Ar cargado
          op.vs,
          # Longitud del vector
          op.avl
        ],
        result_types=[vector_type],
    )

    # 3. Reemplazar la operación de dialecto
    # personalizado por EmitC
    rewriter.replace_op(op, call_op)
```

Fig. 7: Patrón de reescritura para convertir instrucciones FMA en llamadas de `emitc` para RVV.

```
// FMA de RVV con los tipos de datos
%74 = "emitc.call_opaque"(%71, %73, %62, %59)
     <{callee = "__riscv_vfmacc_vf_f32m1"}>
      :(!emitc.opaque<"vfloat32m1_t">, f32,
        !emitc.opaque<"vfloat32m1_t">, index)
      ->!emitc.opaque<"vfloat32m1_t">
```

Fig. 8: Representación en `emitc` de la instrucción FMA reducida.

```
vfloat32m1_t v42 = v21;
...
// Bucle kc
for(size_t i50=v11; i50<v1; i50+=v12) { // L6
  // Carga vectorial de Ac
  vfloat32m1_t v54 =
              __riscv_vle32_v_f32m1(v53, v52);

  // FMA de RVV
  vfloat32m1_t v63 = v42;
  float v64 = v5[v55];
  vfloat32m1_t v65 =
    __riscv_vfmacc_vf_f32m1(v63,v64,v54,v52);

  // Acumulación de resultados
  v42 = v65;
  ...
}
```

Fig. 9: Código C final emitido que implementa el bucle L6 con instrucciones vectoriales.

tamente escrita en nodos `emitc`, salimos del entorno xDSL e invocamos la herramienta oficial `mlir-translate`. Esta herramienta recorre la sintaxis de `emitc` y genera el código C final.

La traducción del bucle L6 se encuentra en la Figura 9, donde los complejos mecanismos de yield de SSA se reducen a actualizaciones de variables y las operaciones matemáticas FMA se representan como intrínsecos nativos del hardware RISC-V.

Podemos ver cómo la etapa de compilación final mapea los tipos de datos opacos de `emitc` estrictamente como estructuras `vfloat32m1_t` nativas de C. Además, los cinco argumentos originales de nuestra instrucción MLIR `rvv.vfmacc_vf_f32m10p` se han transformado sin problemas en las cuatro variables concretas requeridas por la instrucción vectorial `vfmacc`: `v63 (vfloat32m1_t)` es el registro acumulador C_r en ejecución, cargado directamente desde la variable del bucle (`v42`); `v64 (float)` es el valor escalar de B_c. Nótese cómo `emitc` resolvió con éxito los argumentos duales `memref/offset` de MLIR mapeándolos directamente a una solución C estándar (`v5[v55]`); `v54 (vfloat32m1_t)` es el registro vectorial A_r cargado; y `v52 (size_t)` es la longitud del vector de la operación.

Etapas 5, 6: Generación y despliegue del banco de pruebas. Nuestro flujo genera automáticamente un entorno de banco de pruebas completo en C, con rutinas de verificación e infraestructura mediante Makefile. Luego, orquesta el despliegue, la compilación nativa y la evaluación de los micro-kernels generados en el hardware RISC-V remoto objetivo, proporcionando un análisis de rendimiento de extremo a extremo.

El artefacto para generar micro-kernels y reproducir los resultados de este artículo está disponible en `https://github.com/JieGH/RVV_code_gen_via_MLIR_xDSL`.

V. Evaluación de rendimiento

En esta sección, evaluamos primero el rendimiento de los micro-kernels generados de forma independiente. Luego, evaluamos el impacto de los micro-kernels generados automáticamente dentro del algoritmo GEMM completo y comparamos el rendimiento resultante con la biblioteca OpenBLAS en varios escenarios. Los experimentos presentados en este trabajo se realizaron en las siguientes plataformas:

K230. Esta plataforma corresponde a una placa CanMV-K230 que integra el sistema en chip (SoC) K230 que funciona a 1,6 GHz y admite la extensión RISC-V Vector (RVV) versión 1.0 con una unidad vectorial de 128 bits. El procesador incluye 32 KB de caché de datos L1 y 256 KB de caché L2.

BPI. La placa BananaPi F3 integra un procesador SpaceMiT K1 de 8 núcleos que funciona a 1,6 GHz. Cada núcleo cuenta con una caché de datos L1 de 32 KB, mientras que un grupo de cuatro núcleos comparte una caché L2 de 512 KB. El procesador implementa la especificación RVV 1.0 con una longitud de vector de 256 bits.

Los experimentos utilizaron Python `3.13.9`, xDSL `v0.54.3` y LLVM `v22.0`. El código C generado se compiló con `g++-14 -march=rv64gcvzfh -mabi=lp64d`. Comparamos los resultados con OpenBLAS `v0.3.31`, compilada directamente para la K230 (`TARGET=RISCV64_ZVL128B`, núcleo de 8×8) y BPI (`TARGET=RISCV64_ZVL256B`, núcleo de 16×8). Reportamos el rendimiento de un solo núcleo en precisión simple en GFLOPS, promediando 1500 ejecuciones para los micro-kernels y 200 para GEMM. Se excluyó la evaluación paralela para aislar los efectos arquitectónicos del núcleo.

A. Evaluación de micro-kernels

La Figura 10 resume el rendimiento de diferentes configuraciones de micro-kernels evaluadas de forma aislada. Los resultados revelan una relación clara entre las dimensiones del micro-kernel y la utilización de vectores. En particular, las configuraciones en las que la dimensión m_r es un múltiplo del número de elementos que caben en un registro vectorial logran un mayor rendimiento. Para operandos FP32, esto corresponde a múltiplos de 4 elementos en K230 y múltiplos de 8 elementos en BPI.

Entre las configuraciones evaluadas, las mejores dimensiones del micro-kernel son 20×6 para K230 y 16×15 para BPI, alcanzando rendimientos máximos de 8,1 GFLOPS y 16,2 GFLOPS, respectivamente. Nótese que el rango de valores llega hasta 16,2 GFLOPS en ambos gráficos, revelando la diferencia de rendimiento entre dispositivos.

B. Evaluación de GEMM

Para evaluar el impacto de los micro-kernels en la implementación de GEMM, analizamos primero el rendimiento con matrices cuadradas, una configuración de prueba estándar ampliamente utilizada para evaluar implementaciones de álgebra lineal densa. En segundo lugar, evaluamos el algoritmo utilizando las formas de GEMM que surgen durante la inferencia del modelo BERT-Large. Esta doble evaluación nos permite valorar tanto el rendimiento máximo de la implementación como su eficacia en escenarios reales de ML. La Tabla I presenta los diferentes escenarios de GEMM para la evaluación.

Adoptamos la implementación de GEMM de `https://github.com/martineh/gemm_blis_family`. Este diseño proporcionó una implementación eficiente de GEMM en caché que nos permitió conectar directamente nuestro código de núcleo generado por xDSL y evaluar su rendimiento.

La Figura 11 presenta una comparativa de rendimiento entre la implementación propuesta con xDSL y con OpenBLAS en ambas cargas de trabajo.

Para la carga de trabajo de matrices cuadradas en K230, GEMM+xDSL supera consistentemente a OpenBLAS en todas las capas evaluadas (S1–S5). OpenBLAS alcanza valores de rendimiento entre aproximadamente 4,5 y 4,9 GFLOPS, mientras que xDSL alcanza entre 5,1 y 5,4 GFLOPS. Esto corresponde a una mejora promedio de aproximadamente un 10–15 %. Cabe destacar que la implementación optimizada utiliza la misma configuración de micro-kernel (20×6) para todas las capas, lo que indica que esta configuración proporciona una utilización estable y eficiente de los recursos de hardware para matrices con formas regulares.

Para la carga de trabajo BERT-Large en la misma plataforma, las mejoras de rendimiento son más variables pero siguen siendo favorables al enfoque propuesto. OpenBLAS alcanza entre 3,9 y 4,5 GFLOPS, mientras que xDSL llega a entre 4,6 y 5,9 GFLOPS dependiendo de la capa. La mayor mejora ocurre en la capa B2, donde el rendimiento aumenta de 4,0 GFLOPS a 5,9 GFLOPS, lo que corresponde a una aceleración cercana al 47 %. Otras capas muestran mejoras de entre el 10 % y el 35 %. A diferencia de la carga de trabajo cuadrada, se seleccionan diferentes configuraciones de micro-kernels (p. ej., 24×5, 20×6 y 8×11) para cada capa, reflejando la mayor diversidad de formas de las matrices en los modelos basados en transformadores.

En BPI, las ventajas de xDSL se vuelven más pronunciadas. Para la carga de trabajo cuadrada, OpenBLAS alcanza entre 6,1 y 7,0 GFLOPS, mientras que xDSL llega a aproximadamente 8,3–8,6 GFLOPS. Esto corresponde a mejoras de rendimiento que van desde el 20 % hasta el 35 %. Al igual que en la K230, se utiliza una única configuración de micro-kernel (32×7) para todas las capas, demostrando que una configuración bien elegida puede ofrecer un rendimiento estable cuando las dimensiones de las matrices permanecen relativamente uniformes.

Las mayores ganancias de rendimiento se observan en la carga de trabajo BERT-Large en BPI. OpenBLAS alcanza entre 5,1 y 7,3 GFLOPS, mientras que GEMM+xDSL llega a entre 6,8 y 12,2 GFLOPS. En particular, la capa B2 muestra una mejora significativa de 5,1 GFLOPS a 12,2 GFLOPS, lo que representa una aceleración de más del doble. Las capas restantes muestran mejoras de entre el 15 % y el 30 % aproximadamente. Al igual que en los experimentos de la K230, se seleccionan diferentes configuraciones de micro-kernels (p. ej., 32×6, 16×15 y 32×7) para adaptarse mejor a las características computacionales de cada capa.

En general, los resultados demuestran que los micro-kernels generados con xDSL superan consistentemente a OpenBLAS en ambas plataformas y cargas de trabajo.

VI. Conclusiones

Este artículo ha presentado un flujo de compilación híbrido MLIR–xDSL que permite la generación de código RISC-V optimizado a partir de abstracciones MLIR de alto nivel. El enfoque propuesto aborda una limitación práctica de la infraestructura actual de MLIR: la falta de transformaciones de reducción completas que traduzcan los dialectos de alto nivel en implementaciones RVV conscientes del hardware y adecuadas para entornos de producción.

Aprovechando la flexibilidad de xDSL, hemos implementado etapas de reducción personalizadas que cierran la brecha entre las representaciones de alto nivel de MLIR y el dialecto `emitc`, lo que permite finalmente la generación automática de código C portátil que invoca instrucciones vectoriales RVV.

La evaluación demostró que los micro-kernels generados automáticamente con xDSL alcanzan un rendimiento comparable al de las implementaciones escritas a mano, lo que confirma la eficacia de la estrategia de generación de código propuesta. Al integrarse en una implementación completa de GEMM, los núcleos generados superaron consistentemente al modelo base de OpenBLAS en ambas cargas de tra-

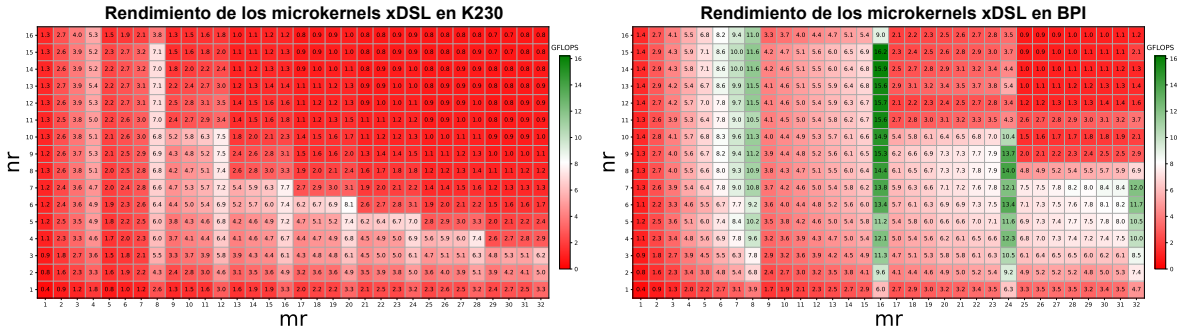


Fig. 10: Rendimiento de los micro-kernels para K230 (izquierda) y BPI (derecha).

Tabla I: Dimensiones de las capas (m, n, k) para los escenarios evaluados.

Caso	ID	m	n	k	Caso	ID	m	n	k
	S1	1000	1000	1000		B1	1024	384	1024
	S2	2000	2000	2000		B2	384	384	64
Cuadrada	S3	3000	3000	3000	BERT	B3	64	384	384
	S4	4000	4000	4000		B4	4096	384	1024
	S5	5000	5000	5000		B5	1024	384	4096

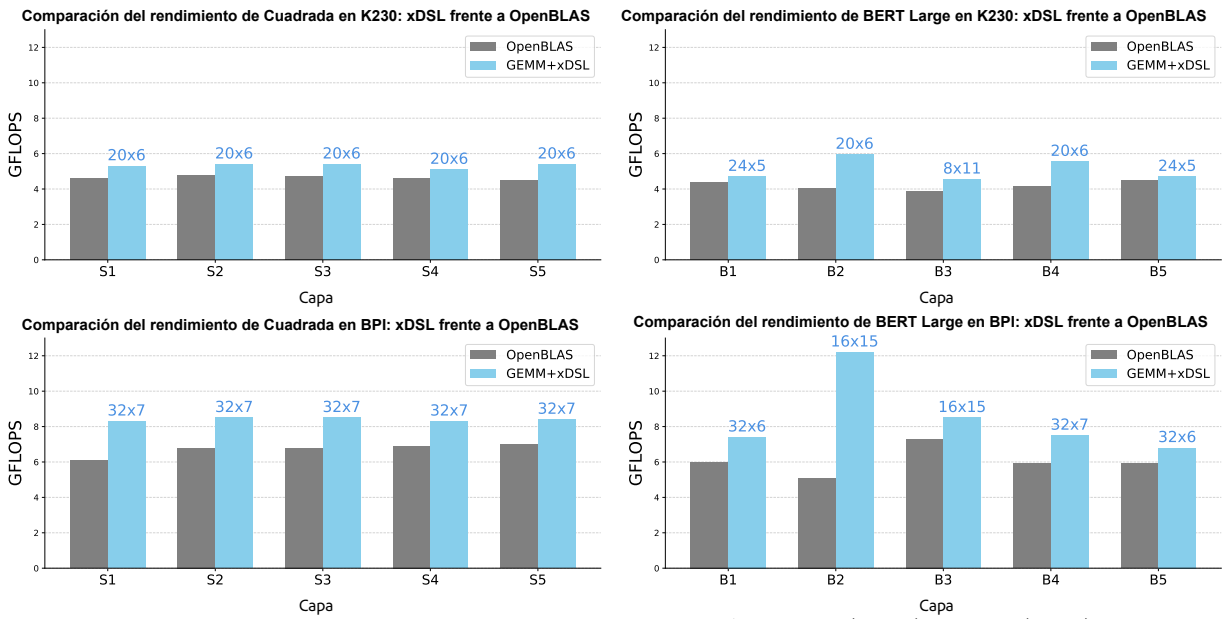


Fig. 11: Comparación entre GEMM con micro-kernels xDSL frente a OpenBLAS en K230 (arriba) y en BPI (abajo), para matrices cuadradas (izquierda) y BERT (derecha).

bajo evaluadas. Para matrices cuadradas, la implementación propuesta logró mejoras de rendimiento de hasta un 35 %, mientras que para las cargas de trabajo basadas en transformadores derivadas del modelo BERT-Large, las aceleraciones alcanzaron hasta 2,4×. Estos resultados destacan los beneficios de generar automáticamente múltiples configuraciones de micro-kernels y seleccionar la más adecuada para cada carga de trabajo o cada dimensión de capa.

Como trabajo futuro, planeamos ampliar el flujo de generación de código para soportar micro-kernels de precisión mixta (p. ej., INT8 para las matrices de entrada y INT32 para el acumulador). Esta evolución técnica implicará la reducción del dialecto `arith` para aprovechar las capacidades de extensión de signo y las instrucciones de multiplicación-acumulación con ensanchamiento (*widening multiply-accumulate*) de la extensión RVV. Se espera que este enfoque proporcione un aumento significativo del rendimiento, ya que la aritmética de enteros permite una mayor tasa de operaciones por segundo en comparación con la de coma flotante, optimizando así la eficiencia en tareas de inferencia de aprendizaje profundo.

Agradecimientos

Este trabajo ha recibido financiación de los proyectos PID2023-146569NB-C2 de MCIN/AEI/10.13039/501100011033 y CIPROM/2022/20 de la Generalitat Valenciana, el Proyecto DARE SGA1 de la EuroHPC JU bajo el Acuerdo de Subvención nº 101202459 y del Proyecto PCI2024-161687-3 financiado por MICIU/AEI/10.13039/501100011033 y por la Unión Europea "NextGenerationEU"/PRTR". La JU recibe apoyo del programa de investigación e innovación Horizonte Europa de la Unión Europea y de España,

Alemania, Chequia, Italia, Países Bajos, Bélgica, Finlandia, Grecia, Croacia, Portugal, Polonia, Suecia, Francia y Austria.

Referencias

[1] Jonathan Ragan-Kelley et al., "Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," 2013, PLDI, p. 519–530.

[2] Chris Lattner et al., "MLIR: Scaling compiler infrastructure for domain specific computation," in *IEEE/ACM Int. Symp. Code Gen. & Optimization*, 2021, pp. 2–14.

[3] Chris Lattner et al., "MLIR: A compiler infrastructure for the end of Moore's law," *CoRR*, 2020.

[4] Tianqi Chen et al., "TVM: end-to-end optimization stack for deep learning," *CoRR*, vol. abs/1802.04799, 2018.

[5] Yuka Ikarashi et al., "Exocompilation for productive programming of hardware accelerators," 2022, PLDI, p. 703–718.

[6] Nick Brown et al., "xDSL: A common compiler ecosystem for domain specific languages," Nov. 2022, Supercomputing 2023, SC23.

[7] Mathieu Fehr et al., "xDSL: Sidekick compilation for SSA-based compilers," in *ACM/IEEE Int. Symp. Code Gen. & Optimization*, 2025, p. 179–192.

[8] Kazushige Goto and Robert A. van de Geijn, "Anatomy of a high-performance matrix multiplication," *ACM Trans. Math. Softw.*, vol. 34, no. 3, pp. 12:1–12:25, May 2008.

[9] Guillermo Alaejos et al., "Micro-kernels for portable and efficient matrix multiplication in deep learning," *The Journal of Supercomputing*, pp. 1–24, 2022.

[10] P. Siwinska et al., "Enhancing transformer performance and portability through auto-tuning frameworks," in *The Journal of Supercomputing*, 2025.

[11] Uday Bondhugula, "High performance code generation in MLIR: an early case study with GEMM," 2020.

[12] Adrián Castelló et al., "Tackling the matrix multiplication micro-kernel generation with Exo," in *IEEE/ACM Int. Symp. Code Gen. & Optimization*, 2024, pp. 182–193.

[13] Adrián Castelló et al., "Evaluation of RVV-enabled COTS platforms with matrix multiplication and Exo," in *High Performance Computing. Lecture Notes in Computer Science, vol. 16091*, 2026, pp. 534–547.

[14] Alexandre Lopoukhine et al., "A multi-level compiler backend for accelerated micro-kernels targeting RISC-V ISA extensions," in *ACM/IEEE Int. Symp. Code Gen. & Optimization*, 2025, CGO '25, p. 163–178.

Tecnologías clúster, plataformas distribuidas, BigData y Deep Learning (3/4)

Enabling intra-node multi-GPU malleability for general-purpose applications

Alex Amenabar[1], Unai Garciarena[1], Javier Navaridas[1] and Jose A. Pascual[1]

Abstract— **High Performance Computing environments are becoming increasingly heterogeneous due to the widespread use of hardware accelerators, particularly GPUs. However, resources allocated to applications typically remain fixed throughout execution, leading to significant underutilization when GPUs are idle. Given the limited availability of GPUs, improving their utilization is critical for overall system efficiency. Malleable applications enable dynamic resource reconfiguration at runtime, offering a potential solution. Nevertheless, most existing approaches either do not support resizing the number of GPUs assigned to a job or are limited to deep learning applications. In this work, we propose a framework for developing intra-node multi-GPU malleable applications that can reconfigure the number of GPUs assigned at runtime. The framework also enables evaluating applications within workloads that combine both malleable and non-malleable applications. The results show that malleable applications can significantly improve throughput and system resource utilization across different scenarios. However, the current implementation does not account for the overhead associated with resource allocation and deallocation by the Resource Management System, which should be incorporated in future work to provide a more comprehensive evaluation. More broadly, this work represents a first step toward enabling dynamic GPU resource management in intra-node HPC systems.**

keywords— **Malleability, CUDA, High Performance Computing.**

I. Introduction

RESEARCH in many scientific and engineering domains, including physics, computer science, and chemistry, relies on computationally intensive simulations executed on High Performance Computing (HPC) systems. These systems are typically shared among multiple users and managed by Resource Management Systems (RMS), which schedule jobs and allocate resources according to predefined policies. Since applications often require different types of resources, such as CPUs for general computation and GPUs for accelerated workloads, the RMS must determine not only when and where jobs are executed, but also how heterogeneous resources are assigned.

However, in most HPC systems, resources are allocated to a job at the beginning of execution and remain fixed until completion. This static allocation model often leads to inefficient resource utilization [1], [2], [3], [4]. Users tend to overestimate resource requirements, and many applications exhibit multiple execution phases with different demands, such as CPU-bound computation, communication phases with little or no computation, and GPU-intensive phases. As a consequence, resources may remain idle or underutilized for significant portions of execution despite being exclusively allocated to a single job.

Malleable applications, which can dynamically adjust their resource allocation at runtime [5], [6], [7], offer a promising solution. By enabling jobs to expand or shrink according to system conditions, the RMS can improve utilization, efficiency, and throughput [8]. However, this introduces overhead: the RMS must monitor system state and make reconfiguration decisions, while applications must adapt to new resource configurations during execution.

Several works have explored malleability in applications based on the Message Passing Interface (MPI). Some rely on fault tolerance techniques, while others leverage MPI features to reconfigure applications without interrupting execution [9]. At the same time, HPC systems are becoming increasingly heterogeneous, with a growing reliance on GPUs driven by large-scale deep learning workloads [10], [11] and scientific applications. As GPUs are typically more limited than other resources, their efficient utilization has become a critical concern, particularly for multi-GPU applications.

Despite these advances, there is still a lack of comprehensive solutions that enable full malleability in GPU applications. Some approaches leverage task-parallel programming models [12] to dynamically balance workloads across GPUs; however, they do not provide true malleability, as the number of GPUs assigned to a job remains fixed regardless of utilization.

Moreover, current GPU programming frameworks, such as CUDA [13], [14], [15] and ROCm [16], [17], do not provide native support for dynamically reassigning GPUs to running applications. As a result, enabling malleability at the GPU level requires additional mechanisms beyond what is currently available in these systems. In this context, our work represents a first step toward supporting dynamic GPU reconfiguration for general-purpose intra-node HPC applications.

Some solutions support resizing the number of GPUs assigned to a job; however, they rely on spawning new processes and integrating them into the running application, introducing additional overhead. Furthermore, many of these approaches are tailored to deep learning workloads and cannot be easily generalized. They are also primarily designed for cloud environments, where resource availability is higher and scheduling objectives differ from traditional HPC systems, often prioritizing utilization at the cost of performance variability [18], [19].

To the best of our knowledge, few approaches support intra-node GPU malleability by dynami-

[1]Dept. of Computer Architecture and Technology, University of the Basque Country, e-mail: alex.amenabar@ehu.eus

cally resizing the number of GPUs assigned to a job without spawning new processes, while also targeting general-purpose multi-GPU applications. In this work, we propose a framework for developing intra-node multi-GPU applications that incorporate malleability, enabling dynamic adjustment of GPU resources within a node with minimal overhead. This is limited to resizing jobs using only the GPUs available within a single node.

Reconfiguring a GPU application involves the following stages: (i) allocating new resources, (ii) redistributing data, (iii) releasing unused resources, (iv) reconfiguring kernels, and (v) resuming execution. The framework provides an API for managing these stages, including mechanisms for defining data redistribution across GPUs. However, not all data transfers are handled automatically, requiring user-defined functions for more complex redistributions.

The framework also supports workloads composed of both malleable and non-malleable applications, enabling evaluation under realistic conditions. Although jobs are executed and reconfigured dynamically, the current implementation does not include a full RMS. Instead, a main thread acting as a simplified RMS manages job execution by spawning threads representing individual jobs.

We evaluate the framework by measuring the cost of reconfiguration under different data sizes and scenarios, and by analyzing workloads composed of malleable and non-malleable applications. Results show that the cost of data redistribution across GPUs is low, even for relatively large data sizes, and remains small compared to overall execution time. Furthermore, malleability improves GPU utilization and system throughput. However, since the implementation does not include a full RMS, the overhead associated with system monitoring and scheduling decisions is not considered.

II. Background and related Work

The RMS is responsible for launching jobs in an HPC facility according to user-specified resource requirements. According to Feitelson and Rudolph [20], jobs in HPC systems can be classified as *static* or *dynamic*. Static jobs cannot be resized during execution, whereas dynamic jobs can adapt to changes in resource availability. Static jobs can be further categorized as *rigid* or *moldable*, with moldable jobs allowing execution with a variable number of resources before execution. Dynamic jobs can be classified as *evolving* or *malleable*; evolving jobs decide autonomously when to reconfigure, while malleable jobs follow instructions from the RMS. Jobs that are both moldable and malleable are commonly referred to as *flexible* or *elastic*.

Solutions for enabling malleability have followed different approaches. Early work extended fault-tolerance tools. Since these mechanisms store application state to allow stopping and resuming after errors, they can be adapted to support reconfiguration before resuming execution. However, several works extending fault-tolerance tools for MPI-based applications using checkpoint/restart (C/R) mechanisms [21], [22], [23] do not incorporate GPUs, limiting their applicability in modern heterogeneous HPC systems. Some works provide malleability in multi-GPU applications based on these techniques (also referred to as elasticity), but they primarily focus on large-scale deep learning training. For example, ElasticDL [24] is a Kubernetes-based deep learning framework that supports fault tolerance and elastic scheduling, providing primitives for defining distributed PyTorch or TensorFlow tasks [25]. Similarly, [26] leverages ULFM [27], [28] to incorporate fault management into MPI applications. When a process fails, all processes can identify failed operations and rerun only those without rolling back to the last checkpoint. Elasticity is handled upon failure, allowing the system to continue with fewer processes, recover failed ones, or add new processes. Nevertheless, elasticity in this context is tied to fault events rather than general resource management.

Apart from fault-tolerance–based solutions, some works leverage MPI features, which support dynamic process spawning since version 2.0 [29] via *MPI_Comm_spawn*. Reconfigurations and data redistribution can be managed through MPI communication, reducing overhead compared to checkpoint-based approaches. Frameworks such as Elastic MPI [30], Flex-MPI [31], and Dynamic Management of Resources (DMR) [32] extend MPI functionalities to enable malleable applications. Elastic MPI provides a small API for incorporating malleability into existing MPI applications and extends SLURM [33], [34] through a plugin to support scheduling reconfigurations. Flex-MPI provides an execution model for malleability and dynamic load balancing, including modules for monitoring, predicting beneficial reconfigurations, managing data redistribution, and balancing load; however, it requires significant modifications to the application, which may limit its adoption.

The DMR framework combines an RMS (SLURM) with an MPI-based runtime, providing a communication layer that enables malleable application execution within workloads. It offers an API and semi-automatic data redistribution. DMR was the first framework to support malleability for GPU applications [35]. However, it achieves malleability by spawning MPI processes tied to GPUs rather than directly reconfiguring GPU allocations, introducing additional overhead and complexity. In contrast, our proposal does not require MPI, allowing single-process multi-GPU applications to be reconfigured by redistributing data from previously used GPUs to newly allocated ones.

Other approaches based on the task-parallel programming model, while not providing malleability, enable dynamic resource management. Frameworks such as StarPU [36] and PaRSEC [37] support task-based applications where heterogeneous resources are dynamically scheduled. However, these approa-

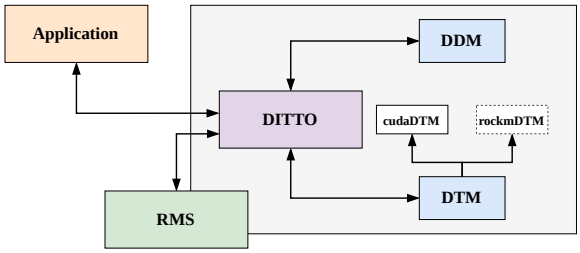


Fig. 1: Software architecture of DITTO.

ches focus on task scheduling rather than dynamic resource resizing and therefore do not address malleability at the job level. Moreover, current GPU programming models (e.g., CUDA and ROCm) do not provide native support for dynamic GPU reassignment during execution, which further limits the applicability of existing approaches.

Finally, some works provide elastic GPU scheduling by adapting GPU allocations during runtime through process spawning. However, most focus on cloud environments and distributed deep learning. For example, Kale [38] uses Kubernetes to scale GPU allocations and predicts workload behavior to decide reconfigurations. Although it can be used in HPC, it is primarily designed for cloud-native environments, where Kubernetes-based orchestration is standard and resource constraints differ from traditional HPC systems.

Our proposal differs from these approaches by enabling malleability in multi-GPU applications without focusing on distributed deep learning and without relying on fault-tolerance mechanisms. Additionally, our proposal does not require MPI, allowing single-process multi-GPU applications to be reconfigured.

III. Design and Implementation

In this section, we describe the design and implementation of the proposed framework, including its architecture, mechanisms for developing malleable applications, and workload execution model. We also discuss its current limitations.

A. DITTO framework

Malleable multi-GPU applications require redistributing data and reconfiguring kernels when the RMS triggers a reconfiguration. Therefore, the framework must coordinate both the RMS and the application, acting as an intermediary between them. Figure 1 shows the software architecture of DITTO, whose core module provides the functionality for developing malleable multi-GPU applications and mediates communication between the application and the RMS.

DITTO consists of two main submodules: the *Data Distribution Module* (DDM), which determines how data is redistributed during reconfiguration, and the *Data Transmission Module* (DTM), which performs the corresponding transfers. The DTM is backend-dependent, allowing flexibility across different GPU technologies.

Algorithm 1: Algorithm for reconfiguring an application.

```
1  state ← getState();
2  reconfData ← getReconfData();
3  for DTI in arrDTI do
4      DTI.mGPU2CPU(state);
5  state ← updateState(reconfData);
6  for DTI in arrDTI do
7      reconfigureDTI(DTI, state);
8      DTI.mCPU2GPU(state);
9  reconfigureKernels();
10 notifyReconfDone();
```

The current implementation does not still interfaces with a RMS (e.g., SLURM or FLUX [39], [40]). Instead, the main thread simulates an RMS by spawning a thread per job (using pthreads [41]). Each job remains active within the same thread for its entire execution. Communication is implemented through shared variables protected by locks, which act as notification mechanisms. As a result, jobs execute within a single process, and GPUs are assigned at startup, while scheduling and reconfiguration are handled internally.

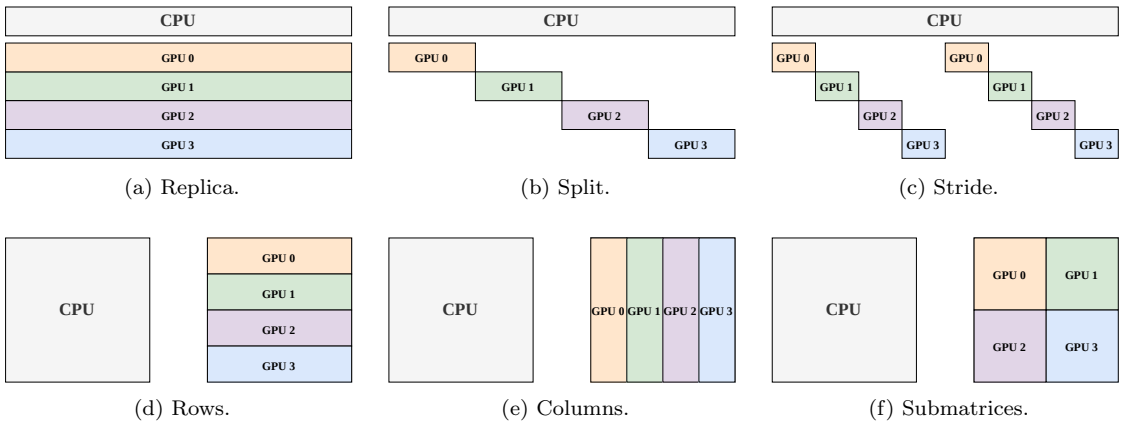
The framework defines notifications for coordination between the RMS and applications, including: (i) RMS notifications indicating pending reconfigurations and new GPU allocations, and (ii) application notifications confirming completion. Additionally, applications can provide hints to guide scheduling decisions, such as indicating phases without GPU usage or requesting additional GPUs for compute-intensive sections.

DITTO also provides a mechanism for data redistribution based on structures called *Data Transfer Interfaces* (DTIs). For each data structure required on the GPU, the user defines a DTI describing how the data is partitioned across GPUs. Once defined, the framework automatically handles data redistribution during reconfiguration.

Algorithm 1 illustrates the reconfiguration process. First, data is transferred from GPUs to CPU memory using *mGPU2CPU*. Then, the application state is updated and DTIs are reconfigured. Next, data is redistributed to the newly assigned GPUs using *mCPU2GPU*. Finally, kernels are updated and the application notifies the RMS that reconfiguration has completed.

B. Data Distribution Module

The *Data Distribution Module* (DDM) uses information from DTIs to prepare the data required by the DTM for redistribution across GPUs during reconfiguration. DTIs can be defined in two ways: (i) automatic patterns for 1D arrays and 2D matrices, and (ii) user-defined functions for more complex data structures and distribution schemes. These corres-



(a) Replica. (b) Split. (c) Stride.

(d) Rows. (e) Columns. (f) Submatrices.

Fig. 2: Automatic data redistribution patterns offered by DITTO for 1D arrays and 2D matrices.

Algorithm 2: Algorithm for enabling malleability during application execution.

```
Input: N
1 appData ← loadData(N);
2 initDITTO(nGPUs);
3 dscp ← createDescription(N, ptrn2);
4 createAutomaticDTI(appData, dscp);
5 for s ← 0 to steps do
6     reconfigurationPoint();
7     runStep(appData);
```

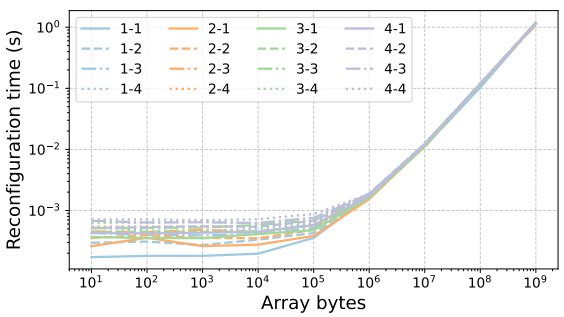


Fig. 3: Time elapsed reconfiguring an application for arrays of increasing sizes, for all combinations of source and destination GPUs.

pond to *automatic* and *manual* DTIs, respectively.

Figure 2 illustrates the distribution patterns available for both 1D arrays and 2D matrices. For 1D arrays (Figures 2a, 2b, and 2c), the framework supports full replication across GPUs, contiguous partitioning into segments assigned to each GPU, and partitioning into user-defined blocks that are distributed across GPUs, resulting in non-contiguous segments per device. For 2D matrices (Figures 2d, 2e, and 2f), the supported patterns include row-wise partitioning, column-wise partitioning, and division into submatrices.

Automatic DTIs are created using the function *createAutomaticDTI()*, which takes as input the data structure in CPU memory (1D array or 2D matrix), its dimensions, and a description of the distribution pattern. For more complex cases, manual DTIs can be defined using *createManualDTI()*, which requires user-provided functions for CPU-to-GPU and GPU-to-CPU data transfers. Multiple DTIs can be defined within an application. Each DTI is registered in an internal structure managed by the framework, and all DTIs are iterated during reconfiguration to perform data redistribution.

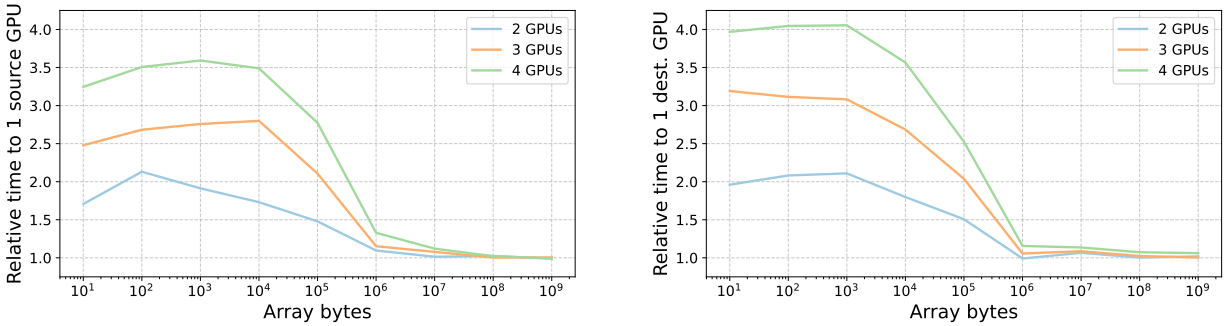
C. Malleable application development.

DITTO is designed to minimize programmer effort when incorporating malleability. A few function calls and the definition of DTIs are sufficient to enable dynamic reconfiguration during execution. To enable malleability, the developer must initialize the framework using *initDITTO()*, define DTIs when needed, and insert reconfiguration points using *reconfigurationPoint()*. Optional calls such as *noGPUsSection()* and *requestGPUs()* allow the application to inform the RMS about GPU usage phases. Algorithm 2 illustrates this process, where a DTI is defined for an array of size N using the *split* distribution pattern (Figure 2b) and *reconfigurationPoint()* is invoked at each iteration. DTIs are automatically registered upon creation and do not require explicit management, although they can be stored and modified if needed.

The current implementation presents several limitations. First, it does not integrate with a real RMS, as scheduling and resource management are simulated within a single process. Second, resource allocation is not performed at the system level, which prevents evaluating the full overhead of real deployments. Finally, the framework requires user-defined DTIs for complex data structures, which may increase development effort.

IV. Experimental results

We performed several experiments to evaluate (i) the correctness of the framework, (ii) the cost of reconfiguring applications for different data sizes, and (iii) the benefits of malleable applications.



(a) Time to move data from multiple GPUs to the CPU normalized to a single GPU.

(b) Time to move data from the CPU to multiple GPUs normalized to a single GPU.

Fig. 4: Relative time required for moving data from and to multiple GPUs compared to doing it from or to a single GPU.

A. Experimental setup

All experiments were executed on a single node equipped with two 20-core Intel Xeon Silver 4210R processors, each at 2.4 GHz, with 192 GB of memory. The system includes four RTX 2080 Ti GPUs (11 GB each) with CPU-GPU communication performed through a 16x PCIe 3.0 bus.

The correctness of the framework was evaluated using a simple application in which, at each iteration, one is added to every element of an array while multiple reconfigurations are performed. After each reconfiguration, we verified that the elements assigned to each GPU were correct according to the array size, the number of GPUs, and the selected data redistribution pattern.

The cost of reconfiguration was evaluated by running experiments in which an application waits for a reconfiguration after starting execution. We measured the time required to complete reconfigurations for array sizes ranging from 10 bytes to 10 GB, increasing the size by a factor of 10 at each step and considering all combinations of source and destination GPUs.

Finally, we executed two workloads composed of malleable and non-malleable applications to evaluate the benefits of malleability. Algorithms 3 and 4 illustrate the two applications used. The first represents a simple iterative application that distributes an array of N elements across GPUs and executes a GPU kernel (shown in Algorithm 5) at each iteration, with a reconfiguration point placed at the beginning of each step.

The second extends this application by introducing two phases per iteration: a CPU phase and a GPU phase. During the CPU phase, the application releases its GPUs, and later requests them again before continuing execution. Both malleable and non-malleable versions were implemented to compare their behavior under different workloads.

Algorithm 3: Iterative malleable application.

```
Input: N, steps, dSteps
data ← loadData(N);
initDITTO(nGPUs);
dscp ← createDescription(N, ptrn2);
dti ← createAutomaticDTI(data, dscp);
for s ← 0 to steps do
    reconfigurationPoint();
    for gpu ← 0 to getNumberGPUs() do
        runGPU(dti.dData[gpu], dti.dN[gpu],
          dSteps);
```

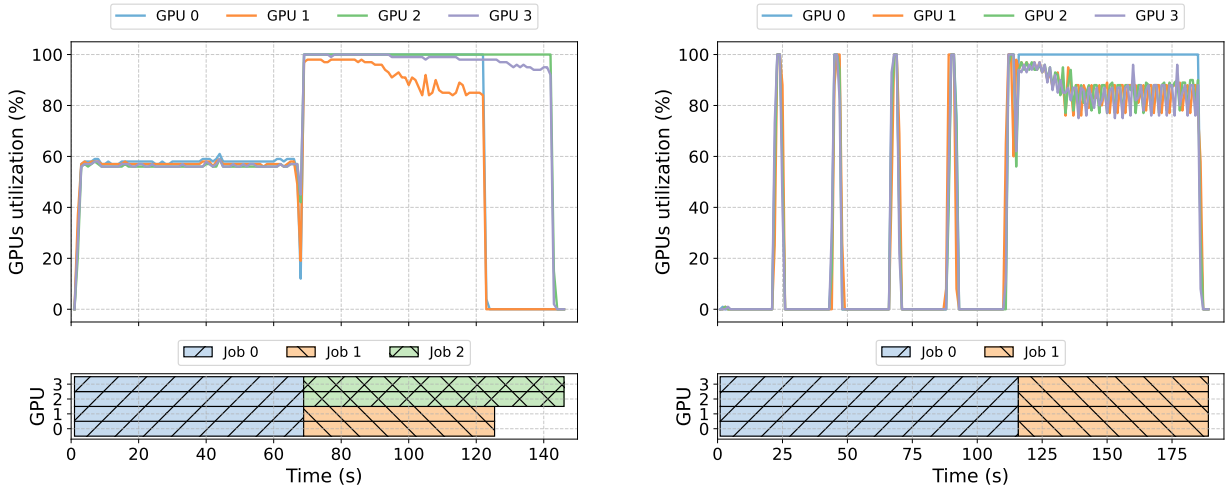
Algorithm 4: Iterative malleable application with two phases.

```
Input: N, steps, cpuSteps, gpuSteps
data ← loadData(N);
initDITTO(nGPUs);
dscp ← createDescription(N, ptrn2);
dti ← createAutomaticDTI(data, dscp);
for s ← 0 to steps do
    noGPUsSection();
    for i ← 0 to N do
        for s ← 0 to cpuSteps do
            data[i] ← data[i] * 10^-5 - 10^-5;
    requestGPUs();
    for gpu ← 0 to getNumberGPUs() do
        runGPU(dti.dData[gpu], dti.dN[gpu],
          gpuSteps);
```

B. Analysis of the results

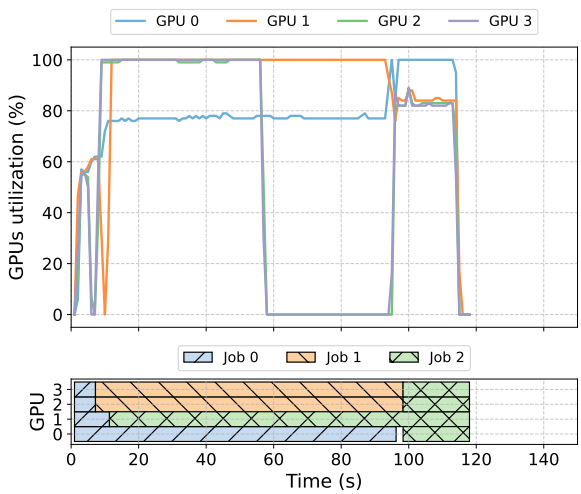
Figure 3 shows the time required (in seconds) to reconfigure the application for arrays of increasing sizes, from 10 bytes up to 10 GB. The reconfiguration process includes: (i) transferring data from the GPUs to the CPU, (ii) updating the application state, (iii) reconfiguring DTIs, and (iv) transferring data back to the GPUs.

The results show that the time required for reconfiguration is relatively small given the amount of data transferred, especially since reconfigurations are infrequent compared to computation steps; therefore, computation time dominates the overall execution time. They also show that when smaller amounts of

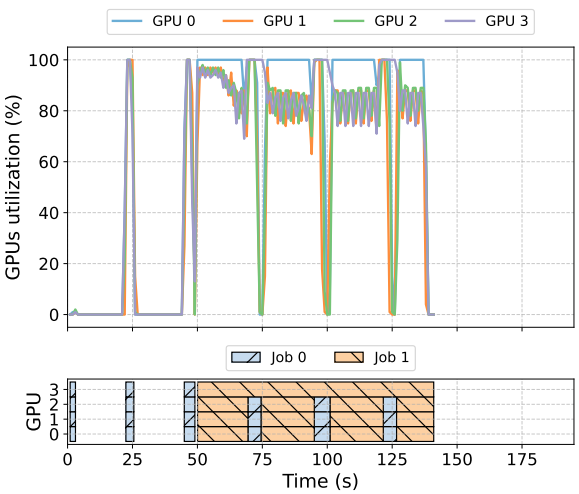


(a) Non-malleable version of the first workload.

(b) Non-malleable version of the second workload.



(c) Malleable version of the first workload.



(d) Malleable version of the second workload.

Fig. 5: Comparison of the execution and GPU allocation for the non-malleable (above) and malleable (below) versions of two workloads.

Algorithm 5: Implementation of the *runGPU()* kernel executed in the GPU.

```
Input: dAppData, dN, dSteps
thrId ← getThrId();
if thrId < dN then
    val ← dAppData[thrId];
    for s ← 0 to dSteps do
        val ← val * 10^-5 - 10^-5;
    val ← val / 2;
    dAppData[thrId] ← val;
```

data are redistributed, the number of source or destination GPUs has a greater impact.

This observation is further supported by Figure 4, which shows the relative time required for data transfers. When small data sizes are involved, the number of GPUs significantly affects transfer time; however, as data size increases, this overhead becomes negligible. This behavior can be explained by PCIe communication, where protocol headers introduce overhead that dominates for small transfers [42]. Because transfers are performed sequentially, this effect is amplified, whereas for larger data sizes, bandwidth utilization improves and transfer times become comparable across different GPU counts.

Figure 5 shows the execution of two workloads with malleable and non-malleable versions. In Workload 1, composed of three jobs corresponding to the application in Algorithm 3, the non-malleable version (Figure 5a) allocates all four GPUs to the first job, although it only utilizes approximately 60 % of each, preventing other jobs that require fewer GPUs from starting until it finishes. In contrast, the malleable version (Figure 5c) dynamically reduces the number of GPUs assigned to the first job during execution, allowing subsequent jobs to start earlier. As jobs complete, GPUs are reassigned to remaining jobs, improving overall utilization and reducing idle resources.

Workload 2, shown in Figures 5b and 5d, consists of two jobs: one corresponding to the applica-

tion with CPU and GPU phases (Algorithm 4) and another iterative application. In the non-malleable case, the first job occupies all GPUs throughout execution, including during CPU phases where GPUs remain idle, preventing the second job from starting. In the malleable version, GPUs are released during CPU phases and reassigned when needed, enabling concurrent execution and improving resource utilization.

Overall, these results demonstrate that malleability improves resource utilization and enables more efficient execution of intra-node multi-GPU workloads.

V. Conclusions and future work

This paper presents DITTO, a framework for developing malleable intra-node multi-GPU applications and evaluating them within workloads composed of both malleable and non-malleable applications. The framework enables malleability to be integrated with minimal effort, requiring only a few function calls and the definition of data structures to support automatic data redistribution during reconfiguration.

The framework includes several patterns for defining data distribution for 1D arrays and 2D matrices, which are commonly used in scientific applications. The results show that the reconfiguration cost is small, even for relatively large data sizes, and that malleability improves throughput and system utilization, even in small workloads composed of short jobs across different scenarios.

It is important to note that the current implementation does not incorporate a real RMS. Job resource allocation occurs within a single process that simulates the RMS, meaning that the overhead associated with resource allocation and deallocation, as well as communication between the RMS and the application, is not captured. Additionally, the framework lacks a job scheduler, so workloads must be defined manually.

Future work should focus on integrating a real RMS and scheduler to provide more realistic overheads in both scheduling and resource management. It should also explore larger data volumes and alternative communication methods, such as asynchronous and peer-to-peer transfers. More broadly, this work represents a first step toward enabling dynamic GPU resource management in intra-node HPC systems, particularly in the absence of native support for GPU reassignment in current programming frameworks such as CUDA and ROCm.

Acknowledgements

This work is supported by the Basque Government grants IT1504-22 and KK-2025/00098. It is also supported by the grants PID2023-152390NB-I00 and CNS2023-144315 funded by MICIU/AEI/10.13039/9/501100011033, "European Union NextGenerationEU/PRTR" and by "FEDER funds". Dr. Javier Navaridas holds a Ramón y Cajal fellowship funded by the Spanish Ministry of Science, Innovation and Universities (RYC2018-024829-I), funded by MCIN/ AEI/10.13039/501100011033 and, as appropriate, by "ESF Investing in your future" or by "European Union NextGenerationEU/PRTR".

References

[1] Robert Keßler, Simon Volpert, and Stefan Wesner, "Towards improving resource allocation for multi-tenant hpc systems: An exploratory hpc cluster utilization case study," in *2024 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops)*, 2024, pp. 66–75.

[2] Efe Sencan, Dhruva Kulkarni, Ayse Coskun, and Kadidia Konate, "Analyzing gpu utilization in hpc workloads: Insights from large-scale systems," in *Practice and Experience in Advanced Research Computing 2025: The Power of Collaboration*, New York, NY, USA, 2025, PEARC '25, Association for Computing Machinery.

[3] Hoda Sedighi, Fetahi Wuhib, and Roch H. Glitho, "Dynamic task scheduling and adaptive gpu resource allocation in the cloud," *IEEE Transactions on Network and Service Management*, vol. 23, pp. 1112–1127, 2026.

[4] Yanjie Gao, Yichen He, Xinze Li, Bo Zhao, Haoxiang Lin, Yoyo Liang, Jing Zhong, Hongyu Zhang, Jingzhou Wang, Yonghua Zeng, Keli Gui, Jie Tong, and Mao Yang, "An empirical study on low gpu utilization of deep learning jobs," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, New York, NY, USA, 2024, ICSE '24, Association for Computing Machinery.

[5] Ahmad Tarraf, Martin Schreiber, Alberto Cascajo, Jean-Baptiste Besnard, Marc-André Vef, Dominik Huber, Sonja Happ, André Brinkmann, David E. Singh, Hans-Christian Hoppe, Alberto Miranda, Antonio J. Peña, Rui Machado, Marta Garcia-Gasulla, Martin Schulz, Paul Carpenter, Simon Pickartz, Tiberiu Rotaru, Sergio Iserte, Victor Lopez, Jorge Ejarque, Heena Sirwani, Jesus Carretero, and Felix Wolf, "Malleability in modern hpc systems: Current experiences, challenges, and future opportunities," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 9, pp. 1551–1564, 2024.

[6] Travis Desell, Kaoutar El Maghraoui, and Carlos A Varela, "Malleable applications for scalable high performance computing," *Cluster Computing*, vol. 10, no. 3, pp. 323–337, 2007.

[7] Jesus Carretero, David Exposito, Alberto Cascajo, and Raffaele Montella, "Malleability techniques for hpc systems," in *Parallel Processing and Applied Mathematics*, Roman Wyrzykowski, Jack Dongarra, Ewa Deelman, and Konrad Karczewski, Eds., Cham, 2023, pp. 77–88, Springer International Publishing.

[8] Abhishek Gupta, Bilge Acun, Osman Sarood, and Laxmikant V. Kalé, "Towards realizing the potential of malleable jobs," in *2014 21st International Conference on High Performance Computing (HiPC)*, 2014, pp. 1–10.

[9] David E Bernholdt, Swen Boehm, George Bosilca, Manjunath Gorentla Venkata, Ryan E Grant, Thomas Naughton, Howard P Pritchard, Martin Schulz, and Geoffroy R Vallee, "A survey of mpi usage in the us exascale computing project," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 3, pp. e4851, 2020.

[10] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang, "Characterization and prediction of deep learning workloads in large-scale gpu datacenters," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2021, SC '21, Association for Computing Machinery.

[11] Weizheng Xu, Youtao Zhang, and Xulong Tang, "Parallelizing dnn training on gpus: Challenges and opportunities," in *Companion Proceedings of the Web Conference 2021*, New York, NY, USA, 2021, WWW '21, p. 174–178, Association for Computing Machinery.

[12] Peter Thoman, Kiril Dichev, Thomas Heller, Roman Iakymchuk, Xavier Aguilar, Khalid Hasanov, Philipp Gschwandtner, Pierre Lemarinier, Stefano Markidis, Herbert Jordan, et al., "A taxonomy of task-based parallel programming technologies for high-performance computing," *The Journal of Supercomputing*, vol. 74, no. 4, pp. 1422–1434, 2018.

[13] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron, "Scalable parallel programming with cuda: Is

cuda the parallel programming model that application developers have been waiting for?," *Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[14] NVIDIA Corporation, "NVIDIA CUDA c programming guide," 2024, Version 12.x.

[15] Jason Sanders and Edward Kandrot, *CUDA by example: an introduction to general-purpose GPU programming*, Addison-Wesley Professional, 2010.

[16] Inc. Advanced Micro Devices, "ROCm Documentation v6.0.3," 2024, Accessed: 2026-04-10.

[17] Inc. Advanced Micro Devices, "ROCm - Radeon open compute platform," 2026, Accessed: 2026-04-10.

[18] Zhisheng Ye, Wei Gao, Qinghao Hu, Peng Sun, Xiaolin Wang, Yingwei Luo, Tianwei Zhang, and Yonggang Wen, "Deep learning workload scheduling in gpu datacenters: A survey," *ACM Comput. Surv.*, vol. 56, no. 6, Jan. 2024.

[19] Tianhao Fu, Zehua Yang, Zhisheng Ye, Chenxiang Ma, Yang Han, Yingwei Luo, Xiaolin Wang, and Zhenlin Wang, "A survey on the scheduling of dl and llm training jobs in gpu clusters," *Chinese Journal of Electronics*, vol. 34, no. 3, pp. 881–905, 2025.

[20] Dror G. Feitelson, "Packing schemes for gang scheduling," in *Job Scheduling Strategies for Parallel Processing*, Dror G. Feitelson and Larry Rudolph, Eds., Berlin, Heidelberg, 1996, pp. 89–110, Springer Berlin Heidelberg.

[21] Sathish S Vadhiyar and Jack J Dongarra, "Srs: A framework for developing malleable and migratable parallel applications for distributed systems," *Parallel Processing Letters*, vol. 13, no. 02, pp. 291–312, 2003.

[22] Kaoutar El Maghraoui, Boleslaw K Szymanski, and Carlos Varela, "An architecture for reconfigurable iterative mpi applications in dynamic environments," in *International Conference on Parallel Processing and Applied Mathematics*. Springer, 2005, pp. 258–271.

[23] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R De Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2010, pp. 1–11.

[24] Jun Zhou, Ke Zhang, Feng Zhu, Qitao Shi, Wenjing Fang, Lin Wang, and Yi Wang, "Elasticdl: A kubernetes-native deep learning framework with fault-tolerance and elastic scheduling," in *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, New York, NY, USA, 2023, WSDM '23, p. 1148–1151, Association for Computing Machinery.

[25] T Kubernetes, "Kubernetes," *Kubernetes. Retrieved May*, vol. 24, pp. 2019, 2019.

[26] Jiali Li, George Bosilca, Aurelien Bouteiller, and Bogdan Nicolae, "Elastic deep learning through resilient collective operations," in *Proceedings of the SC'23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 44–50.

[27] Keita Teranishi and Michael A Heroux, "Toward local failure local recovery resilience model using mpi-ulfm," in *Proceedings of the 21st european mpi users' group meeting*, 2014, pp. 51–56.

[28] Nuria Losada, Patricia González, María J Martín, George Bosilca, Aurélien Bouteiller, and Keita Teranishi, "Fault tolerance of mpi applications in exascale systems: The ulfm solution," *Future Generation Computer Systems*, vol. 106, pp. 467–481, 2020.

[29] William Gropp, "Mpich2: A new start for mpi implementations," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 2002, pp. 7–7.

[30] Isaías Comprés, Ao Mo-Hellenbrand, Michael Gerndt, and Hans-Joachim Bungartz, "Infrastructure and api extensions for elastic execution of mpi applications," in *Proceedings of the 23rd European MPI Users' Group Meeting*, 2016, pp. 82–97.

[31] Gonzalo Martin, Maria-Cristina Marinescu, David E Singh, and Jesus Carretero, "Flex-mpi: an mpi extension for supporting dynamic load balancing on heterogeneous non-dedicated systems," in *European Conference on Parallel Processing*. Springer, 2013, pp. 138–149.

[32] Sergio Iserte Agut and Antonio José Peña Monferrer, *High-throughput Computation through Efficient Resource Management*, Ph.D. thesis, Ph. D. Thesis, Universitat Jaume I, 2018. Accepted: 2018-12-05T10: 00: 25Z . . . , 2018.

[33] Andy B Yoo, Morris A Jette, and Mark Grondona, "Slurm: Simple linux utility for resource management," in *Workshop on job scheduling strategies for parallel processing*. Springer, 2003, pp. 44–60.

[34] Morris A Jette and Tim Wickberg, "Architecture of the slurm workload manager," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2023, pp. 3–23.

[35] Sergio Iserte and Krzysztof Rojek, "An study of the effect of process malleability in the energy efficiency on gpu-based clusters: S. iserte, k. rojek," *The Journal of Supercomputing*, vol. 76, no. 1, pp. 255–274, 2020.

[36] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier, "Starpu: a unified platform for task scheduling on heterogeneous multicore architectures," in *European Conference on Parallel Processing*. Springer, 2009, pp. 863–874.

[37] George Bosilca, Aurelien Bouteiller, Anthony Danalis, Mathieu Faverge, Thomas Hérault, and Jack J Dongarra, "Parsec: Exploiting heterogeneity to enhance scalability," *Computing in Science & Engineering*, vol. 15, no. 6, pp. 36–45, 2013.

[38] Ziyang Liu, Renyu Yang, Jin Ouyang, Weihan Jiang, Tianyu Ye, Menghao Zhang, Sui Huang, Jiaming Huang, Chengru Song, Di Zhang, Tianyu Wo, and Chunming Hu, "Kale: Elastic gpu scheduling for online dl model training," in *Proceedings of the 2024 ACM Symposium on Cloud Computing*, New York, NY, USA, 2024, SoCC '24, p. 36–51, Association for Computing Machinery.

[39] Dong H Ahn, Jim Garlick, Mark Grondona, Don Lipari, Becky Springmeyer, and Martin Schulz, "Flux: A next-generation resource management framework for large hpc centers," in *2014 43rd International Conference on Parallel Processing Workshops*. IEEE, 2014, pp. 9–17.

[40] Dong H Ahn, Ned Bass, Albert Chu, Jim Garlick, Mark Grondona, Stephen Herbein, Helgi I Ingólfsson, Joseph Koning, Tapasya Patki, Thomas RW Scogland, et al., "Flux: Overcoming scheduling challenges for exascale workflows," *Future Generation Computer Systems*, vol. 110, pp. 202–213, 2020.

[41] Dick Buttlar, Jacqueline Farrell, and Bradford Nichols, *Pthreads programming: A POSIX standard for better multiprocessing*, .ºO'Reilly Media, Inc.", 1996.

[42] Rolf Neugebauer, Gianni Antichi, José Fernando Zazo, Yury Audzevich, Sergio López-Buedo, and Andrew W Moore, "Understanding pcie performance for end host networking," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 327–341.

[43] Jose I Aliaga, Maribel Castillo, Sergio Iserte, Iker Martín-Álvarez, and Rafael Mayo, "A survey on malleability solutions for high-performance distributed computing," *Applied Sciences*, vol. 12, no. 10, pp. 5231, 2022.

[44] Andy B Yoo, Morris A Jette, and Mark Grondona, "Slurm: Simple linux utility for resource management," in *Workshop on job scheduling strategies for parallel processing*. Springer, 2003, pp. 44–60.

[45] Javier Bueno, Judit Planas, Alejandro Duran, Rosa M Badia, Xavier Martorell, Eduard Ayguade, and Jesús Labarta, "Productive programming of gpu clusters with ompss," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE, 2012, pp. 557–568.

[46] Xueze Kang, Guangyu Xiang, Yuxin Wang, Hao Zhang, Yuchu Fang, Yuhang Zhou, Zhenheng Tang, Youhui Lv, Eliran Maman, Mark Wasserman, et al., "Elaswave: An elastic-native system for scalable hybrid-parallel training," *arXiv preprint arXiv:2510.00606*, 2025.

Poda estructurada en BERT: Un estudio de rendimiento y consumo en el Borde

Nicolás Hernández[1], Pedro Toledo[1] Vicente Blanco[1] y Francisco Almeida [1]

Resumen— **Los modelos de lenguaje basados en Transformadores, especialmente BERT, han revolucionado el procesamiento del lenguaje natural. Sin embargo, su elevado costo computacional limita su despliegue en dispositivos de borde con restricciones de memoria, latencia y energía. Con el objetivo de optimizar el despliegue de modelos en dispositivos con recursos limitados, este trabajo presenta un análisis comparativo exhaustivo de tres técnicas de poda estructurada (poda 2:4, *Attention Head Pruning* y *FFN Width Pruning*) aplicadas a tres variantes de BERT (base, large, DistilBERT). Utilizamos TensorRT para optimizar la inferencia aprovechando los *sparse tensor cores* nativos de hardware moderno. Los experimentos se realizan sobre WikiText-2 usando una Jetson AGX Orin como plataforma de borde representativa, midiendo precisión mediante pseudo-perplexity, rendimiento a través de latencia con distintos tamaño de lote, y eficiencia energética en términos de energía neta por inferencia. *FFN Width Pruning* logra la mayor reducción de latencia, alcanzando 10.67 % en BERT-large, mientras que *Attention Head Pruning* preserva la precisión con degradación mínima de pseudo-perplexity. La poda 2:4 demuestra ser la técnica más eficiente energéticamente, alcanzando hasta 46 muestras por joule en DistilBERT. Los resultados demuestran que la poda estructurada es viable en dispositivos de borde, permitiendo aceleraciones significativas con balances controlados entre precisión, latencia y consumo energético. Este trabajo contribuye a reducir la distancia entre investigación de compresión de modelos y despliegue práctico en sistemas de borde con restricciones reales.**

Palabras clave— **BERT, Poda estructurada, Computación en el borde, Eficiencia energética, TensorRT.**

I. Introducción

LA s redes de lenguaje profundas basadas en arquitectura Transformador, particularmente BERT, han establecido nuevos estándares de rendimiento en tareas de procesamiento del lenguaje natural (PLN) [1]. Sin embargo, estos modelos presentan un desafío crítico para su despliegue en entornos de recursos limitados: sus millones de parámetros (110 millones en BERT-base, 340 millones en BERT-large) requieren altas demandas de memoria, potencia computacional y energía [2].

El paradigma de la computación en el borde emerge como solución estratégica para desplegar inteligencia artificial distribuida, evitando los costos de comunicación y latencia inherentes al procesamiento centralizado en la nube. Dispositivos como Jetson AGX Orin representan una nueva generación de aceleradores de borde con capacidades de tensor core especializadas para cálculos de matriz de alto rendimiento [3]. Sin embargo, ejecutar modelos PLN de tamaño completo en estas plataformas sigue siendo inviable.

Entre las diversas técnicas de compresión disponibles (destilación de conocimiento, cuantización, arquitecturas eficientes), el *pruning* o poda de conexiones destaca por su simplicidad de implementación y compatibilidad universal [4]. La poda estructurado, que elimina componentes enteros (canales, cabezas de atención, capas intermedias), ofrece beneficios particulares: (i) genera redes que permiten aceleración real en hardware estándar, (ii) reduce tanto el tamaño del modelo como la complejidad computacional, y (iii) no requiere hardware especializado para inferencia acelerada, a diferencia de técnicas basadas en cuantización de baja precisión.

Específicamente, tres enfoques de poda estructurada son particularmente relevantes para Transformadores. La poda 2:4, que respeta patrones específicos de dispersidad aprovechando los *sparse tensor cores* nativos de GPUs modernas [5], ha demostrado viabilidad en plataformas de alto rendimiento pero su efectividad en dispositivos de borde permanece subexplorada. El *Attention Head Pruning*, que elimina cabezas redundantes del mecanismo multi-head [6], ha sido ampliamente estudiado en BERT con resultados prometedores [7]. Finalmente, el *Feed-Forward Network (FFN) Width Pruning*, que reduce las dimensiones intermedias de las capas feed-forward [8], ofrece un potencial significativo pero requiere validación experimental sistemática en combinación con otros métodos en hardware específico de borde.

La literatura actual presenta varios huecos críticos. Primero, falta un estudio comparativo integral de estos tres métodos de poda aplicados conjuntamente a las variantes BERT más comúnmente desplegadas (base, large, distilado). Segundo, aunque la cuantización ha sido extensamente evaluada en el borde [9], los estudios de poda en dispositivos de borde como Jetson son limitados, tal como se aborda en trabajos del grupo de investigación [10]. Tercero, las medidas de eficiencia energética (consumo real de potencia durante inferencia) se reportan raramente, limitando la comprensión del impacto verdadero de la compresión en sostenibilidad [11]. Finalmente, la optimización mediante TensorRT en plataformas Jetson, particularmente aprovechando *sparse tensor cores* para poda 2:4, no ha sido sistemáticamente evaluada.

Las principales contribuciones de este trabajo son las siguientes. Primero, presentamos una evaluación experimental exhaustiva de tres técnicas de poda estructurada aplicadas a modelos BERT desplegados en una plataforma de borde representativa (Jetson

[1]HPC Group, Escuela Superior de Ingeniería y Tecnología, Universidad de La Laguna, e-mail: nhernang@ull.es, falmeida@ull.es, vblanco@ull.es.

AGX Orin), midiendo de forma integral el impacto en precisión, latencia y eficiencia energética. Segundo, analizamos el comportamiento comparativo de poda 2:4, *Attention Head Pruning* y *FFN Width Pruning* a través de múltiples variantes de modelos (base, large, DistilBERT) y configuraciones de tamaño de lote, identificando patrones de escalabilidad y trade-offs entre técnicas. Tercero, proporcionamos una metodología experimental detallada y reproducible para evaluar técnicas de poda en dispositivos de borde, incluyendo mediciones de energía neta y eficiencia relativa, que permite análisis sistemático de los compromisos rendimiento-energía y ofrece recomendaciones prácticas para seleccionar estrategias de despliegue en entornos de borde con restricciones reales.

Este artículo se estructura de la siguiente manera. La sección 2 presenta el estado del arte en compresión de Transformadoress, con énfasis en técnicas de poda. La sección 3 detalla la metodología experimental, incluyendo descripción de métodos, modelos, dataset y protocolo de medición. La sección 4 expone los resultados de experimentos comparativos en términos de perplexity, latencia y consumo energético. Finalmente, la sección 5 sintetiza hallazgos principales y discute trabajo futuro.

II. Estado del arte

A. Compresión de modelos Transformadores: visión general

Los modelos basados en arquitectura Transformador han revolucionado el procesamiento del lenguaje natural, logrando resultados sobresalientes en tareas como traducción, clasificación y generación de texto [1], [12]. Sin embargo, esta capacidad expresiva conlleva un costo computacional significativo. BERT-base contiene 110 millones de parámetros y requiere cientos de millones de operaciones por inferencia, mientras que BERT-large alcanza los 340 millones de parámetros. Esta complejidad limita severamente su despliegue en dispositivos con recursos restringidos [2].

La compresión de modelos emerge como estrategia fundamental para hacer viables estos modelos en escenarios de borde. Las técnicas principales en la literatura incluyen: destilación de conocimiento (transferencia de conocimiento de un modelo grande a uno pequeño), cuantización (reducción de precisión de pesos y activaciones), arquitecturas eficientes (diseño de modelos intrinsecamente ligeros), y poda de conexiones. La poda estructurado, que elimina componentes enteros de la red (capas, cabezas de atención, filtros), destaca particularmente porque: (i) genera redes que permiten aceleración real en hardware estándar sin requerir kernels especializados, (ii) reduce simultáneamente el tamaño del modelo y la complejidad computacional, y (iii) es agnóstico respecto al framework de deep learning utilizado [13].

B. poda 2:4 (dispersión estructurada N:M)

La dispersión 2:4 representa un patrón estructurado donde exactamente 2 de cada 4 pesos consecutivos en una fila de una matriz son cero. Este patrón fue introducido por NVIDIA con la arquitectura Ampere y sus *Sparse Tensor Cores* [14], que proporcionan hardware especializado capaz de duplicar el throughput en operaciones de multiplicación de matrices cuando los pesos respetan este patrón de dispersidad.

En [15] se introduce un workflow de tres pasos (entrenar denso, podar a 2:4, reentrenar con máscara fija) con aceleraciones de aproximadamente 2× en GEMMs grandes mediante las librerías cuSPARSELt de NVIDIA, siendo los experimentos incluidos BERT en SQuAD y redes convolucionales en ImageNet. Para entrenar redes N:M desde cero sin preentrenamiento denso, se introduce SR-STE (Sparse-Refined Straight-Through Estimator), logrando 2× de aceleración en A100 con pérdida de precisión negligible en clasificación de imágenes, detección de objetos y traducción automática [16]. Posteriormente, se formula la dispersión N:M como un problema de optimización con restricciones resuelto mediante ADMM, aplicándolo específicamente a BERT-base en el benchmark GLUE completo, donde supera métodos de poda por magnitud por aproximadamente 1.7 puntos en score promedio [17].

Los avances recientes en métodos de entrenamiento 2:4 incluyen técnicas que identifican fallos de SR-STE cuando se usa con Adam (optimizador estándar en Transformadores) y proponen AutoSwitch con fases separadas de precondición y aprendizaje de máscara [18]. Asimismo, se introduce una función de poda continua para resolver la discontinuidad inherente al *Straight-Through Estimator* [19]. De manera similar, se modela la selección de máscaras N:M como aprendizaje de distribución con Gumbel Softmax, logrando 6.72 perplexity en WikiText-2 para LLaMA-2-7B con 2:4 (comparado con 5.12 denso), superando ampliamente métodos post-entrenamiento [20].

En el contexto de Transformadores de lenguaje, se demuestra aceleración end-to-end durante preentrenamiento, siendo el primer trabajo en reportar aceleraciones en forward y backward pass aprovechando *Sparse Tensor Cores* 2:4 [21]. Para aplicaciones a gran escala, se presenta un método de segundo orden que poda modelos de 175 mil millones de parámetros al 50 % de dispersión sin reentrenamiento en menos de 4.5 horas en una A100 única [22].

A nivel de despliegue, el ecosistema TensorRT de NVIDIA ha madurado significativamente, permitiendo BERT-Large con latencia de 1.2 milisegundos [23]. La combinación de dispersión 2:4 con INT8 cuantizado logra 1.4× de aceleración sobre dense-INT8 en TensorRT con pérdida de precisión mínimas [24]. A nivel de despliegue en plataformas de borde, se evalúa recientemente la viabilidad de la poda 2:4 en Jetson AGX Orin mediante análisis integral de latencia, energía y precisión, demostrando la efecti-

vidad de esta técnica en dispositivos con restricciones energéticas severas cuando se optimiza adecuadamente con TensorRT [25].

C. Attention Head Pruning

El mecanismo *multi-head attention* es un componente central de los Transformadores, donde la representación se divide en H cabezas que operan en paralelo. Un hallazgo temprano y fundamental es que no todas las cabezas contribuyen equitativamente al desempeño, demostrándose que 20-40 % de las cabezas de BERT pueden eliminarse en tareas como MNLI con pérdida mínimas, siendo algunos análisis posteriores reveladores de que capas enteras pueden reducirse a una sola cabeza de atención [6].

Se complementa este análisis identificando que las cabezas tienen roles especializados (posicionales, sintácticas, de palabras raras) mediante regularización L0 con puertas estocásticas, mostrando que estas cabezas especializadas tienden a ser las últimas en eliminarse durante poda, sugiriendo que contienen información irreducible [26].

Bajo la lente de la hipótesis del *lottery ticket*, se demuestra que para BERT, todas las subredes estructuradas resultantes de poda de cabezas mantienen alta entrenabilidad incluso después de podado agresivo, hallazgo contrastante con lo observado en redes convolucionales que sugiere propiedades particulares de redundancia en Transformadores pre-entrenados [27].

Se introduce poda por subconjuntos diferenciable, aprendiendo variables de importancia por cabeza con una restricción exacta sobre el número de cabezas retenidas, mejorando sobre métodos anteriores en términos de control de dispersión [28]. De manera complementaria, se adoptan valores de Shapley como criterio de importancia, particularmente efectivo para detectar interferencia entre cabezas en modelos multilingües [29].

D. FFN Width Pruning (poda de dimensiones intermedias)

Las capas feed-forward (FFN) en Transformadores consisten en dos transformaciones lineales con una capa de activación no-lineal entre ellas. La dimensión intermedia (típicamente 4 veces la dimensión oculta) introduce redundancia frecuentemente no explorada hasta hace poco.

Se introduce un framework unificado que aprende conjuntamente cinco máscaras: capas FFN completas, dimensiones intermedias FFN, capas MHA completas, cabezas individuales y dimensiones ocultas, usando regularización L0 con multiplicador Lagrangiano combinado con destilación de conocimiento. A 95 % de dispersión en BERT-base, logra un modelo 12× más pequeño y 2-3× más rápido con menos de 2 % de caída en GLUE y SQuAD, reportando aceleraciones reales en CPU. Un hallazgo crucial es que en hardware de CPU, las capas FFN se convierten en cuello de botella y tienden a podarse más agresivamente que las capas MHA [30].

Se presenta un modelo BERT dinámico que ejecuta sub-redes con ancho adaptativo (número de cabezas y neuronas FFN) y profundidad variables, siendo los puntuaciones de importancia los que determinan qué componentes retener. A tamaños menores, este enfoque supera consistentemente a TinyBERT y DistilBERT en GLUE y SQuAD, demostrando que la adaptación de ancho es más robusta a compresión agresiva que la reducción de profundidad [31].

Se extiende poda estructurada a modelos generativos (BART) con énfasis en dimensiones intermedias FFN para mantener calidad de generación [8]. A nivel de LLMs, se poda LLaMA2-7B a 1.3B y 2.7B eliminando capas, cabezas y dimensiones intermedias FFN de forma conjunta, logrando modelos que superan a Pythia, OpenLLaMA y TinyLLaMA con solo el 3 % del cómputo de pre-entrenamiento desde cero [32].

E. Eficiencia energética y computación en el borde

Más allá de métricas de latencia y tamaño de modelo, la eficiencia energética es cada vez más crítica dado el costo ambiental del aprendizaje automático a escala. Se cuantifican por primera vez los costos energéticos reales del entrenamiento de modelos NLP, catalizando el movimiento de Green AI [2]. Para inferencia, se co-diseña algoritmo y hardware, combinando salida temprana adaptativa, poda selectiva y cuantización FP8 con control de frecuencia dinámico (DVFS) a nivel de oración, logrando hasta 7× de reducción en energía por inferencia en acelerador personalizados [33].

Se investiga cómo la configuración de hardware (frecuencia de GPU, memoria, CPU) afecta el consumo energético en dispositivos de borde específicamente. Probado en Jetson TX2 y Jetson Orin con BERT-base, se demuestra hasta 36 % de ahorro energético manteniendo SLOs de latencia, con un hallazgo clave revelando que BERT es memory-bound en el borde y se beneficia más del ajuste de frecuencia de memoria que de CPU [34].

Estudios recientes de consumo energético integral incluyen un benchmark estandarizado con 1,841 mediciones de 60 sistemas incluyendo Jetson AGX Orin [35], y la evaluación de 28 LLMs cuantizados en Raspberry Pi 4 usando medidor hardware de precisión [36]. Un hallazgo consistente es que la combinación poda + cuantización reduce consumo energético entre 30 % y 70 % respecto a modelos densos, aunque se necesita más trabajo específico combinando poda estructurada de Transformadores con medición rigurosa de energía en plataformas Jetson.

F. Evaluación comparativa de técnicas de poda

La literatura ha establecido que existe un balance fundamental entre dispersión de pesos lograda y aceleración hardware obtenido. La poda no-estructurada (eliminación de pesos individuales) puede alcanzar niveles de dispersión muy altas (70-90 %) con pérdidas de precisión mínimas, pero estos patrones de pesos dispersos raramente se traducen en

aceleración en hardware estándar. La poda estructurada elimina componentes enteros (cabezas, filtros, neuronas FFN, capas) y genera modelos que corren significativamente más rápido en CPU/GPU sin requerir kernels especializados.

Un meta-análisis de 81 papers de poda subraya la ausencia de benchmarks estandarizados y las dificultades de comparación reproducible [37]. El estado del arte actual en poda estructurada con garantías de inferencia remueve iterativamente componentes con peor compromiso pérdida-tiempo de ejecución, superando todos los métodos previos de poda y destilación (CoFi, MiniLM, TinyBERT) en BERT-base, e igualando la compresión de MobileBERT simplemente podando BERT-large, logrando aceleraciones de hasta 15× con caídas pequeñas en GLUE y SQuAD [38].

Análisis recientes comparan el impacto de la poda, la destilación y la cuantización en la eficiencia energética. Se ha hallado que BERT, al combinar poda y destilación, logra una reducción del 32.1 % en el consumo de energía, mientras que TinyBERT alcanza una mejora del 91.26 % respecto a la versión completa de BERT [39]. La tendencia observada es que métodos híbridos que combinan múltiples técnicas (poda + destilación + cuantización) generan mejores resultados que técnicas aisladas.

Para modelos Transformadores pre-entrenados como BERT, se ha demostrado que es posible retener el 95 % de la precisión manteniendo únicamente el 5 % de los parámetros en el dataset MNLI, empleando una poda basada en la magnitud de los cambios de los pesos durante el ajuste fino (*fine-tuning*) [40]. Asimismo, se ha identificado la existencia de subredes ganadoras (*matching subnetworks*) —configuraciones que, tras ser entrenadas desde su inicio, igualan el rendimiento del modelo completo— con niveles de dispersión del 40–90 % que mantienen su capacidad de generalización en tareas de transferencia (*downstream tasks*) [41]. De manera complementaria, se ha logrado una aceleración de 10× en CPU con una degradación inferior al 2 % mediante la integración con entornos de ejecución optimizados como DeepSparse y HuggingFace [42].

A nivel de implementación práctica, la maduración de ecosistemas como TensorRT ha sido crucial. Se demuestra viabilidad de 2:4 poda en plataformas de borde (Jetson Orin), mientras que otros trabajos establecen metodología para medición conjunta de latencia y energía en dispositivos embebidos, sentando precedentes para evaluación rigurosa en el borde que este trabajo extiende a modelos BERT y Transformadores [43], [9].

III. Metodología

A. Métodos de poda evaluados

Se evaluaron tres técnicas de poda estructurada aplicadas de forma **one-shot sin reentrenamiento**, junto con el modelo denso como baseline.

poda 2:4 (dispersión N:M). Aplicado al patrón 2:4 nativo soportado por *sparse tensor cores*. Para cada fila de pesos, se seleccionan los 2 de cada 4 pesos consecutivos con mayor magnitud absoluta, fijando el resto a cero.

Attention Head Pruning. En BERT-base (12 cabezas × 64 dimensiones), se eliminan 3 cabezas por capa (25 % de reducción). Las cabezas se ordenan por norma L1 de sus pesos y se descartan las de menor norma, una estrategia directa y consistente con trabajos previos [6].

FFN Width Pruning. Se podan las neuronas de la capa `intermediate.dense` (filas) y las correspondientes en `output.dense` (columnas) de forma coordinada, reduciendo la dimensión intermedia en un 25 %.

B. Modelos y dataset

Se evaluaron tres variantes de BERT en fp16:

- **BERT-base**: 12 capas, 768 dimensión oculta, 110M parámetros.
- **BERT-large**: 24 capas, 1024 dimensión oculta, 340M parámetros.
- **DistilBERT-base**: 6 capas, 768 dimensión oculta, 66M parámetros (referencia comprimida).

El dataset de evaluación es **WikiText-2**, evaluando pseudo-perplexity en el split de test mediante masked language modeling (MLM). Para modelos enmascarados, cada token se enmascara individualmente para calcular $-\log P(\text{token} \mid \text{contexto})$; la pseudo-perplexity se define como [44]:

La pérdida por cada token i se define como $\mathcal{L}_\rangle = -\log \mathcal{P}(\text{token}_\rangle \mid \text{ctx}_\rangle)$. Entonces, la Pseudo-Perplexity se calcula como:

$$\text{P-PPL} = \exp\left(\frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_\rangle\right) \tag{1}$$

donde N es el número total de tokens en la fracción evaluada del set de test.

C. Plataforma de despliegue

Las mediciones de latencia y energía se realizan en **Jetson AGX Orin** (ARM64 + JetPack), desplegando los modelos mediante **TensorRT** para aprovechar los *sparse tensor cores* nativos. La configuración de precisión es fp16. Se evalúan siete tamaños de lote: [1, 2, 4, 8, 16, 32, 64].

Nota de implementación: Se optó por TensorRT en lugar de cuSPARSELt porque la compilación de PyTorch para JetPack no incluye cuSPARSELt enlazado, y pruebas previas con CUTLASS disperso en modo *eager* revelaron que el overhead de sincronización dominaba el ahorro computacional de los *sparse tensor cores*.

D. Métricas y protocolo de medición

D.1 Latencia

Se mide como latencia promedio de 200 inferencias consecutivas mediante CUDA Events, descartan-

do 30 iteraciones de warmup. Se reporta en milisegundos. El throughput (muestras/segundo) se deriva como:

$$\text{Throughput} = \frac{\text{tamaño_lote} \times 200}{\text{tiempo_total_ms}} \quad (2)$$

Se realiza una única medición por combinación de modelo, técnica de poda y tamaño de lote (varianza $< 1\,\%$).

D.2 Energía

Se mide el consumo energético neto (activo menos idle) en julios usando un script que captura valores de voltaje y corriente de los sensores INA3221 en canales GPU/SoC y CPU a 30 Hz. El consumo idle se estima midiendo durante 5 segundos en reposo (10 muestras a 30 Hz) para cada modelo. Cada combinación se repite en 3 ejecuciones independientes con 20 iteraciones de warmup térmico previo. Se reportan media, desviación estándar, mín y máx. Se registran temperaturas CPU al inicio y final para detectar throttling.

D.3 Precisión (Pseudo-Perplexity)

Se evalúa pseudo-perplexity (definida en la Sección 3.2) una única vez por modelo y técnica en fp16 nativo de PyTorch (sin TensorRT) en modo inferencia. Dado el costo computacional en Jetson AGX Orin ($O(n)$ propagaciones hacia adelante por secuencia), se evalúa sobre una fracción del dataset de test de WikiText-2.

E. Diseño experimental

Se analizaron cuatro variantes para cada modelo base (Baseline, poda 2:4, poda de cabezas de atención y de capas FFN). Para garantizar la robustez de los resultados de eficiencia, se adoptó el siguiente protocolo:

- **Inferencia:** Las métricas se evaluaron sobre siete tamaños de lote distintos para observar el escalado en el borde.
- **Energía:** Debido a la variabilidad térmica y de consumo, se realizaron tres réplicas por configuración, reportando el valor promedio.
- **Latencia y Perplexity:** Se realizaron mediciones sistemáticas para cada variante y configuración de tamaño de lote.

IV. Resultados experimentales

Se presentan los resultados de la evaluación experimental de tres técnicas de poda estructurada (2:4, *Attention Head Pruning* y *FFN Width Pruning*) aplicadas a tres variantes de BERT (base, large, DistilBERT-base) desplegadas en Jetson AGX Orin. Los hallazgos se organizan según tres dimensiones de análisis: precisión del modelo (medida mediante pseudo-perplexity en WikiText-2), rendimiento de inferencia (latencia y throughput) y eficiencia energética. El objetivo es evaluar el trade-off entre compresión lograda, preservación de precisión y ganancia real en velocidad y consumo energético en un dispositivo de borde representativo.

A. Degradación de precisión (Pseudo-Perplexity)

La pseudo-perplexity en modelos MLM es una métrica sensible a la capacidad del modelo para predecir tokens enmascarados. Aplicar poda estructurada introduce redundancia y reduce la capacidad expresiva, impactando directamente en esta métrica. Se evaluó el aumento relativo de pseudo-perplexity respecto al modelo baseline denso para cada técnica de poda y cada variante de BERT.

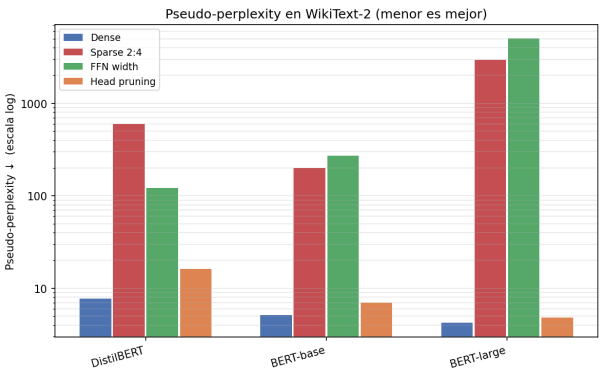


Fig. 1: Comparativa de pseudo-perplexity: BERT baseline vs técnicas de poda estructurada.

Los resultados ilustrados en la Figura 1 revelan diferencias significativas en cómo cada técnica afecta la precisión. La poda 2:4 y *FFN Width Pruning* muestran degradación sustancial, particularmente evidente en BERT-large, donde ambas técnicas causan aumentos notables en pseudo-perplexity. En contraste, *Attention Head Pruning* mantiene valores de pseudo-perplexity muy cercanos al baseline, demostrando que muchas cabezas de atención son redundantes y su eliminación impacta mínimamente en la capacidad predictiva del modelo. Este hallazgo es consistente con trabajos previos [6] que identifican cabezas redundantes en Transformadores.

B. Latencia

La latencia de inferencia es crítica para aplicaciones en el borde en tiempo real. Se midió el tiempo de ejecución end-to-end en Jetson AGX Orin para cada modelo y técnica de poda a distintos tamaños de lote. El análisis se presenta en dos perspectivas: comparativa directa de latencia absoluta a tamaño de lote fijo, y análisis de escalabilidad mediante una aceleración relativo al baseline denso.

Con un tamaño de lote fijo de 8 (Figura 2), *FFN Width Pruning* emerge como la técnica más efectiva en reducción de latencia. Este beneficio se acentúa conforme aumenta el tamaño del modelo: la reducción es modesta en DistilBERT-base, más pronunciada en BERT-base, y máxima en BERT-large. Este patrón refleja que la dimensión intermedia de las capas FFN representa una fracción mayor del cómputo en modelos más grandes, por lo que su eliminación genera mayor ahorro.

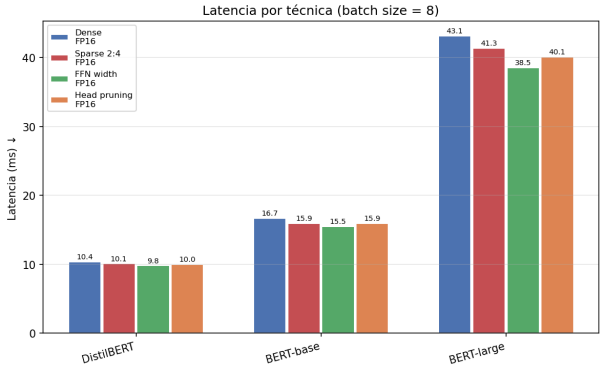


Fig. 2: Latencia absoluta a tamaño de lote 8: comparativa de técnicas de poda.

La Figura 3 muestra que todos los modelos y técnicas exhiben una aceleración máxima en tamaño de lote 1, con caída a lote de 2 y estabilización a partir de lote de 4. La poda 2:4 muestra ascenso posterior en lotes mayores, particularmente pronunciado en BERT-base y DistilBERT-base, mientras que en BERT-large el crecimiento es mínimo. Para *Attention Head Pruning* y *FFN Width Pruning*, la mejora al aumentar el tamaño de lote se observa solo en BERT-base; en BERT-large ambas técnicas muestran variabilidad marginal, indicando que el tamaño del modelo domina sobre el efecto del tamaño de lote en la efectividad de la poda.

C. Eficiencia energética

La eficiencia energética es crítica en dispositivos de borde con presupuestos de potencia limitados. Se evaluó tanto el consumo energético neto (activo menos *idle*) por inferencia como la métrica de eficiencia relativa (muestras por joule), que captura la productividad energética de cada técnica.

El consumo energético neto (Figura 4) sigue un patrón similar al de latencia, con reducciones mayores conforme aumenta el tamaño del modelo. Un hallazgo destacable es que la poda 2:4, aunque no es la técnica más rápida, es la más eficiente energéticamente en BERT-base y DistilBERT-base. En BERT-large, *FFN Width Pruning* lidera ligeramente, lo que se explica por su mayor reducción de latencia en ese modelo específico.

La métrica de eficiencia relativa (Figura 5) corrobora estos hallazgos: todas las técnicas mejoran la eficiencia energética respecto al baseline, con mejoras más pronunciadas en modelos más pequeños. La poda 2:4 se muestra como la técnica más eficiente en términos de muestras por joule, excepto en BERT-large donde es superado por poco por *FFN Width Pruning*. Este resultado sugiere que 2:4 ofrece un balance superior entre reducción de latencia y consumo energético, particularmente relevante para dispositivos de borde donde ambas métricas son críticas.

D. Tamaño de modelo

El tamaño en disco de los modelos es una métrica relevante para despliegue en dispositivos de borde con restricciones de almacenamiento. Se midió el tamaño de los archivos serializados en formato *SafeTensors* fp16 para cada variante de BERT y técnica de poda.

Los resultados (Figura 6) revelan diferencias significativas en el impacto de cada técnica sobre el tamaño del modelo. La poda 2:4 no reduce el tamaño en disco, ya que los pesos se ponen a cero sin eliminar la estructura subyacente de la matriz. En contraste, *Attention Head Pruning* logra reducciones moderadas de 4-7 %, mientras que *FFN Width Pruning* produce reducciones más sustanciales de 8-14 %. Notablemente, la reducción de tamaño aumenta con el tamaño del modelo base: en DistilBERT-base los ahorros son menores, mientras que en BERT-large son máximos. Este patrón refleja que técnicas que eliminan estructuras completas (cabezas, neuronas intermedias) reducen proporcionalmente más el tamaño en modelos grandes. Estos resultados subrayan que aunque la poda 2:4 es eficiente energéticamente, no proporciona beneficio de compresión de almacenamiento, haciendo técnicas como *FFN Width Pruning* más atractivas para escenarios donde el footprint de disco es crítico.

Un aspecto a considerar es que *SafeTensors*, el formato utilizado para serializar los modelos, almacena todos los valores de peso sin optimizaciones para dispersión. Por tanto, aunque la poda 2:4 introduce abundancia de ceros en la matriz de pesos, éstos ocupan igual espacio en disco que los valores originales. Sin embargo, formatos con representación dispersa nativa (ej. CSR) o compresión estándar (gzip, bzip2) explotarían esta dispersión, produciendo reducciones significativas de tamaño relevantes para despliegue en el borde.

V. Conclusiones

Este trabajo evalúa tres técnicas de poda estructurada (2:4, *Attention Head Pruning* y *FFN Width Pruning*) aplicadas a modelos BERT desplegados en Jetson AGX Orin. Los resultados demuestran que todas las técnicas mejoran simultáneamente latencia y eficiencia energética sin comprometer significativamente la precisión. *Attention Head Pruning* ofrece el mejor balance entre preservación de precisión y compresión, manteniendo pseudo-perplexity muy cercana al baseline. *FFN Width Pruning* logra la mayor reducción de latencia, particularmente en modelos grandes. La poda 2:4 emerge como la técnica más eficiente energéticamente en términos de muestras por joule, aunque su efectividad depende del tamaño del modelo y la configuración de tamaño de lote. El comportamiento de aceleración varía significativamente con el tamaño de lote, mostrando máximos en lotes de 1 y estabilización a partir de lotes de 4, con patrones distintos según la técnica. Estos hallazgos subrayan que la elección de técnica de poda debe considerar el compromiso entre precisión, latencia y energía según los requisitos específicos de la aplicación de borde.

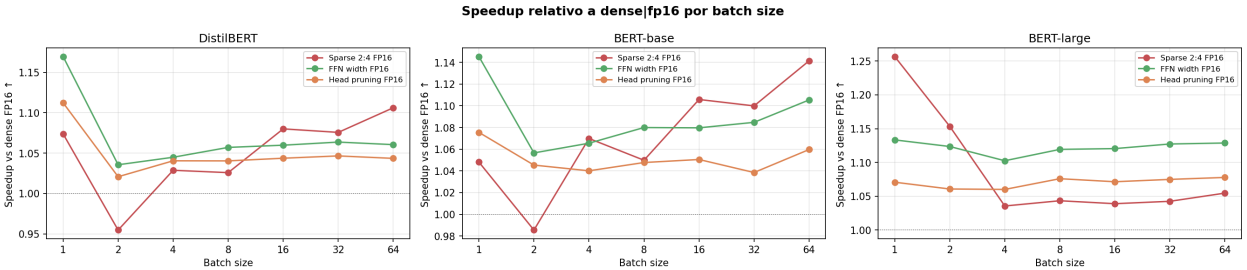


Fig. 3: Aceleración relativo al baseline denso en función del tamaño de lote.

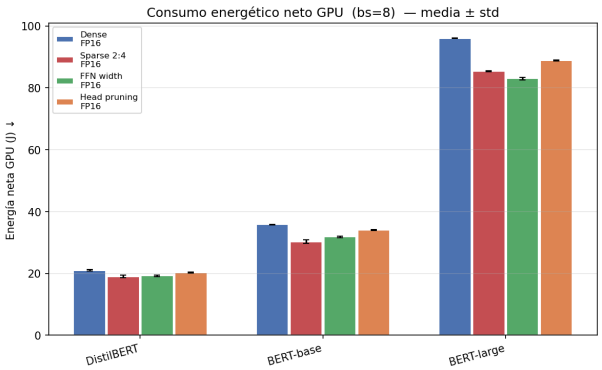


Fig. 4: Consumo energético neto por inferencia en julios para tamaño de lote 8.

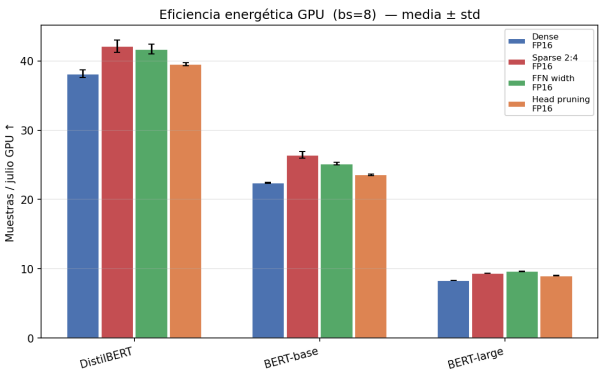


Fig. 5: Eficiencia energética (muestras por joule) para técnicas de poda.

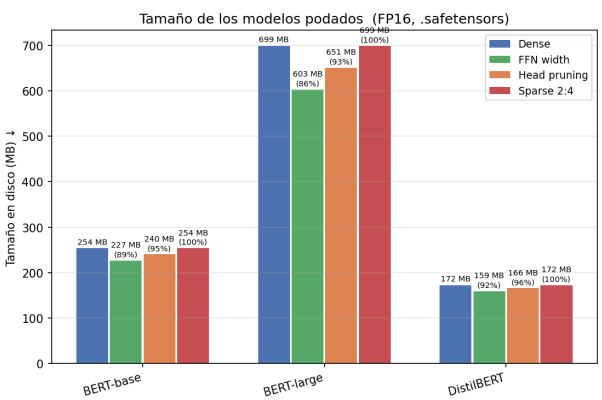


Fig. 6: Tamaño en disco (MB) de modelos baseline y podados, con porcentaje de reducción respecto al modelo denso.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia e Innovación de España a través del proyecto PID2023-151073NB-I00. Asimismo, ha sido cofinanciado por la Agencia Canaria de Investigación, Innovación y Sociedad de la Información del Departamento de Universidades, Ciencia, Innovación y Cultura, y por el Fondo Social Europeo Plus (FSE+) dentro del Programa Operativo Integrado de Canarias 2021-2027, Eje Prioritario 3, Tema 74 (85 %).

Referencias

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[2] Emma Strubell, Ananya Ganesh, and Andrew McCallum, "Energy and policy considerations for deep learning in NLP," *arXiv preprint arXiv:1910.09341*, 2019.

[3] NVIDIA, "Jetson AGX orin developer kit," 2023.

[4] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and David Grangier, "What's hidden in a randomly weighted neural network?," *arXiv preprint arXiv:2012.05897*, 2020.

[5] Scott Gray, Alec Radford, and Diederik P. Kingma, "GPU kernels for block-sparse weights," in *OpenAI Blog*, 2017.

[6] Paul Michel, Omer Levy, and Graham Neubig, "Are sixteen heads really better than one?," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019.

[7] Zhiqing Liu, Fei Li, Gang Li, and Jing Cheng, "EBERT: Efficient-yet-effective entity embeddings for BERT," in *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, 2021, pp. 4814–4823.

[8] Chaofan Tao, Lu Hou, Haoli Bai, and Jiansheng Wei, "Structured pruning for efficient generative pre-trained language models," *arXiv preprint arXiv:2306.14853*, 2023.

[9] S. Naveen and M. R. Kounte, "Optimizing convolutional neural networks for IoT devices: Performance and energy efficiency of quantization techniques," *The Journal of Supercomputing*, 2024.

[10] E. Diao, G. Wang, J. Zhan, Y. Yang, J. Ding, and V. Tarokh, "2:4 pruning on edge devices: Performance, energy efficiency and accuracy," in *Proceedings of Euro-Par 2024*, 2024, pp.–.

[11] David J. Green et al., "Carbontracker: Tracking and predicting the carbon footprint of training deep learning models," *arXiv preprint arXiv:2007.03051*, 2021.

[12] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever, "Language models are unsupervised multitask learners," 2019.

[13] Yu Cheng, Dingyi Wang, Pan Zhou, and Tao Zhang, "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations," *arXiv preprint arXiv:2308.06767*, 2024.

[14] NVIDIA, "Nvidia ampere architecture: In-depth," 2021.

[15] Deepak Mishra, Jeff Pool, and Paulius Micikevicius, "Accelerating sparse deep neural networks," *arXiv preprint arXiv:2104.08378*, 2021.

[16] Hao Zhou, Jing Miao, and Lei Ji, "Learning N:M fine-grained structured sparse neural networks from scratch," in *International Conference on Learning Representations*, 2021.

[17] Gregory Holmes, Hangfei Zhang, and Pan He, "NxM-transformer: Semi-structured sparsification for natural language understanding via ADMM," *arXiv preprint arXiv:2110.15766*, 2021.

[18] Xin Lu, Abhimanyu Agrawal, and Subu Subramanian, "STEP: Learning N:M structured sparsity masks from scratch with precondition," *Proceedings of the International Conference on Machine Learning*, 2023.

[19] Wei Hu, Yu Zhao, and Siyu Huang, "S-STE: Continuous pruning function for efficient 2:4 sparse pre-training," *arXiv preprint arXiv:2409.09099*, 2024.

[20] Gongfan Fang, Jie Yin, and Saurav Muralidharan, "MaskLLM: Learnable semi-structured sparsity for large language models," *arXiv preprint arXiv:2409.17481*, 2024.

[21] Wei Hu, Yu Zhao, and Siyu Huang, "Accelerating transformer pre-training with 2:4 sparsity," *arXiv preprint arXiv:2404.01847*, 2024.

[22] Elias Frantar and Dan Alistarh, "SparseGPT: Massive language models can be accurately pruned in one-shot," *arXiv preprint arXiv:2301.00774*, 2023.

[23] Jeff Pool, Aniruddha Sawarkar, and Rohit Rodge, "Accelerating inference with sparsity using the NVIDIA ampere architecture and NVIDIA TensorRT," in *NVIDIA Developer Blog*, 2021.

[24] NVIDIA, "Structured sparsity in the NVIDIA ampere architecture and applications in search engines," 2022.

[25] Nicolás Hernández, Pedro Toledo, Vicente Blanco, and Francisco Almeida, "2:4 pruning on edge devices: Performance, energy efficiency and accuracy," in *Euro-Par 2025: Parallel Processing*, Wolfgang E. Nagel, Diana Goehringer, and Pedro C. Diniz, Eds., Cham, 2026, pp. 278–291, Springer Nature Switzerland.

[26] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov, "Analyzing multi-head self-attention: Specialized heads do the heavy lifting," *arXiv preprint arXiv:1905.09418*, 2019.

[27] Sunil Prasanna, Anna Rogers, and Anna Rumshisky, "When BERT plays the lottery, all tickets are winning," *arXiv preprint arXiv:2005.00561*, 2020.

[28] Jiaoda Li, Ryan Cotterell, and Mrinmaya Sachan, "Differentiable subset pruning of transformer heads," *Transactions of the Association for Computational Linguistics*, 2023.

[29] William Held and Diyi Yang, "Shapley head pruning: Identifying and removing interference in multilingual transformers," *arXiv preprint arXiv:2210.05709*, 2022.

[30] Mengzhou Xia, Zexuan Zhong, and Dani Chen, "Structured pruning learns compact and accurate models," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, 2022.

[31] Lu Hou, Lifeng Huang, Lingxiao Shang, Xin Jiang, Xiao Liu, and Zhiyuan Wei, "DynaBERT: Dynamic BERT with adaptive width and depth," in *Advances in Neural Information Processing Systems*, 2020.

[32] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen, "Sheared LLaMA: Accelerating language model pre-training via structured pruning," *arXiv preprint arXiv:2310.06694*, 2023.

[33] Thierry Tambe, Xiaoning Xu, Akshay Menon, and Vikram S. Adve, "EdgeBERT: Sentence-level energy optimizations for latency-aware multi-task NLP inference," *arXiv preprint arXiv:2011.14203*, 2021.

[34] Ningning Yan, Wenqi Wang, and Shivaram Venkataraman, "PolyThrottle: Energy-efficient neural network inference on edge devices," *arXiv preprint arXiv:2310.19991*, 2023.

[35] David Tschand, Mitesh Daga, Wei Gao, and et al., "MLPerf power: Benchmarking the energy efficiency of machine learning systems," *arXiv preprint arXiv:2410.12032*, 2024.

[36] Eivind Husom and et al., "Sustainable LLM inference for edge AI," *arXiv preprint arXiv:2504.03360*, 2025.

[37] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and David Grangier, "What is the state of neural network pruning?," in *Proceedings of Machine Learning and Systems*, 2020.

[38] Eldar Kurtic, Daniel Campos, Hang Nguyen, Elias Frantar, and Dan Alistarh, "ZipLM: Inference-aware structured pruning of language models," *arXiv preprint arXiv:2302.04089*, 2023.

[39] João Paula, Isha Soni, and et al., "Comparative analysis of model compression techniques for carbon-efficient AI," *Scientific Reports*, vol. 15, 2025.

[40] Victor Sanh, Thomas Wolf, and Alexander M. Rush, "Movement pruning: Adaptive sparsity by fine-tuning," *arXiv preprint arXiv:2005.07683*, 2020.

[41] Tianlong Chen, Jonathan Frankle, Michail I. Chang, Sijia Liu, and Yang Zhang, "The lottery ticket hypothesis for pre-trained BERT networks," *arXiv preprint arXiv:2007.12223*, 2020.

[42] Eldar Kurtic, Daniel Campos, Hang Nguyen, Elias Frantar, and Dan Alistarh, "The optimal BERT surgeon: Scalable and accurate second-order pruning for large language models," *arXiv preprint arXiv:2203.07259*, 2022.

[43] E. Diao, G. Wang, J. Zhan, Y. Yang, J. Ding, and V. Tarokh, "2:4 pruning on edge devices: Performance, energy efficiency and accuracy," in *Proceedings of Euro-Par 2024*, 2024.

[44] Julian Salazar, Davis Liang, Toan Q. Nguyen, and Katrin Kirchhoff, "Masked language model scoring," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 2699–2712.

Generación sintética multilingüe para HTR: diseño de plataforma, escalabilidad y evaluación de prestaciones

Joaquim Oliva, Daniel Grao, Francesc Xavier Carrera, Francesc Giné, Vitor Luiz da Silva[1]

Resumen— **El reconocimiento de texto manuscrito (HTR) requiere conjuntos de datos amplios y bien anotados, un recurso todavía escaso en muchas lenguas y dominios. Este trabajo reposiciona un *framework* abierto de generación sintética multilingüe como una plataforma reproducible para soportar cargas de entrenamiento y *fine-tuning* en HTR. El sistema integra adquisición de texto, validación automática de tipografías, composición de fondos, perturbaciones controladas y metadatos estructurados, y ejecuta la generación en paralelo mediante `multiprocessing`. Además de validar la utilidad *downstream* sobre RIMES (francés) e IAM (inglés), caracterizamos el comportamiento computacional de la plataforma mediante experimentos de escalabilidad, coste de realismo visual, comparación multilingüe y análisis de etapas del *pipeline*. Los resultados muestran que el sistema ofrece *throughput* elevado y escalabilidad práctica en configuración multicore, mientras que la complejidad visual incrementa el coste temporal y de almacenamiento de forma medible. Como validación funcional, el *fine-tuning* mixto con datos reales y sintéticos reduce el CER entre un 50 % y un 58 % respecto al *baseline*, mientras que el ajuste exclusivamente sintético provoca *catastrophic forgetting*. Adicionalmente, un protocolo de *curriculum learning* con *learning rate* reducido sobre los mismos datos sintéticos alcanza 6,60 % CER en RIMES y 6,03 % CER en IAM, superando al *fine-tuning* exclusivamente real y demostrando que la plataforma soporta cargas de entrenamiento que extraen valor real de los datos generados. En conjunto, el trabajo posiciona la plataforma como una infraestructura reproducible y evaluable para cargas de Deep Learning en escenarios multilingües con baja disponibilidad de datos.**

Palabras clave— **HTR, datos sintéticos, evaluación de prestaciones, escalabilidad, paralelismo, multilingüe, Deep Learning.**

I. Introducción

El reconocimiento de texto manuscrito (*Handwritten Text Recognition*, HTR) ha avanzado rápidamente gracias a los modelos secuenciales y, más recientemente, a arquitecturas basadas en transformadores [1–4]. Sin embargo, el rendimiento sigue dependiendo de conjuntos de datos amplios, bien anotados y cercanos al dominio objetivo. Esta dependencia es especialmente problemática en lenguas con escasa disponibilidad de recursos, donde los corpus públicos son insuficientes para entrenar o adaptar modelos de forma robusta. Esta necesidad es especialmente acusada en lenguas minorizadas o con escasos recursos digitales —como el catalán, el euskera o el bretón, entre muchas otras—, para las que no existen corpus públicos de escritura manuscrita contemporánea.

Una práctica habitual consiste en preentrenar con *benchmarks* dominantes como IAM [7] o RIMES [8] y realizar posteriormente *fine-tuning* sobre pequeñas muestras reales del dominio final. Aunque este enfoque es útil, presupone acceso a datos anotados y no elimina la brecha de dominio producida por diferencias en estilo de escritura, soporte, calidad de captura o variabilidad léxica. La generación de datos sintéticos ofrece una alternativa complementaria [10, 11]: permite producir corpus controlados, ampliar la cobertura multilingüe, estudiar perturbaciones de forma aislada y preparar cargas de entrenamiento antes de disponer de grandes corpus reales.

En este trabajo presentamos una plataforma abierta para generación sintética multilingüe de escritura manuscrita orientada a HTR. A diferencia de una lectura puramente centrada en datasets, aquí enfatizamos cuatro ejes: (i) diseño de plataforma, (ii) paralelización de la generación, (iii) evaluación de prestaciones y (iv) utilidad práctica mediante experimentos de *fine-tuning* sobre *benchmarks* reales. El sistema integra adquisición de textos, validación de tipografías, composición de fondos, perturbaciones controladas y metadatos reproducibles, y ejecuta el proceso de síntesis en paralelo mediante `multiprocessing`. La validación funcional incluye tanto estrategias clásicas de *fine-tuning* como un protocolo de *curriculum learning* con *learning rate* reducido, que demuestra que los datos generados por la plataforma pueden producir mejoras sustanciales sobre los *benchmarks* de referencia cuando se integran con un protocolo de entrenamiento adecuado.

Este trabajo no se limita a determinar si los datos sintéticos mejoran el rendimiento en HTR, sino que se amplía a cómo diseñar y evaluar una plataforma abierta y escalable de generación sintética multilingüe para cargas de entrenamiento y *fine-tuning*, así como a analizar su impacto en términos de rendimiento, coste computacional y capacidad de transferencia. Esta reformulación desplaza el foco del trabajo desde el conjunto de datos en sí hacia la infraestructura computacional que lo sustenta, sin renunciar al valor aplicado de la validación funcional.

A. Contribuciones

Las contribuciones principales del trabajo son las siguientes:

1. Diseñamos una plataforma abierta y configurable para generación sintética multilingüe de muestras manuscritas orientadas a entrenamiento y *fine-tuning* en HTR.

[1]Dpto. de Informática, Universitat de Lleida (UdL), e-mail: `joc13@alumnes.udl.cat`.

2. Caracterizamos su escalabilidad y rendimiento en generación paralela, incluyendo *throughput*, tiempo por imagen y coste por configuración visual.
3. Analizamos el papel de los datos sintéticos como soporte a experimentos de *fine-tuning* sobre *benchmarks* reales, comparando dos estrategias: entrenamiento exclusivamente sintético y entrenamiento mixto, sobre RIMES e IAM.
4. Liberamos código, configuraciones y una estructura experimental reproducible que facilita replicación, trazabilidad y extensión a nuevas lenguas y escenarios.

El resto del artículo se organiza del siguiente modo. La Sección II resume los trabajos relacionados. La Sección III describe datasets y métricas. La Sección IV presenta la plataforma. La Sección V incorpora tanto la evaluación de prestaciones como la validación downstream en HTR. Finalmente, la Sección VI recoge las conclusiones y líneas futuras.

II. Trabajos relacionados

Los trabajos previos sobre HTR y generación sintética pueden agruparse en tres grandes familias resumidas esquemáticamente en la Figura 1

La primera comprende los *pipelines* basados en renderizado y transformaciones, donde el texto se sintetiza a partir de tipografías y se degrada mediante perturbaciones controladas [10, 11]. Este enfoque es atractivo porque escala con facilidad, genera transcripciones *ground-truth* de forma natural y permite adaptar el *pipeline* a nuevas lenguas sin necesidad de muestras reales. La segunda familia agrupa aproximaciones composicionales para análisis de documentos, donde la escritura sintética se combina con fondos o capas documentales más complejas para aproximarse mejor a condiciones reales [13,14]. La tercera familia incluye modelos generativos aprendidos —por ejemplo, GAN o difusión [15, 16]— que persiguen un mayor realismo visual a costa de una complejidad computacional significativamente superior.

En la literatura de HTR, los datos sintéticos se han utilizado sobre todo como mecanismo de arranque o ampliación del entrenamiento. La evidencia previa sugiere que el preentrenamiento o la mezcla de datos sintéticos y reales puede mejorar la transferencia [5,12], especialmente cuando existe cercanía visual o lingüística entre ambos dominios. Sin embargo, gran parte de los trabajos reportan el valor de los datos sintéticos desde la óptica del reconocimiento, mientras que la plataforma de generación queda descrita de forma secundaria y rara vez se acompaña de una caracterización detallada de rendimiento, escalabilidad o coste.

Esa es precisamente la brecha que aborda este artículo. En lugar de presentar únicamente un nuevo conjunto de datos o una nueva validación de HTR, tratamos el generador como una plataforma computacional reproducible cuya utilidad puede analizarse en dos planos complementarios: (i) prestaciones de generación y (ii) valor downstream. Desde esta perspectiva, la complejidad del realismo visual, la disponibilidad de tipografías por idioma, el paralelismo multicore y la huella de almacenamiento dejan de ser detalles de implementación y pasan a ser variables experimentales relevantes, tal como se aborda en la literatura de computación paralela [20, 21].

Para contextualizar este posicionamiento, la Tabla I resume los conjuntos sintéticos más relevantes identificados en la literatura, atendiendo especialmente a su disponibilidad, cobertura lingüística y esquema de metadatos.

Del análisis se desprenden dos carencias principales. Primero, la disponibilidad efectiva es limitada: solo Font[2] ofrece código de generación abierto; los demás carecen de descarga funcional. Segundo, ninguno define un esquema de metadatos que asocie a cada muestra los factores de generación —tipografía, fondo, perturbaciones, lengua—, lo que impide medir efectos aislados sobre el reconocimiento y dificulta la extensión a nuevas lenguas. Nuestro framework aborda directamente ambas limitaciones mediante un pipeline abierto con configuración declarativa y un esquema mínimo de metadatos por muestra que habilita ablaciones controladas y particiones estratificadas reproducibles.

III. Conjuntos de datos y métricas

A. Métricas de reconocimiento

Las métricas principales utilizadas en HTR son la *Character Error Rate* (CER) y la *Word Error Rate* (WER), ambas derivadas de la distancia de edición de Levenshtein [6]. La CER es más granular y resulta útil para comparar pequeñas diferencias entre modelos, mientras que la WER refleja mejor el impacto funcional a nivel de lectura. En este trabajo las empleamos como métricas *downstream* para validar si los datos generados por la plataforma son útiles como soporte de entrenamiento.

B. Benchmarks reales

Para la validación funcional utilizamos dos *benchmarks* ampliamente reconocidos:

- RIMES (francés), corpus de correo manuscrito con particiones de entrenamiento, validación y test ampliamente utilizadas en HTR.
- IAM (inglés), *benchmark* de referencia con particiones *writer-independent*, adecuado para estudiar transferencia y *fine-tuning*.

Estos conjuntos permiten contrastar dos escenarios: uno donde el modelo base dispone de una cercanía lingüística más favorable (francés) y otro donde la lengua objetivo no está representada en el preentrenamiento original (inglés), lo que incrementa la dificultad de la transferencia.

C. Datos sintéticos generados

A nivel experimental, empleamos configuraciones con distintos fondos y niveles de perturbación para caracterizar el coste de la generación. En la va-

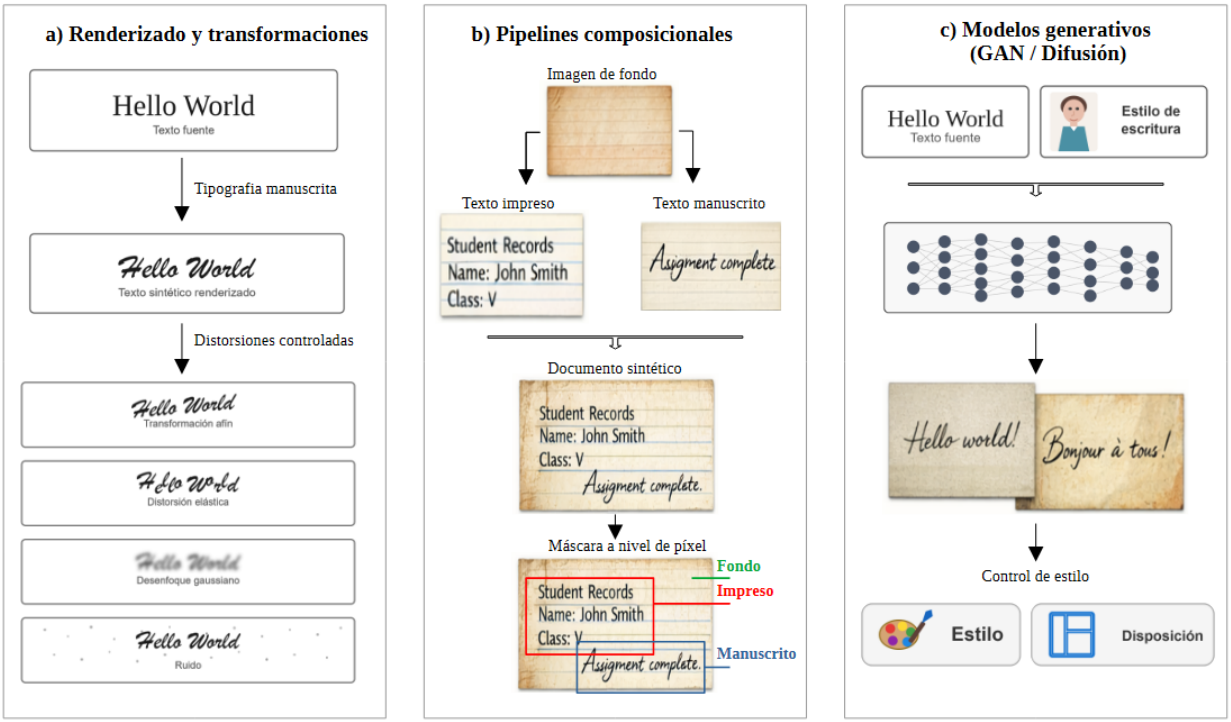


Fig. 1: Taxonomía de los tres enfoques principales para la generación de datos sintéticos de escritura manuscrita: (a) renderizado con tipografías y distorsiones controladas, (b) composición por capas sobre fondos de documento, y (c) modelos generativos aprendidos (GAN y difusión).

Tabla I: Resumen comparativo de conjuntos de datos sintéticos para HTR según lengua, granularidad y tamaño aproximado del Ground Truth (GT), descarga y principales resultados publicados.

Dataset	**Lengua**	**Granularidad y tamaño del GT**	**Descarga**	**Publicación**	**Uso reportado / validación**
IIIT-HWS	Inglés	Palabra (9M)	CVIT (inactivo)	Krishnan et al. (2016)	Preentrenamiento para *word recognition* y *word spotting*; validado en IAM, GW, Botany y Konzilsprotokolle.
Synthetic source words	Inglés Francés Catalán Alemán	Palabra (5.6M)	N/D	Kang et al. (2020)	Adaptación *synthetic-to-real* no supervisada; validado en GW, IAM, RIMES, CVL y ESPOSALLES. Reducción del gap sintético–real entre 24 % y 63 %.
Leopardi Synth	Italiano (histórico)	Línea (111.4k)	N/D	Cascianelli et al. (2021)	Preentrenamiento para HTR histórico de línea. Supera preentrenar con IAM, RIMES, ICFHR14 o ICFHR16; CER 3.2 % / WER 12.7 % tras *fine-tuning* con 100 % del real.
Font[2]	Inglés	Palabra (108M)	GitHub	Pippi et al. (2023a)	Preentrenamiento para *writer retrieval*, identificación, verificación y clasificación; validado en IAM, CVL y CEDAR. Resultados competitivos con enfoques específicos de cada tarea.

lidación downstream reportada aquí, los conjuntos empleados para *fine-tuning* funcional corresponden a imágenes en modo línea, fondo blanco y sin perturbaciones, con 8.445 muestras en francés y 8.574 en inglés.

IV. Marco propuesto

La plataforma propuesta se organiza como un *pipeline* secuencial con etapas bien delimitadas: (i) adquisición de textos, (ii) adquisición y verificación de tipografías, (iii) generación de fondos, (iv) renderizado de imágenes, (v) aplicación de perturbaciones y (vi) serialización de metadatos. La configuración es declarativa, lo que permite adaptar rápidamente la generación a nuevas lenguas, alfabetos compatibles y convenciones de escritura. La Figura 2 resume la arquitectura del sistema.

A. Adquisición de textos

El sistema obtiene textos de dominio público mediante la API de Wikipedia y los normaliza en un formato apto para la generación. La plataforma soporta 37 lenguas de alfabeto latino, desde lenguas de alta disponibilidad (inglés, español, francés) hasta lenguas minoritarias (bretón, asturiano, aragonés). Junto al contenido textual se preservan metadatos de procedencia que facilitan trazabilidad y reproducibilidad.

B. Tipografías y validación

La plataforma localiza tipografías manuscritas candidatas, las descarga y verifica en dos fases: com-

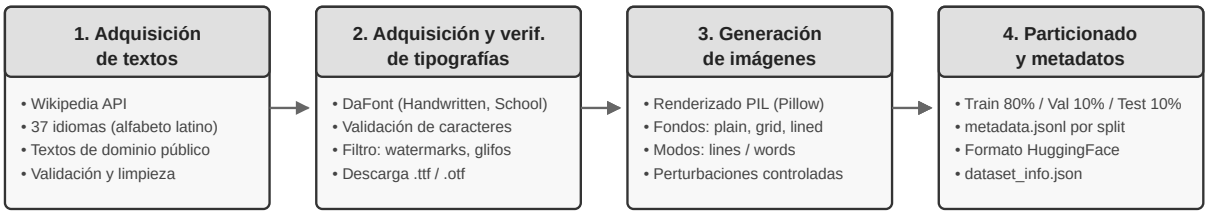


Fig. 2: Pipeline del marco propuesto para la generación de conjuntos de datos sintéticos de escritura manuscrita.

patibilidad de caracteres y renderizado efectivo. Este paso es particularmente relevante en escenarios multilingües, ya que la disponibilidad real de fuentes válidas depende del repertorio de caracteres exigido por cada idioma. Por ello, el número de fuentes válidas se considera una variable experimental útil en la evaluación de prestaciones.

C. Fondos y perturbaciones

El sistema puede generar fondos lisos, pautados o cuadriculados en varios colores y aplicar distintos niveles de perturbación visual, desde configuraciones *clean* hasta degradaciones más severas (*degraded* y *severe*). Este diseño no solo permite simular escenarios de captura realistas, sino también estudiar el compromiso entre fidelidad visual y coste computacional. La Figura 3 ilustra ejemplos de imágenes sintéticas con los tres niveles de calidad.

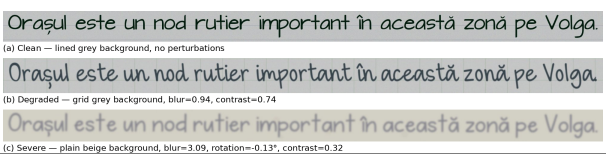


Fig. 3: Ejemplos de imágenes sintéticas generadas: (a) *clean*, (b) *degraded*, (c) *severe*.

D. Paralelización y metadatos

La generación de imágenes se ejecuta en paralelo mediante `multiprocessing`, siguiendo patrones de paralelismo de datos [22]. Cada imagen producida se acompaña de una entrada de metadatos que registra texto *ground-truth*, idioma, tipografía, fondo, calidad y parámetros de perturbación. Este diseño resulta útil tanto para reproducibilidad como para experimentos de ablación, al permitir agrupar posteriormente por cualquier factor controlado.

V. Evaluación experimental

Esta sección se divide en dos bloques complementarios. En primer lugar caracterizamos la plataforma desde el punto de vista de prestaciones: escalabilidad, *throughput*, tiempo por imagen, impacto de la complejidad visual, variación multilingüe y contribución relativa de las etapas del *pipeline*. En segundo lugar, validamos la utilidad downstream de los datos sintéticos sobre *benchmarks* HTR reales.

A. Entorno y protocolo de benchmarking

Los experimentos se ejecutaron en un equipo con procesador Intel Core i7-14650HX (16 núcleos físicos, 24 hilos), 32 GB de RAM y sistema operativo Windows 11.

La evaluación experimental se articuló sobre ejecuciones instrumentadas que registran tiempo total, imágenes generadas, *throughput*, latencia media por imagen, tamaño de salida y tiempos por etapa. La métrica de escalabilidad se expresa mediante el *throughput* y su evolución al aumentar el número de *workers*. Para el análisis multilingüe se monitoriza además el número de fuentes válidas por idioma, ya que esta variable condiciona tanto la diversidad como el coste del preprocesado.

B. Prestaciones de generación

B.1 Escalabilidad con número de *workers*

La Figura 4 muestra el *throughput* de generación a medida que aumenta el número de *workers*. El comportamiento es consistente con una plataforma multicore: el rendimiento crece claramente en los primeros niveles de paralelismo y después del cuarto *worker* aparece una región de saturación donde la mejora marginal disminuye.



Fig. 4: *Throughput* de generación en función del número de *workers*.

A medida que crece el paralelismo, el *throughput* aumenta con rapidez en los primeros niveles y después se estabiliza, lo que indica la presencia de costes no paralelizables y de sobrecarga de coordinación, en línea con la ley de Amdahl [19]. Este comportamiento refuerza la idea de que la plataforma ofrece un compromiso adecuado entre simplicidad de implementación y aprovechamiento multicore.

B.2 Coste del realismo visual en la generación sintética

El realismo visual es un factor relevante en la generación de datos sintéticos, ya que una mayor similitud con los datos reales puede mejorar el rendimiento

de los modelos de aprendizaje profundo. Para cuantificar el impacto del realismo visual, comparamos configuraciones sin fondos ni perturbaciones frente a variantes con fondos y degradaciones. La Figura 5 muestra el coste temporal global por configuración. Como era esperable, el tiempo total aumenta conforme se incrementa la complejidad visual.

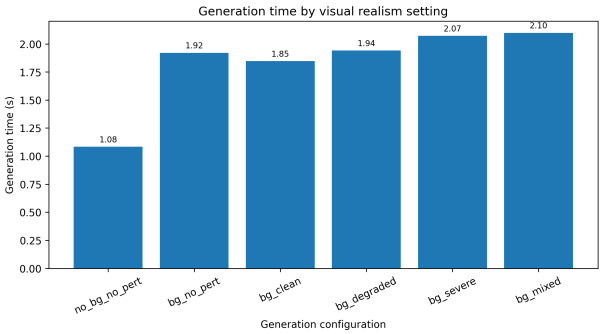


Fig. 5: Tiempo total de generación por configuración visual.

La Figura 6 presenta el *throughput* asociado a cada configuración, mostrando con claridad el compromiso entre fidelidad visual y coste computacional: configuraciones más ricas desde el punto de vista visual reducen el *throughput* y elevan el tiempo por imagen.

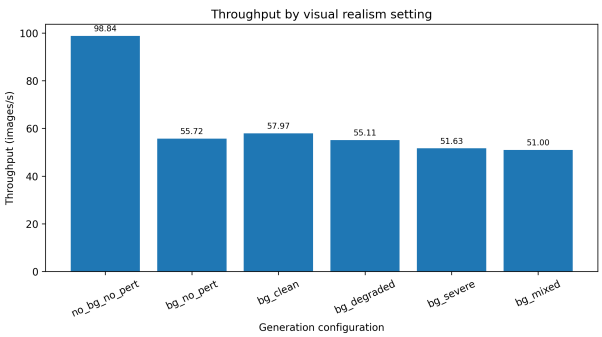


Fig. 6: *throughput* de generación por configuración visual.

Además del coste temporal, la complejidad visual afecta al tamaño final del *dataset*. La Figura 7 resume esta relación. Este aspecto es relevante para escenarios donde almacenamiento y transferencia de datos también forman parte del coste experimental.

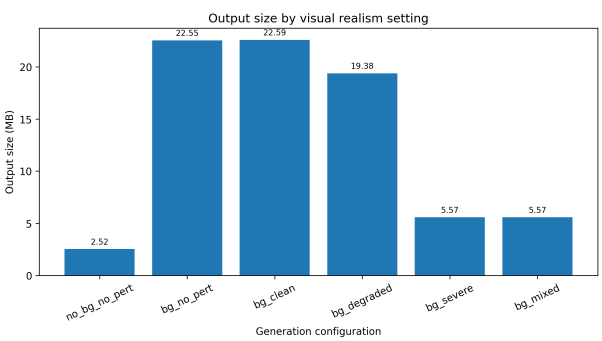


Fig. 7: Tamaño de salida por configuración visual.

B.3 Comparación multilingüe

La Figura 8 muestra el *throughput* de generación por idioma.

El *throughput* varía de forma no uniforme entre idiomas: el francés alcanza el valor más alto (71,19 imágenes/s), seguido del catalán (61,82) y el inglés

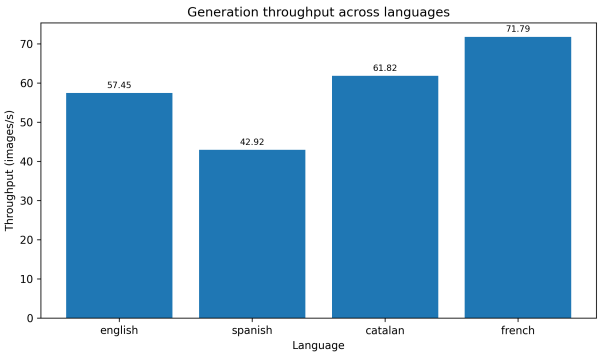


Fig. 8: Comparación multilingüe de *throughput*.

(57,45), mientras que el español presenta el rendimiento más bajo (42,92). Esta variación no es atribuible únicamente al volumen de imágenes generadas, sino también al número de fuentes válidas por idioma, como se discute más adelante.

La Figura 9 muestra el tamaño de salida por idioma.

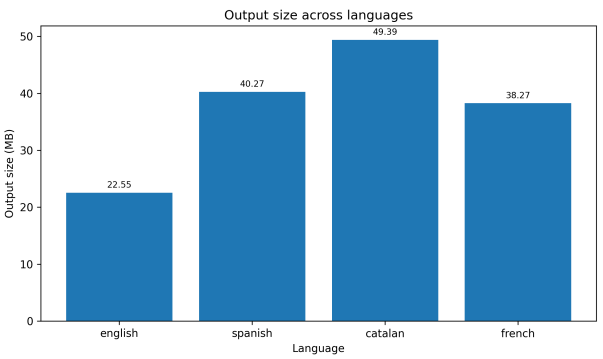


Fig. 9: Comparación multilingüe de tamaño de salida.

El catalán produce el dataset más voluminoso (46,59 MB), seguido del español (40,27 MB) y el francés (38,27 MB), mientras que el inglés genera el dataset más pequeño (22,55 MB). Esta variación refleja diferencias en la longitud media de los textos y en la distribución de modos de generación por idioma.

La Figura 10 recoge el número de fuentes válidas por idioma, variable que condiciona tanto la diversidad del dataset como el coste del preprocesado.

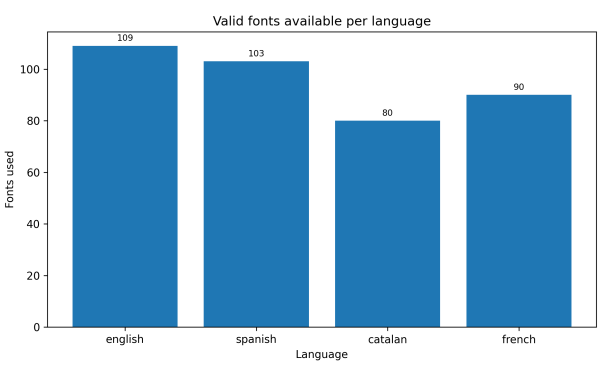


Fig. 10: Número de fuentes válidas por idioma.

El inglés y el español disponen del mayor número de fuentes válidas (109 y 103 respectivamente), lo que incrementa el esfuerzo de validación y filtrado previo. El catalán y el francés, con menos fuentes válidas (80 y 90), concentran la generación en un conjunto más reducido, lo que explica en parte su mayor *throughput* relativo observado en la Figura 8.

La relación entre disponibilidad de fuentes válidas y *throughput* se resume en la Figura 11.

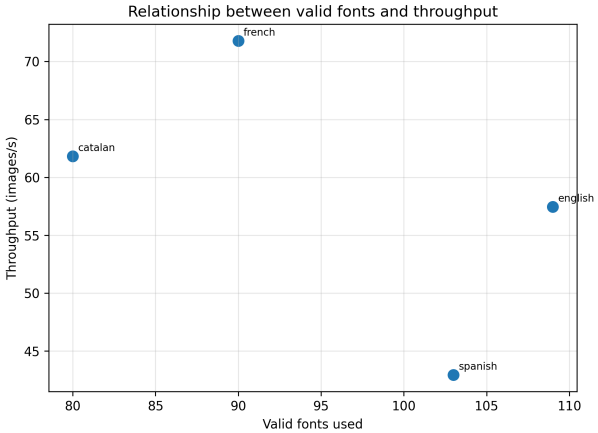


Fig. 11: Relación entre fuentes válidas y *throughput* en el experimento multilingüe.

La figura muestra una tendencia inversa entre el número de fuentes válidas y el *throughput*: los idiomas con más fuentes disponibles (inglés y español, con 109 y 103 respectivamente) presentan un *throughput* menor, mientras que los idiomas con menos fuentes (francés y catalán, con 90 y 80) alcanzan valores más altos. Este comportamiento indica que el coste de validación y filtrado de tipografías tiene un impacto medible en el rendimiento global de la plataforma, y sugiere que optimizar esta etapa podría mejorar el *throughput* en escenarios multilingües con repertorios de fuentes amplios.

B.4 Análisis de etapas del *pipeline*

Más allá del coste global, interesa identificar qué etapas dominan el coste computacional. La Figura 12 presenta la contribución relativa de cada etapa al coste total.

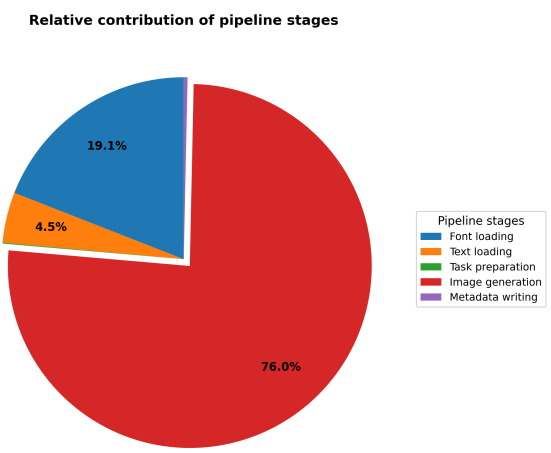


Fig. 12: Contribución relativa de cada etapa al coste total.

La generación de imágenes domina claramente el coste computacional, concentrando el 76,0 % del coste total. La carga de tipografías representa el segundo factor más relevante (19,1 %), lo que es coherente con el proceso de validación y filtrado de fuentes descrito en la Sección IV-B. La preparación de tareas contribuye con un 4,5 %, mientras que la carga de textos y la escritura de metadatos tienen un impacto marginal. Esta descomposición indica que las optimizaciones futuras deberían focalizarse en la etapa de renderizado y, secundariamente, en la validación de tipografías, que representa el principal cuello de botella fuera de la generación propiamente dicha.

Por último, la Figura 13 relaciona *throughput* y tamaño de salida, permitiendo comparar conjuntamente todos los experimentos realizados.

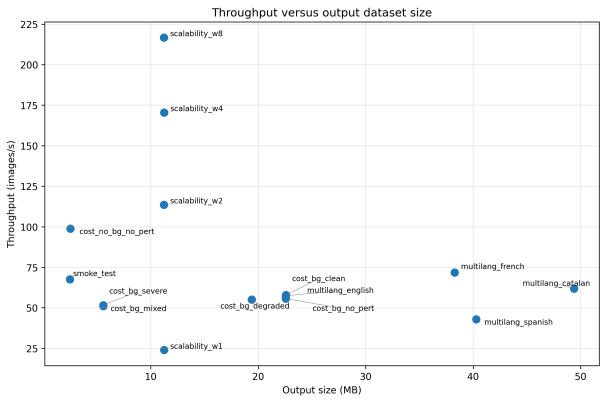


Fig. 13: Relación entre *throughput* y tamaño de salida del dataset.

La figura muestra dos clústeres bien diferenciados. El primero, en la zona superior izquierda, agrupa los experimentos de escalabilidad (*scalability_ w1*, *w4*, *w8*), que alcanzan el *throughput* más alto con un tamaño de salida reducido. El segundo clúster, en la zona derecha, agrupa los experimentos multilingües (*multilang_ french*, *multilang_ catalan*, etc.), que producen datasets más voluminosos a costa de un *throughput* menor. Las configuraciones de coste visual (*cost_ bg_ **) se sitúan en la zona inferior izquierda, con bajo *throughput* y tamaño de salida moderado. Este gráfico resume el compromiso de ingeniería de la plataforma: maximizar *throughput* y minimizar el tamaño de salida son objetivos que no siempre apuntan en la misma dirección.

C. Validación funcional downstream en HTR

C.1 Configuración experimental

Empleamos los modelos preentrenados FoNDUE-GD v2 [9]. Para francés utilizamos `FoNDUE-GD_v2_fr.mlmodel` y para inglés el modelo general `FoNDUE-GD_v2.mlmodel`. Los datasets reales fueron RIMES-2011-line (10.188 *train* / 1.138 *val* / 778 *test*) e IAM (8.917 *train* / 2.732 *val* / 1.704 *test*), con particiones *writer-independent* en el caso de IAM.

Los datasets sintéticos usados en esta validación contienen 8.445 líneas en francés y 8.574 en inglés, en modo línea, fondo blanco y sin perturbaciones. Para cada idioma se evaluaron tres condiciones: baseline, *fine-tuning* sintético y *fine-tuning* mixto. El entrenamiento se realizó con Kraken 7.0, *learning rate* 10^{-4}, *early stopping* con lag de 5 épocas y normalización NFD.

C.2 Resultados en RIMES

La Tabla II resume los resultados sobre RIMES.

Tabla II: Resultados sobre RIMES test (francés).

Modelo	CER (%)	WER (%)	Δ CER
Baseline FoNDUE-GD_v2_fr	19,43	53,86	—
Fine-tuning sintético	23,03	65,38	+3,60 pp
Fine-tuning mixto	9,74	28,64	–9,69 pp

El fine-tuning exclusivamente con datos sintéticos empeora el rendimiento y evidencia *catastrophic forgetting* [17, 18]. En cambio, el *fine-tuning* mixto reduce el CER aproximadamente un 50 % respecto al baseline, lo que indica que la plataforma es útil como soporte de entrenamiento, pero no como sustituto completo de datos reales.

C.3 Resultados en IAM

La Tabla III resume los resultados sobre IAM.

Tabla III: Resultados sobre IAM test (inglés).

Modelo	CER (%)	WER (%)	Δ CER
Baseline FoNDUE-GD_v2	24,45	70,77	—
Fine-tuning sintético	75,10	95,86	+50,65 pp
Fine-tuning mixto	10,29	32,47	–14,16 pp

En este escenario el *catastrophic forgetting* es más severo, coherente con la ausencia de inglés en el entrenamiento original del modelo base. Aun así, el *fine-tuning* mixto consigue una reducción del CER del 58 %, reforzando la robustez del enfoque cuando los datos sintéticos se emplean como complemento.

C.4 Síntesis comparativa y curriculum-LR

Los resultados anteriores corresponden al protocolo clásico de mezcla concurrente. Para complementar la validación funcional, evaluamos también un protocolo de *curriculum learning* con *learning rate* reducido (denotado curriculum-LR), consistente en dos fases secuenciales: (i) preentrenamiento sobre datos sintéticos con *learning rate* 10^{-4} y (ii) refinamiento sobre datos reales con *learning rate* reducido a 10^{-5}. Esta variante separa temporalmente la exposición al sintético y al real, y reduce las actualizaciones de peso durante la adaptación para preservar las representaciones adquiridas en la fase sintética. La Tabla IV sintetiza los mejores resultados de cada protocolo sobre ambos benchmarks.

Tabla IV: Síntesis de resultados sobre RIMES (francés) e IAM (inglés). Mejora relativa respecto al baseline correspondiente.

Protocolo	Francés (RIMES)	Inglés (IAM)
Baseline	19,43 %	24,45 %
Sintético-only	23,03 % (+18 % rel.)	75,10 % (+207 % rel.)
Mixto convencional	9,74 % (–50 % rel.)	10,29 % (–58 % rel.)
Real-only	7,83 % (–60 % rel.)	9,44 % (–61 % rel.)
Curriculum-LR	**6,60 % (–66 % rel.)**	**6,03 % (–75 % rel.)**

El protocolo curriculum-LR supera al *fine-tuning* exclusivamente real en ambos idiomas, con una mejora absoluta de 1,23 pp CER en francés y 3,41 pp en inglés, lo que refuerza la utilidad de los datos generados por la plataforma cuando se integran con un protocolo de entrenamiento adecuado. La mejora es marcadamente mayor en inglés, donde el modelo base no incluye la lengua objetivo en su preentrenamiento original, lo que apunta a que el aporte del preentrenamiento sintético crece en escenarios de baja cobertura lingüística previa.

Estos resultados permiten reinterpretar la validación funcional como una prueba completa del valor downstream de la plataforma. La aportación principal del trabajo no es un nuevo estado del arte de reconocimiento, sino la demostración de que una infraestructura abierta de generación sintética puede producir datos útiles tanto para *fine-tuning* mixto como para preentrenamiento de protocolos de *curriculum learning*, al tiempo que ofrece una caracterización cuantitativa de su coste y comportamiento computacional.

VI. Conclusiones y trabajo futuro

Hemos presentado una plataforma abierta y configurable para generación sintética multilingüe de escritura manuscrita orientada a HTR desde una perspectiva paralelizable y escalable. El trabajo mantiene la motivación aplicada del reconocimiento manuscrito, pero desplaza el foco hacia la infraestructura computacional: diseño de plataforma, paralelismo, coste, reproducibilidad y utilidad downstream.

La caracterización experimental permite extraer cinco conclusiones principales:

1. La plataforma escala de forma práctica en configuración multicore, con mejoras claras de *throughput* al aumentar el número de *workers*, aunque con rendimientos decrecientes a medida que crecen la contención y los costes no paralelizables.
2. El realismo visual tiene un coste medible: fondos y perturbaciones incrementan tiempo, latencia por imagen y tamaño de salida, de modo que existe un compromiso explícito entre fidelidad visual y coste computacional.
3. El comportamiento multilingüe no es uniforme; la disponibilidad de fuentes válidas y la compatibilidad de caracteres influyen en el coste global y deben tratarse como variables de plataforma, no solo como detalles de preprocesado.
4. Desde el punto de vista funcional, el *fine-tuning* mixto con datos reales y sintéticos reduce el CER entre un 50 % y un 58 % respecto al baseline, mientras que el ajuste exclusivamente sintético degrada la generalización por *catastrophic forgetting*.
5. Un protocolo de *curriculum learning* con *learning rate* reducido aplicado sobre los mismos datos generados alcanza 6,60 % CER en RIMES y 6,03 % CER en IAM, superando al *fine-tuning* exclusivamente real y demostrando que la plataforma puede producir datos útiles más allá del régimen mixto convencional.

En conjunto, estos resultados apoyan la idea de que la generación sintética no debe entenderse únicamente como un problema de datasets, sino también como una carga computacional evaluable que puede diseñarse, instrumentarse y optimizarse como infraestructura de Deep Learning.

Como trabajo futuro proponemos: (i) incorporar de forma sistemática configuraciones *clean/degrade-*

d/severe en la validación downstream para estudiar el *trade-off* entre coste y utilidad; (ii) comparar distintas plataformas hardware o estrategias de particionado por lotes; (iii) ampliar el marco a párrafos y páginas completas; (iv) contrastar el coste-beneficio del enfoque de renderizado frente a generadores aprendidos basados en GAN o difusión; y (v) explorar el coste computacional de protocolos de entrenamiento en dos fases (*curriculum learning*), donde el sintético actúa como capa de preentrenamiento, en términos de tiempo total de entrenamiento y huella de memoria.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades (MICIU) AEI/10.13039/501100011033 bajo el contrato PID2023-146193OB-I00.

Referencias

[1] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke y J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 855–868, 2009.

[2] B. Shi, X. Bai y C. Yao, "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 11, pp. 2298–2304, 2017.

[3] J. Puigcerver, "Are multidimensional recurrent layers really necessary for handwritten text recognition?," en *Proceedings of ICDAR*, 2017, pp. 67–72.

[4] M. Li et al., "TrOCR: Transformer-based optical character recognition with pre-trained models," en *Advances in Neural Information Processing Systems*, 2021.

[5] K. Dutta, P. Krishnan, M. Mathew y C. V. Jawahar, "Improving CNN-RNN hybrid networks for handwriting recognition," en *Proceedings of ICFHR*, 2018, pp. 80–85.

[6] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[7] U.-V. Martí y H. Bunke, "The IAM-database: an English sentence database for offline handwriting recognition," *International Journal on Document Analysis and Recognition*, vol. 5, no. 1, pp. 39–46, 2002.

[8] E. Grosicki y H. El Abed, "RIMES evaluation campaign for handwritten mail processing," en *Proceedings of ICDAR*, 2009, pp. 941–945.

[9] E. Solé i Martí, C. Vidal-Gorène y S. Gabay, "FoNDUE ground-truth document understanding engine models," Zenodo, 2024.

[10] M. Jaderberg, K. Simonyan, A. Vedaldi y A. Zisserman, "Synthetic data and artificial neural networks for natural scene text recognition," en *Workshop on Deep Learning, NeurIPS*, 2014.

[11] P. Krishnan y C. V. Jawahar, "Generating synthetic data for text recognition," *arXiv preprint arXiv:1608.04224*, 2016.

[12] S. Gaur, S. Sonkar y P. P. Roy, "Generation of synthetic training data for handwritten indic script recognition," en *Proceedings of ICDAR*, 2015, pp. 491–495.

[13] S. Capobianco y S. Marinai, "DocEmul: a toolkit to generate structured historical documents," en *Proceedings of ICDAR*, 2017, pp. 1186–1191.

[14] J. Jo, H. I. Koo, J. W. Soh y N. I. Cho, "Handwritten text segmentation via end-to-end learning of convolutional neural networks," *Multimedia Tools and Applications*, vol. 79, no. 43–44, pp. 32137–32150, 2020.

[15] S. Fogel, H. Averbuch-Elor, S. Cohen, S. Mazor y R. Litman, "ScrabbleGAN: Semi-supervised varying length handwritten text generation," en *Proceedings of CVPR*, 2020, pp. 4324–4333.

[16] L. Kang, P. Riba, Y. Wang, M. Rusiñol, A. Fornés y M. Villegas, "GANwriting: Content-conditioned generation of styled handwritten word images," en *Proceedings of ECCV*, 2020, pp. 273–289.

[17] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in Cognitive Sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[18] J. Kirkpatrick et al., "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[19] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," en *Proceedings of the AFIPS Spring Joint Computer Conference*, 1967, pp. 483–485.

[20] J. Dean y S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[21] L. Dagum y R. Menon, "OpenMP: An industry-standard API for shared-memory programming," *IEEE Computational Science and Engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[22] M. McCool, J. Reinders y A. Robison, *Structured Parallel Programming: Patterns for Efficient Computation*. Morgan Kaufmann, 2012.

Securizando el sistema de ficheros Expand utilizando *blockchain*

Diego Camarmas-Alonso[1], Félix García-Carballeira[1], Alejandro Calderón-Mateos[1] y Jesús Carretero[1]

Resumen— **Actualmente, vivimos en una etapa tecnológica en la que la generación, almacenamiento, y procesamiento de datos utilizando aplicaciones de *big data* e inteligencia artificial cada vez es más común. Además, han surgido nuevos paradigmas, como el *Computing Continuum*, que recopilan y procesan datos en tiempo real. Sin embargo, estos datos, en muchas ocasiones, contienen información crítica, por lo que deben almacenarse de forma segura, garantizando su inmutabilidad y trazabilidad. Para ello, existen tecnologías como la *blockchain*, que permite almacenar los datos de forma segura, pero esta tecnología presenta problemas de escalabilidad, especialmente cuando los datos son generados por dispositivos IoT, ya que el tiempo de validación de las transacciones aumenta junto con el volumen de los datos a almacenar. Por todo ello, en este trabajo se tiene como objetivo integrar el sistema de ficheros ad-hoc Expand con la tecnología *blockchain*, lo que permitirá securizar el sistema de ficheros para almacenar datos críticos, incrementar sustancialmente el espacio disponible, aprovechar el rendimiento de E/S proporcionado por Expand y reducir el tamaño de las transacciones de *blockchain* para que el proceso de validación sea menos costoso.**

Palabras clave— **Expand, *Blockchain*, Sistema de ficheros ad-hoc, *High performance computing*.**

I. Introducción

En los últimos años, vivimos en una etapa tecnológica en la que la generación, el almacenamiento y el procesamiento de datos mediante aplicaciones de *big data* o de inteligencia artificial cada vez es más común. Además, recientemente ha surgido un nuevo paradigma llamado *Computing Continuum* [1], que consiste en la recopilación y el procesamiento de datos en tiempo real, lo que mejora los tiempos de respuesta.

Sin embargo, muchos de estos datos contienen información crítica que debe almacenarse de forma segura, garantizando su integridad y trazabilidad. Evitando que los datos almacenados puedan ser modificados por un agente malicioso, así como mantener el historial de cambios de los datos, respectivamente.

Para ello, han surgido tecnologías como la *blockchain* [2], [3], que es una de las tecnologías más populares en la actualidad para el almacenamiento de transacciones críticas. Esto se debe a que esta tecnología permite garantizar la seguridad, la integridad y la trazabilidad de los datos en sistemas distribuidos. Para ello, esta tecnología funciona como un libro de contabilidad descentralizado, inmutable y transparente que permite registrar las transacciones de forma segura. Lo que la hace muy apropiada para aplicaciones que requieren una gran trazabilidad de los datos. Por ejemplo, el sistema sanitario, las cadenas de suministro, el sistema financiero o el sector inmobiliario, entre otros [4].

No obstante, *blockchain* sigue presentando retos de escalabilidad, ya que a medida que aumenta el volumen de datos, el tiempo de procesamiento y validación de una transacción aumenta. Este desafío es aún mayor cuando se utilizan dispositivos IoT, ya que estos generan grandes cantidades de datos en periodos muy cortos y deben validarse y almacenarse casi en tiempo real. Además de la escalabilidad, otro problema que presenta la *blockchain* es el almacenamiento, ya que cada transacción, además de los datos de esta, contiene el material criptográfico necesario para garantizar la seguridad. Por lo tanto, cuando se utilizan dispositivos IoT, que generan grandes volúmenes de datos, el espacio de almacenamiento destinado al material criptográfico es significativo, lo que reduce el espacio disponible para almacenar los propios datos [5].

Para ayudar a mejorar la escalabilidad y ofrecer un mayor espacio de almacenamiento, existen multitud de sistemas de ficheros paralelos y distribuidos, especialmente los nuevos sistemas de ficheros ad-hoc que se caracterizan por ser capaces de adaptar su despliegue a las necesidades de las aplicaciones que los utilizan y utilizar el almacenamiento local disponible en los nodos de cómputo de los entornos HPC.

Pero la clave sería utilizar un sistema de ficheros ad-hoc para guardar los datos recogidos para su posterior postprocesado por las aplicaciones *big data*, en lugar de almacenar todo en la red *blockchain*, ya que las operaciones de E/S son más eficientes en los sistemas de ficheros ad-hoc. Pero la mayoría de los sistemas de ficheros actuales no son capaces de garantizar la integridad y no repudio de los datos de forma nativa y, por lo tanto, no son seguros para su uso por aplicaciones críticas. Este problema se ha abordado más en el ámbito del almacenamiento distribuido [6], pero no en sistemas HPC.

En este trabajo, partiendo de nuestro trabajo anterior en *blockchain* [5] y del sistema de ficheros ad-hoc Expand [7], se realizan contribuciones para:

- Securizar el sistema de ficheros ad-hoc Expand utilizando para ello la tecnología *blockchain*. Lo que le permite a Expand ofrecer integridad y no repudio de los datos, cuando la aplicación que lo utiliza lo requiera. Así como disponer de los datos almacenados en los mismos nodos de cómputo donde serán postprocesados.
- Usar Expand para el almacenamiento de los da-

[1]Universidad Carlos III de Madrid, ROR: https://ror.org/03ths8210, Departamento de Informática, Avenida de la Universidad, 30 (Edificio Sabatini), 28911 Leganés (Madrid), España. e-mail: {dcamarma,fgcarbal,acaldero,jcarrete}@inf.uc3m.es

tos en *blockchain*, lo que permitirá incrementar sustancialmente el espacio disponible, así como aprovechar su rendimiento de E/S.
- Combinar Expand y *blockchain* para reducir el tamaño de las transacciones de *blockchain*, lo que repercutirá en el tiempo de validación de las transacciones, que también se reducirá.

El sistema de almacenamiento resultante combina las ventajas de ambas tecnologías: el alto rendimiento en las operaciones de E/S, así como poder almacenar los datos de forma segura.

El resto del documento se estructura de la siguiente forma: la Sección II presenta el estado del arte; la Sección III describe las principales características de Expand; la Sección IV presenta la integración de Expand con la *blockchain*; la Sección V presenta un caso de uso para estudiar la viabilidad de esta nueva propuesta. Y, por último, en la Sección VI se presentan las principales conclusiones y los trabajos futuros.

II. Estado del arte

En esta sección se estudian trabajos relacionados con este artículo y que son relevantes para los aspectos tratados en el mismo, a saber, sistemas de ficheros ad-hoc y tecnología blockchain.

A. Sistemas de ficheros paralelos ad-hoc

Un desafío principal para avanzar hacia la computación masiva es equilibrar el rendimiento de cómputo y de almacenamiento de los sistemas *High-End Computing* (HEC) para almacenar y manipular conjuntos de datos extremadamente grandes de forma flexible, al tiempo que se proporcionan soluciones transparentes a los usuarios.

Los *burst buffers* se introdujeron originalmente como un nivel adicional de almacenamiento mediante la extensión de los nodos de E/S con SSDs [8] o memoria no volátil [9]. Nos referiremos a estos *burst buffers* como *burst buffers* externos. Gradualmente, los nodos de cómputo estándar de los clústeres HPC también se ampliaron con SSDs para ser utilizados como *burst buffers* internos al nodo. Hoy en día, los nodos de cómputo están equipados con las tecnologías de almacenamiento más rápidas disponibles, que van desde SSDs NVMe hasta memoria persistente. Ejemplos de la integración de almacenamiento rápido y local al nodo incluyen las arquitecturas del supercomputador Summit [10] y del sistema El Capitan Rabbit [11].

Por lo tanto, es necesario integrar estos *burst buffers* en la arquitectura de almacenamiento por niveles de las implementaciones HPC y hacerlos transparentes para los usuarios y las aplicaciones. Pero, para integrar eficazmente los *burst buffers* para resolver el problema de E/S en HPC, confiar únicamente en la tecnología no es suficiente, y se requieren cambios arquitectónicos en los sistemas de ficheros. Una solución para afrontar los desafíos mencionados es desplegar sistemas de ficheros ad-hoc [12]. Los sistemas de ficheros ad-hoc son sistemas de ficheros que se instancian en el contexto de una aplicación o conjunto de aplicaciones y cuya vida útil está vinculada a la de su contexto, ajustando los parámetros de configuración existentes a nivel del sistema. Además, permiten que las aplicaciones que se ejecutan en el contexto del sistema de ficheros ad-hoc accedan a varios dispositivos de almacenamiento bajo un único espacio de nombres. A continuación se muestran algunas soluciones relevantes.

BurstFS de 2016 es uno de los primeros sistemas de ficheros ad-hoc diseñados explícitamente para combinar temporalmente recursos de almacenamiento rápido locales al nodo pertenecientes a una aplicación HPC [13]. Al operar como un sistema de ficheros temporal, las instancias de BurstFS se inician al comienzo de un trabajo por lotes (*batch job*), proporcionando un espacio de nombres unificado para todos los nodos asociados. Las instancias de BurstFS se limpian y eliminan al final de la ejecución del trabajo. Echofs [14] fue diseñado para proporcionar sistemas de ficheros temporales rápidos para trabajos por lotes HPC utilizando el almacenamiento local del nodo. La motivación de ambos sistemas de ficheros fue la capacidad de los sistemas de ficheros ad-hoc para relajar la semántica POSIX (no la sintaxis) para trabajos seleccionados. UnifyFS [11] de 2023 es un sistema de ficheros de *burst buffer* que agrega recursos de almacenamiento local al nodo y memoria, y los proporciona bajo un espacio de nombres unificado.

GekkoFS [15] es un sistema de ficheros distribuido altamente escalable para aplicaciones HPC que se despliega durante la vida útil de un trabajo de cómputo o flujo de trabajo. GekkoFS combina el almacenamiento local al nodo basado en memoria *flash*, a menudo no utilizado, en un único espacio de nombres global, agregando su capacidad y rendimiento de E/S. MadFS [16] ha sido desarrollado basándose en la arquitectura de GekkoFS y portado al ordenador de alto rendimiento Cloudbrain ARM64 utilizando el lenguaje Rust. Los desarrolladores mantuvieron la arquitectura básica de GekkoFS. Expand [7] fue originalmente un sistema de ficheros paralelo diseñado para clústeres heterogéneos, estrechamente ligado a la interfaz MPI-IO. Recientemente, Expand ha sido rediseñado para convertirse en un sistema de ficheros ad-hoc para entornos HPC denominado Expand Ad-hoc [17]. BeeGFS On Demand (BeeOND) [18] permite la creación fácil e instantánea de instancias de BeeGFS para construir sistemas de ficheros temporales que combinan el rendimiento y la capacidad de SSDs internos al nodo o memoria RAM durante la ejecución de un trabajo de cómputo. Todos los nodos de cómputo de un sistema de ficheros BeeOND son clientes del almacenamiento paralelo y contribuyen a él simultáneamente. Por último, Hercules [19] es un sistema de almacenamiento en memoria que opera en espacio de usuario. Hercules sigue una arquitectura cliente-servidor y puede ser desplegado por cada aplicación, donde el cliente es responsable del despliegue de la entidad servidor. Hercules está vinculado a un proceso de aplicación a través de su capa *front-end*. La capa cliente es responsable

de gestionar la localidad de los datos e implementar diversos patrones de E/S basados en políticas de distribución. Cada aplicación puede personalizar el tamaño de cada despliegue de Hercules según los requisitos de E/S. Esto permite que múltiples despliegues de Hercules coexistan en el mismo sistema informático, manteniendo el aislamiento de los datos.

B. Tecnología blockchain

La tecnología *blockchain* puede clasificarse en función de la naturaleza de la red subyacente. En términos generales, se distinguen dos categorías principales: redes abiertas y redes privadas. En las redes abiertas, cualquier usuario puede participar y registrar operaciones en la cadena de bloques. Ethereum [20] constituye el principal exponente de esta categoría, siendo su aplicación más destacada la implementación de *blockchain* en desarrollos contemporáneos de criptomonedas, como Bitcoin [21]. Por otro lado, en las redes privadas, únicamente un conjunto limitado de nodos posee la capacidad de registrar y gestionar operaciones dentro del sistema *blockchain*. En este contexto, Hyperledger Fabric [22] se ha consolidado como la plataforma de referencia para este tipo de redes. A diferencia de las redes públicas, las redes privadas están compuestas exclusivamente por participantes pertenecientes a una organización específica. Esta característica elimina el anonimato en las transacciones, dado que los participantes deben autenticar su identidad mediante certificados digitales. Aunque en este tipo de redes la modificación de transacciones resulta más factible, el nivel de inmutabilidad se mantiene elevado. En el presente trabajo, se ha optado por el uso de Hyperledger Fabric debido a la necesidad de construir redes formadas por entidades con distintos niveles de privilegio.

Hyperledger Fabric alcanza el consenso mediante el uso de un servicio *backend*, denominado servicio de ordenación, que actúa como intermediario en el intercambio de mensajes entre emisores y receptores [23]. Este servicio garantiza que todos los nodos receptores perciban los mensajes en un orden idéntico; en consecuencia, todos ellos ejecutan las mismas operaciones y validaciones, lo que permite alcanzar el consenso del sistema [24]. En la versión 2.x, Hyperledger Fabric emplea el algoritmo de consenso RAFT, basado en un modelo de tipo líder-seguidor. Asimismo, Hyperledger Fabric implementa un modelo de tolerancia a fallos por caída (*Crash Fault Tolerance*, CFT), que permite alcanzar consenso tanto en sistemas de una única organización como en entornos multiorganización. Dicho modelo garantiza la resiliencia frente a fallos del sistema, tales como caídas de nodos o particiones de red. En un sistema con N nodos, CFT permite tolerar hasta N/2 fallos de este tipo.

No obstante, el rendimiento de las soluciones basadas en *blockchain* ha constituido tradicionalmente una limitación significativa, especialmente en su aplicación a sistemas en tiempo real, como aquellos basados en el Internet de las Cosas (IoT) [25]. Diversos estudios han abordado el análisis del rendimiento de distintas plataformas de desarrollo, mientras que investigaciones más recientes han propuesto optimizaciones orientadas a mejorar la escalabilidad y eficiencia de herramientas basadas en RAFT [26], [27]. Estos aspectos resultan especialmente críticos en entornos con dispositivos IoT, caracterizados por la generación de grandes volúmenes de datos en intervalos de tiempo reducidos, lo que dificulta su gestión y almacenamiento. Sin embargo, la mayoría de las propuestas existentes se centran en modificar la arquitectura de Hyperledger Fabric con el objetivo de reducir el tiempo de ejecución de la fase de validación, identificada como el principal cuello de botella en la tecnología *blockchain*. Como consecuencia, dichas optimizaciones dependen tanto de la herramienta de desarrollo como del protocolo de consenso modificado utilizado en la validación de transacciones, lo que limita su aplicabilidad general y dificulta su adopción por parte de los usuarios.

En trabajos previos, con el objetivo de mitigar esta sobrecarga, se desarrolló una implementación mejorada de las transacciones en Hyperledger que no requiere modificaciones en la plataforma Fabric [5]. En particular, se propuso el uso de transacciones atómicas y técnicas de validación agrupada, aplicadas sobre la propia plataforma *blockchain* y no dependientes de una implementación específica. Los resultados experimentales evidenciaron que dichas optimizaciones permiten mejorar significativamente el rendimiento y la escalabilidad del proceso de validación de transacciones, posibilitando una validación de alta velocidad y un almacenamiento consistente de los datos en condiciones cercanas al tiempo real. Esto, a su vez, facilita la disponibilidad de los datos para su consulta en procesos de validación continua de contratos.

C. Sistemas de ficheros y blockchain

El diseño de los sistemas de ficheros distribuidos globalmente (DFS) presenta desventajas tales como la inestabilidad y la falta de mecanismos de auditoría e incentivos. Por ejemplo, en Hadoop Distributed File System (HDFS), uno de los sistemas de ficheros distribuidos más utilizados, se pueden explotar las vulnerabilidades para realizar actividades maliciosas. Mothukuri et al. [28] han propuesto el sistema BlockHDFS que utiliza la plataforma empresarial Hyperledger Fabric para aprovechar los metadatos de los ficheros y construir un sistema de seguridad de datos confiable y trazable dentro de HDFS. Otras aproximaciones han integrado Inter-Planetary File System (IPFS) y Swarm con tecnologías *blockchain* para crear una nueva generación de sistemas de ficheros distribuidos fiables. En Chen et al. [29] se propuso un esquema mejorado de un sistema de ficheros P2P basado en un modelo de almacenamiento basado en zigzag para mejorar el modelo de almacenamiento por bloques que ofrece IPFS y el uso de *blockchain* para combinar IPFS con este modelo de almacenamiento.

Además, las plataformas actuales de DFS depen-

den de terceros de confianza (TTP), lo que genera falta de transparencia, seguridad e inmutabilidad. Para superar estos problemas, Naz et al. [30] han propuesto una plataforma segura de intercambio de datos basada en *blockchain*, aprovechando las ventajas del sistema de ficheros interplanetario (IPFS). En este escenario, se combinan almacenamiento descentralizado, *blockchain* de Ethereum, cifrado basado en *Shamir Secret Sharing* (SSS) y mecanismos de incentivos con contratos inteligentes. Sin embargo, como se muestra en el artículo de Huang et al. [6] aún existen retos a resolver en la actualidad.

Es de destacar que no se ha encontrado ninguna integración de PFS y *blockchain*.

III. Sistema de Ficheros Expand

En los últimos años, el sistema de ficheros paralelo Expand (XPN) [1] [2] [7] se ha rediseñado para convertirlo en un sistema de ficheros ad-hoc en su última versión [17]. Este rediseño se ha realizado principalmente para incorporar en este sistema de ficheros sus propios servidores de datos, que utilizan el almacenamiento local de los nodos de cómputo (SSD, NVMe, memoria compartida, etc.) y ejecutan en espacio de usuario, sin necesidad de permisos de superusuario.

Este rediseño ha permitido a Expand utilizar las últimas tecnologías de almacenamiento y comunicación disponibles en los entornos HPC actuales, así como poder ajustar el despliegue del sistema de ficheros según los requerimientos de las nuevas aplicaciones intensivas en el uso de datos.

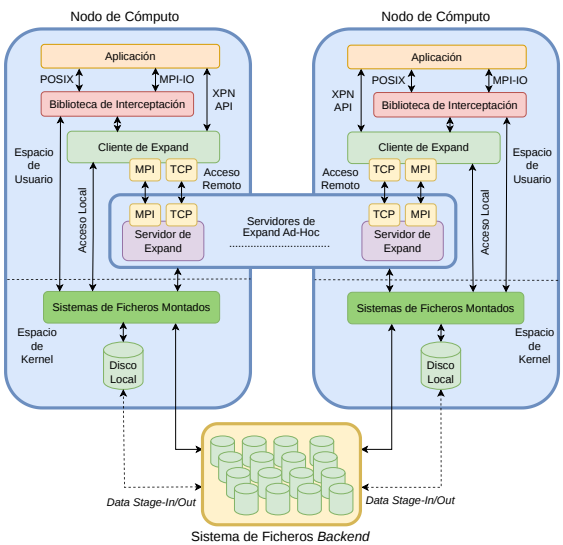


Fig. 1: Diseño de la arquitectura de Expand.

En cuanto al diseño de Expand, como se puede ver en la Figura 1, su arquitectura está basada en el modelo cliente-servidor y permite el uso de diferentes capas abstractas de comunicación para sus comunicaciones. En concreto, Expand ofrece un conector basado en *sockets* TCP, para los entornos distribuidos, y un conector basado en el estándar MPI, para los entornos HPC.

Respecto al modelo de almacenamiento de Expand, su diseño le permite crear particiones virtuales distribuidas entre los distintos servidores de forma transparente a las aplicaciones. Además, estas particiones pueden estar formadas por servidores que utilizan diferentes protocolos de comunicación, ya que los clientes de Expand se comunicarán con cada servidor utilizando el protocolo adecuado.

En cuanto a la distribución de los datos, Expand, al igual que la mayoría de los sistemas de ficheros paralelos, divide los ficheros en bloques del mismo tamaño y los distribuye entre los servidores que forman la partición virtual utilizando el patrón de reparto tipo *round-robin*. Como se puede ver en la Figura 2, este procedimiento genera un subfichero del fichero original en los servidores de datos que sea necesario. Sin embargo, los directorios siempre son replicados en todos los servidores de la partición para acelerar las funciones de búsqueda.

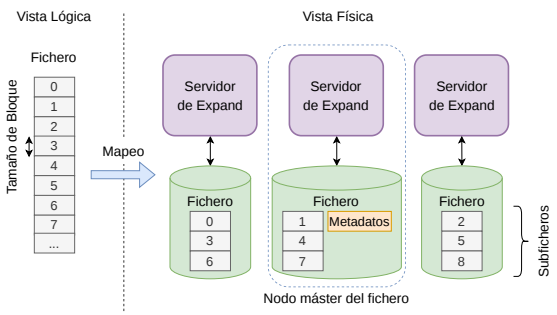


Fig. 2: Distribución de datos en Expand.

Respecto a los metadatos, Expand no utiliza un gestor de metadatos específico para simplificar el proceso de nombrado y reducir los posibles cuellos de botella en el servidor de metadatos. En este sistema de ficheros, los clientes se encargan de la gestión de metadatos y se utilizan dos capas de metadatos. En primer lugar, se encuentran los metadatos asociados al fichero (tamaño, permisos, etc.). Estos se corresponden con los metadatos POSIX y son gestionados por el sistema de ficheros local del nodo de cómputo. En segundo lugar, se encuentran los metadatos asociados a Expand (número de servidores, nodo base, etc.). Estos, en cambio, son almacenados en la cabecera que se encuentra al comienzo del subfichero del nodo máster (obtenido mediante una función *hash* del nombre del fichero), como se puede ver en la Figura 2.

Dado que Expand ha sido especialmente diseñado para ser utilizado por aplicaciones intensivas en el uso de datos, en esta nueva versión del sistema de ficheros se diseñaron diferentes optimizaciones para mejorar el rendimiento de las operaciones de E/S, como la localidad de datos. Además, para mejorar la usabilidad de este sistema de ficheros con este tipo de aplicaciones, también se diseñaron e implementaron dos nuevas funcionalidades, una biblioteca de interceptación de llamadas al sistema y técnicas eficientes

[1] https://xpn-arcos.github.io
[2] https://github.com/xpn-arcos/xpn

de *I/O data staging*.

A. Biblioteca de interceptación de llamadas al sistema

Expand ofrece una API con dos interfaces de E/S para trabajar con las particiones virtuales. Para las aplicaciones secuenciales, ofrece una interfaz basada en las llamadas al sistema POSIX, y, para las aplicaciones paralelas, una interfaz basada en MPI-IO.

Para poder utilizar esta API es necesario modificar el código de las aplicaciones y compilar la aplicación con la biblioteca de Expand. Sin embargo, en algunas ocasiones, no es posible realizar estas modificaciones en el código de las aplicaciones, por ejemplo, como ocurre con las aplicaciones ya existentes.

Por lo tanto, para tratar de solventar este problema se ha diseñado e implementado una nueva biblioteca de interceptación de llamadas al sistema en Expand. Esta biblioteca es capaz de interceptar y tratar alrededor de 60 llamadas al sistema POSIX antes de que se llame a la `libc.so` para su ejecución. Para ello, cuando se intercepta una llamada al sistema, primero, se comprueba si esta llamada utiliza Expand u otro sistema de ficheros. Si se da el primer caso, se llamará a la función de la API de Expand correspondiente para realizarla sobre este sistema de ficheros. En caso contrario, se llamará a la `libc.so` para que se ejecute la llamada sobre el sistema de ficheros correspondiente.

IV. Integración de Expand y blockchain

Como se ha visto en la Sección III, Expand presenta un diseño modular. Por lo tanto, para su integración con la tecnología *blockchain*, se ha seguido la misma filosofía de diseño, como se verá en las siguientes subsecciones.

A. Arquitectura de la integración

Para securizar el sistema de ficheros ad-hoc Expand utilizando la tecnología *blockchain*, se basa en la presentada en [31]. Esta solución consiste, por un lado, en almacenar los datos de la aplicación en Expand, aprovechando su capacidad de almacenamiento y rendimiento de E/S. Y, por otro lado, para almacenar los datos de forma segura, se genera un *hash* (SHA256 [32]) de los datos y se almacena en la red *blockchain*, garantizando su inmutabilidad y trazabilidad. El Algoritmo 1 muestra cómo se realizan la escritura y la lectura de un *buffer* de datos en el entorno integrado.

Para ello, se ha diseñado e implementado una nueva capa (Expand - *Blockchain*), como se puede ver en la Figura 3, que se encargará de escribir los datos en Expand y el *hash* en la red *blockchain* de forma transparente para la aplicación.

Este diseño modular permite utilizar *blockchain* únicamente cuando se quieran almacenar datos de forma segura en Expand, así como permitir el uso de la plataforma *blockchain* (Hyperledger Fabric [22], Ethereum [20], etc.) más adecuada para las necesidades de cada aplicación.

Algorithm 1 Algoritmo de escritura y lectura de una transacción en el entorno integrado.

Require:
Buffer $\neq$ NULL
Operación $\in \{Write, Read\}$

if Operación == Write **then**
  hash $\leftarrow$ compute_hash(Buffer)
  xpn_write(Buffer)
  blockchain_transaction(hash)
end if

if Operación == Read **then**
  xpn_read(Buffer)
  hash $\leftarrow$ compute_hash(Buffer)
  blockchain_check(hash)
end if

De esta forma, se consigue securizar Expand para almacenar datos sensibles, ya que si algún dato es modificado, el *hash* almacenado en la blockchain no coincidirá con el de los datos modificados. Además, se podrá verificar cuándo y por quién fue modificado gracias a la trazabilidad proporcionada por la tecnología *blockchain*. Por lo tanto, esta integración con *blockchain* le permite a Expand ofrecer integridad y no repudio de los datos cuando la aplicación que lo utiliza lo requiera. Así como disponer de los datos almacenados en los mismos nodos de cómputo donde serán postprocesados, evitando tener que mover los datos desde la red *blockchain* al entorno HPC, lo cual sería un proceso mucho más costoso.

Además, al usar Expand para el almacenamiento de datos en *blockchain*, se consigue aumentar significativamente el espacio disponible para almacenar los datos al poderse utilizar para ello, por ejemplo, entornos HPC, que habitualmente ofrecen un mayor espacio de almacenamiento que los dispositivos utilizados en las redes *blockchain*. De esta forma, se evita que el tamaño del bloque utilizado en la *blockchain* suponga un problema para el almacenamiento de datos de transacciones.

Por otra parte, combinar Expand y *blockchain* permite reducir el tamaño de las transacciones de *blockchain*, lo que repercutirá en el tiempo de validación de las transacciones, que también se reducirá, aumentando el rendimiento del sistema *blockchain* en sí mismo.

B. Proxy de Expand para sistemas distribuidos

Cuando se utilizan aplicaciones distribuidas, como es el caso de la *blockchain*, habitualmente estas son ejecutadas en entornos diferentes a donde se encuentran desplegados los servidores de datos de Expand. Esto se debe a que este tipo de aplicaciones, a diferencia de las aplicaciones HPC, se caracterizan por ejecutarse en entornos heterogéneos y geográficamente distribuidos.

Por otro lado, Expand habitualmente se despliega en los nodos de cómputo de un entorno HEC, que

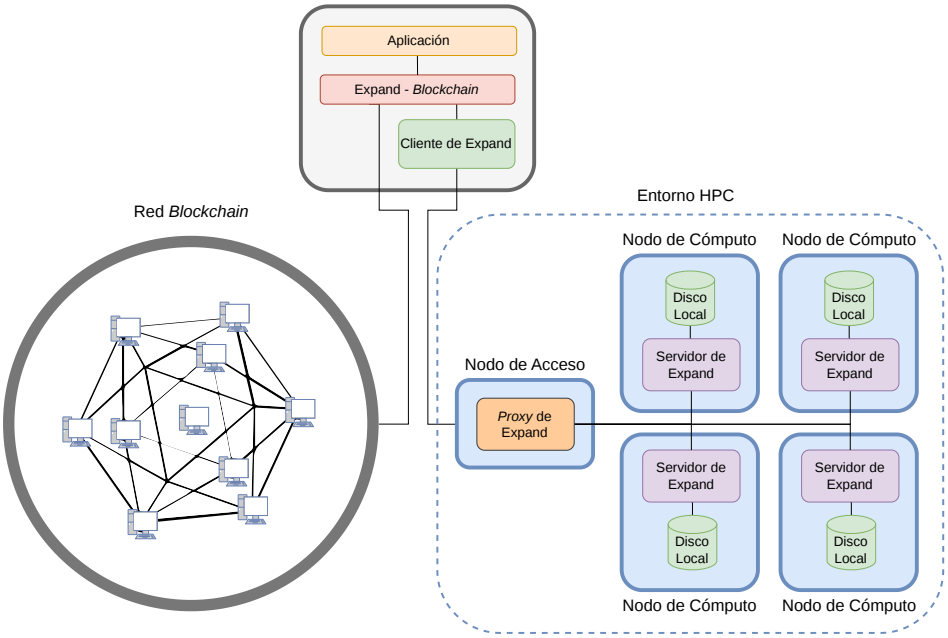


Fig. 3: Arquitectura modular de la integración.

generalmente no son accesibles desde fuera del propio entorno HPC por motivos de seguridad. Por lo tanto, el despliegue de las aplicaciones distribuidas puede ser un problema cuando estas no tienen conexión directa con los servidores de datos de Expand. Ya que las aplicaciones no pueden comunicarse con este sistema de ficheros para almacenar sus datos si se están ejecutando en un entorno distinto.

Para tratar de solventar esta limitación, se ha diseñado y desarrollado un *proxy* en Expand que actúa como intermediario entre las aplicaciones y los servidores de Expand cuando ambos se ejecutan en entornos distintos y no existe una conexión directa, como se muestra en la Figura 3. El diseño del *proxy* de Expand se basa en el patrón de diseño clásico de un *proxy* [33], en concreto, consiste en la combinación de los diseños de los clientes y servidores de Expand. Esto se debe a que, por un lado, para los clientes de Expand, el *proxy* actúa como servidor de Expand. Por otro lado, para los servidores de Expand, el *proxy* actúa como un cliente de Expand. En cuanto a su funcionalidad, el *proxy* de Expand se encarga de recibir todas las peticiones de E/S que realizan los clientes de Expand y, para cada petición, ejecutar dicha operación en los servidores de Expand en su nombre. Cabe destacar que todos estos pasos se realizan de forma transparente para la aplicación.

V. Caso de uso

Para esta prueba de concepto se utilizará como caso de uso una unidad de cuidados intensivos (UCI) de un hospital [3], ya que en este tipo de unidades se monitorizan en tiempo real las constantes vitales de los pacientes, dada la gravedad de estos. En concreto, como se puede ver en la Figura 4, cada hospital (*Hos*) dispone de varios pacientes (*Pat*) a los que se les monitoriza el pulso cardíaco (*Pulse*), la pulsioximetría (*O2*), la temperatura corporal (*Temp*) y la presión arterial (*Press*). Además, para cada paciente se definen umbrales para cada una de las constantes vitales. De esta forma, si alguna de las constantes vitales no se encuentra dentro de estos umbrales, se puede activar una alarma para que acudan los médicos.

Asimismo, en la Figura 4 también se puede ver el despliegue del sistema de ficheros Expand y de la red *blockchain*. Donde, por un lado, el sistema de ficheros Expand se ha desplegado en un entorno HPC para almacenar los datos generados por los dispositivos IoT. Y, por otro lado, la red *blockchain* se ha desplegado en varios ordenadores de propósito general, encargándose de almacenar únicamente los *hashes* de estos datos para garantizar la trazabilidad y la inmutabilidad.

Por un lado, en cuanto al sistema de ficheros Expand, este ha sido desplegado en el supercomputador C3-UC3M [34] de la Universidad Carlos III de Madrid, descrito en la Tabla I. En concreto, se han utilizado 8 nodos de cómputo y el dispositivo de almacenamiento NVMe disponible en cada uno de ellos para almacenar los datos generados por los sensores. Cabe destacar que para poder almacenar los datos en Expand se ha hecho uso del *proxy* en el nodo de acceso del supercomputador, descrito en la Sección IV, ya que por razones de seguridad el único nodo del entorno HPC que era visible al exterior era el nodo de acceso.

También cabe destacar que los parámetros de configuración utilizados en Expand son los usados por defecto. Por ejemplo, el tamaño de bloque de la partición ha sido de 512 KiB.

Por otro lado, en cuanto a la red *blockchain*, en este caso de uso se ha decidido utilizar la platafor-

[3] https://github.com/xpn-arcos/xpn-blockchain

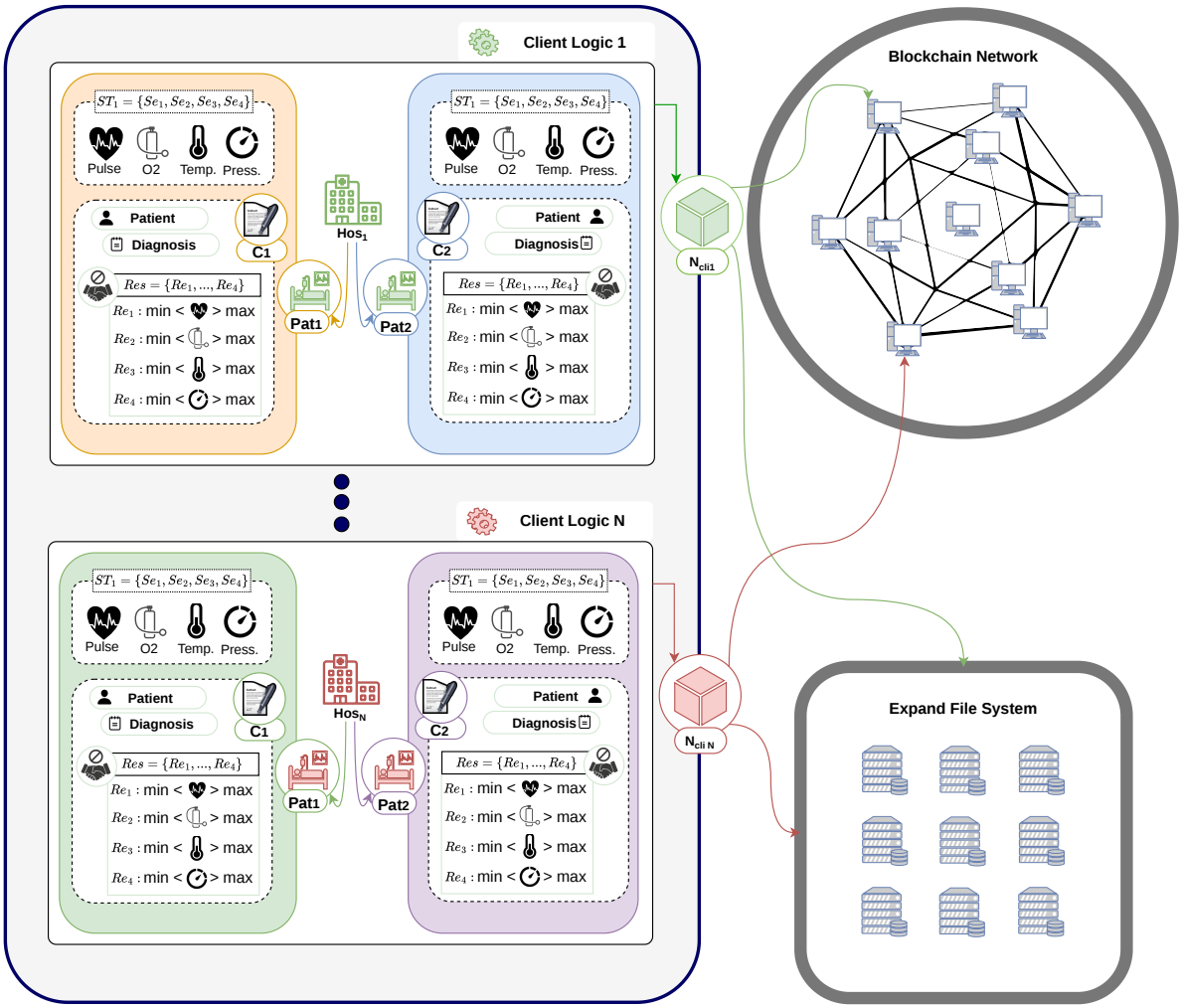


Fig. 4: Arquitectura general del caso de uso de Expand con *blockchain*.

Tabla I: Especificaciones del supercomputador C3-UC3M.

Atributo	**C3-UC3M** [34]
Número de nodos	90 nodos
Proveedor	Dell PowerEdge R6525
CPU por nodo	2 x AMD EPYC 7713 64 procesadores, a 2 GHz
NVMe por nodo	3.5 TiB
RAM por nodo	1 TiB DDR4
Red	NVIDIA Mellanox Infiniband HDR 100 Gbps
Sistema operativo	Rocky Linux 8.6 (Green Obsidian)
Distribución MPI	MPICH 4.3.0
Versión GCC	12.2.0
Versión SLURM	24.11.4

ma Hyperledger Fabric (v2.5) [22]. Esta plataforma utiliza redes privadas, es decir, los participantes de la red se conocen entre sí, a diferencia de las redes públicas, en las que los participantes son anónimos. Esto permite utilizar protocolos de consenso que requieren una potencia de cómputo mucho menor que la empleada en las redes públicas, que suelen basarse en pruebas de trabajo (PoW). Por ejemplo, los protocolos utilizados habitualmente en Hyperledger Fabric son *Crash Fault-Tolerant* (CFT), como Raft [35] y Paxos [36], y *Byzantine Fault Tolerant* (BFT) [37]. Esto ha permitido utilizar tres ordenadores de propósito general para desplegar la red *blockchain* del caso de uso presentado.

En concreto, en este caso de uso se ha simulado un hospital con 150 pacientes monitorizados durante 12 horas, utilizando, por un lado, únicamente la tecnología *blockchain* y, por otro, la tecnología *blockchain* junto con Expand para almacenar los datos. Esto permitirá comparar ambos despliegues y verificar que el uso de Expand junto con la tecnología *blockchain* resulta viable.

Tras realizar dicha simulación, se ha podido ver que el tiempo medio necesario para realizar una transacción utilizando únicamente *blockchain* para el almacenamiento es de 0,143 segundos. Mientras que si se utiliza *blockchain* junto con Expand para el almacenamiento, el tiempo medio para realizar una transacción es de 0,243 segundos. Como se puede ver, el tiempo medio necesario para realizar una transacción es muy similar en ambos casos, ya que el tamaño de los datos a almacenar es pequeño. Además, cabe destacar que durante la simulación de 12 horas de un único hospital con 150 pacientes se han generado 2,6

GiB de datos.

Este comportamiento se debe a dos razones principales. Por un lado, todos los ordenadores de propósito general utilizados para desplegar la red *blockchain* se encuentran en la misma red, mientras que el supercomputador C3-UC3M utilizado para almacenar Expand se encuentra en una red distinta y altamente securizada. Esto hace que exista una mayor latencia al enviar los datos a este supercomputador. Por otro lado, cuando se utiliza únicamente *blockchain*, solo se realiza una operación sobre la *blockchain* para almacenar los datos, mientras que en el caso de utilizar Expand hay que operar tanto en Expand como en la *blockchain*.

En resumen, con esta prueba de concepto se ha podido ver, por un lado, que, desde el punto de vista de Expand, su uso junto con la tecnología *blockchain* permite asegurar este sistema de ficheros para almacenar datos críticos, sin incrementar significativamente el tiempo de procesamiento de cada transacción, lo cual es el objetivo de esta prueba de concepto. Por lo tanto, su uso conjunto permite incorporar a Expand dos de los cinco principios de la *CIA triad* [38]: la integridad y el no repudio.

Por otro lado, desde el punto de vista de la *blockchain*, Expand aporta un incremento en el espacio de almacenamiento disponible para almacenar transacciones. Además, al almacenar los datos en el entorno HPC, se puede realizar su postprocesado de forma más eficiente y aprovechando las capacidades del entorno HPC. Aunque la prueba de concepto presentada se acerca mucho a la realidad, habitualmente en un caso real, se almacenan otros parámetros, como electrocardiogramas (ECG), radiografías (RX) y tomografías computadas (CT), cuyo tamaño es significativamente mayor que el de los datos generados por los sensores IoT de esta prueba de concepto.

VI. Conclusiones

En este trabajo se ha presentado y analizado una nueva solución para securizar el sistema de ficheros Expand mediante la tecnología *blockchain*. Permitiendo que este sistema de ficheros también pueda ser utilizado por aplicaciones que generan y trabajan con datos críticos.

Para ello, se ha estudiado la viabilidad de esta solución utilizando como caso de uso una unidad de cuidados intensivos (UCI) de un hospital, donde se monitorizan en tiempo real las constantes vitales de múltiples pacientes. Con esta prueba de concepto se ha comprobado que es viable el uso de Expand junto con la tecnología *blockchain*, en concreto, con la plataforma Hyperledger Fabric, para añadir integridad y no repudio a Expand, lo que hace que este sistema sea seguro para almacenar datos críticos. Además de la securización, la solución propuesta también permite incrementar el almacenamiento disponible para los datos, que es uno de los principales problemas de la tecnología *blockchain*.

Como trabajos futuros, se quiere estudiar el uso de otras plataformas *blockchain*, como Ethereum, para securizar Expand. También se quiere analizar y evaluar esta nueva funcionalidad de Expand, utilizando el conector basado en MQTT para enviar los datos de los dispositivos IoT a Expand. Por último, también se quieren estudiar otras metodologías para la gestión de los *hashes* y mejorar el rendimiento.

Agradecimientos

Este trabajo ha sido parcialmente apoyado por el proyecto "Expand: High Performance Storage System for HPC and Big Data Environments"(Ref. TED2021-131798B-I00) y por el proyecto "New scalable I/O techniques for hybrid HPC and data intensive workloads (SCIOT)" (Ref. PID2022-138050NB-I00), ambos financiados por la Agencia Estatal de Investigación.

Referencias

[1] Schahram Dustdar, Victor Casamayor Pujol, and Praveen Kumar Donta, "On distributed computing continuum systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 4092–4105, 2023.

[2] Satoshi Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Available at SSRN 3440802*, 2008.

[3] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *2017 IEEE International Congress on Big Data (BigData Congress)*, 2017, pp. 557–564.

[4] Bhabendu Kumar Mohanta, Soumyashree S Panda, and Debasish Jena, "An overview of smart contract and use cases in blockchain technology," in *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2018, pp. 1–4.

[5] Cristhian Martinez-Rendon, Diego Camarmas-Alonso, Jesus Carretero, and Jose L Gonzalez-Compean, "On the continuous contract verification using blockchain and real-time data," *Cluster Computing*, vol. 25, no. 3, pp. 2179–2201, 2022.

[6] Huawei Huang, Jianru Lin, Baichuan Zheng, Zibin Zheng, and Jing Bian, "When blockchain meets distributed file systems: An overview, challenges, and open issues," *IEEE Access*, vol. 8, pp. 50574–50586, 2020.

[7] Félix Garcia-Carballeira, Alejandro Calderon, Jesus Carretero, Javier Fernandez, and Jose M Perez, "The design of the Expand parallel file system," *The International Journal of High Performance Computing Applications*, vol. 17, no. 1, pp. 21–37, 2003.

[8] Ning Liu, Jason Cope, Philip H. Carns, Christopher D. Carothers, Robert B. Ross, Gary Grider, Adam Crume, and Carlos Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST) , April 16-20, Asilomar Conference Grounds, Pacific Grove, CA, USA*, 2012, pp. 1–11.

[9] Donghun Koo, Jaehwan Lee, Jialin Liu, Eun-Kyu Byun, Jae-Hyuck Kwak, Glenn K. Lockwood, Soonwook Hwang, Katie Antypas, Kesheng Wu, and Hyeonsang Eom, "An empirical study of I/O separation for burst buffers in HPC systems," *J. Parallel Distributed Comput.*, vol. 148, pp. 96–108, 2021.

[10] Sarp Oral, Sudharshan S. Vazhkudai, Feiyi Wang, Christopher Zimmer, Christopher Brumgard, Jesse Hanley, George Markomanolis, Ross G. Miller, Dustin Leverman, Scott Atchley, and Verónica G. Vergara Larrea, "End-to-end I/O portfolio for the summit supercomputing ecosystem," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Denver, Colorado, USA, November 17-19*, 2019, pp. 63:1–63:14.

[11] Michael J. Brim, Adam T. Moody, Seung-Hwan Lim, Ross G. Miller, Swen Boehm, Cameron Stanavige, Kathryn M. Mohror, and Sarp Oral, "Unifyfs: A user-level shared file system for unified access to distributed local storage," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS), St. Petersburg, FL, USA, May 15-19*, 2023, pp. 290–300.

[12] Florin Isaila, Javier Garcia-Blas, Jesus Carretero, Weikeng Liao, and Alok N. Choudhary, "A scalable message passing interface implementation of an ad-hoc parallel i/o system," *Int. J. High Perform. Comput. Appl.*, vol. 24, no. 2, pp. 164–184, 2010.

[13] Teng Wang, Kathryn M. Mohror, Adam Moody, Kento Sato, and Weikuan Yu, "An ephemeral burst-buffer file system for scientific applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Salt Lake City, UT, USA, November 13-18*, 2016, pp. 807–818.

[14] Sebastian Oeste, Marc-André Vef, Mehmet Soysal, Wolfgang E. Nagel, André Brinkmann, and Achim Streit, "ADA-FS - advanced data placement via ad hoc file systems at extreme scales," in *Software for Exascale Computing - SPPEXA 2016-2019*, pp. 29–59. Springer, 2020.

[15] Marc-Andre Vef, Nafiseh Moti, Tim Süß, Markus Tacke, Tommaso Tocci, Ramon Nou, Alberto Miranda, Toni Cortes, and André Brinkmann, "Gekkofs - A temporary burst buffer file system for HPC applications," *J. Comput. Sci. Technol.*, vol. 35, no. 1, pp. 72–91, 2020.

[16] Shawn Zhong, Chenhao Ye, Guanzhou Hu, Suyan Qu, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, and Michael Swift, "MadFS: Per-File virtualization for userspace persistent memory filesystems," in *21st USENIX Conference on File and Storage Technologies (FAST 23)*, Santa Clara, CA, Feb. 2023, pp. 265–280, USENIX Association.

[17] Diego Camarmas-Alonso, Felix Garcia-Carballeira, Alejandro Calderon-Mateos, and Jesus Carretero, "Improving I/O performance in HPC environments using the Expand Ad-Hoc file system," *The Journal of Supercomputing*, vol. 81, no. 16, pp. 1541, 2025.

[18] Frank Herold and Sven Breuner, "An introduction to BeeGFS, v 2.0," Tech. Rep., ThinkParQ, Germany, June 2018.

[19] Javier García Blas, Genaro Sanchez-Gallegos, Cosmin Petre, Alberto Riccardo Martinelli, Marco Aldinucci, and Jesús Carretero, "Hercules: Scalable and network portable in-memory ad-hoc file system for data-centric and high-performance applications," in *29th International European Conference on Parallel and Distributed Computing (Euro-Par), Limassol, Cyprus, August 28 - September 1*, 2023, pp. 679–693.

[20] Vitalik Buterin et al., "Ethereum white paper," *GitHub repository*, vol. 1, no. 22-23, pp. 5–7, 2013.

[21] Dejan Vujičić, Dijana Jagodić, and Siniša Randjić, "Blockchain technology, bitcoin, and ethereum: A brief overview," in *2018 17th international symposium infoteh-jahorina (infoteh)*. IEEE, 2018, pp. 1–6.

[22] Elli Androulaki, Artem Barger, Vita Bortnikov, and more, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, New York, NY, USA, 2018, EuroSys '18, Association for Computing Machinery.

[23] Xiaoqiong Xu, Gang Sun, Long Luo, Huilong Cao, Hongfang Yu, and Athanasios V. Vasilakos, "Latency performance modeling and analysis for hyperledger fabric blockchain network," *Information Processing & Management*, vol. 58, no. 1, pp. 102436, 2021.

[24] Sunny Pahlajani, Avinash Kshirsagar, and Vinod Pachghare, "Survey on private blockchain consensus algorithms," in *2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT)*. IEEE, 2019, pp. 1–6.

[25] Thanh Son Lam Nguyen, Guillaume Jourjon, Maria Potop-Butucaru, and Kim Loan Thai, "Impact of network delays on hyperledger fabric," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 222–227.

[26] Yuchen Wang, Shuang Li, Lei Xu, and Lizhen Xu, "Improved raft consensus algorithm in high real-time and highly adversarial environment," in *International Conference on Web Information Systems and Applications*. Springer, 2021, pp. 718–726.

[27] Wei Fu, Xuefeng Wei, and Shihua Tong, "An improved blockchain consensus algorithm based on raft," *Arabian Journal for Science and Engineering*, vol. 46, no. 9, pp. 8137–8149, 2021.

[28] Viraaji Mothukuri, Sai S Cheerla, Reza M Parizi, Qi Zhang, and Kim-Kwang Raymond Choo, "Blockhdfs: Blockchain-integrated hadoop distributed file system for secure provenance traceability," *Blockchain: Research and Applications*, vol. 2, no. 4, pp. 100032, 2021.

[29] Yongle Chen, Hui Li, Kejiao Li, and Jiyang Zhang, "An improved p2p file system scheme based on ipfs and blockchain," in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 2652–2657.

[30] Muqaddas Naz, Fahad A Al-Zahrani, Rabiya Khalid, Nadeem Javaid, Ali Mustafa Qamar, Muhammad Khalil Afzal, and Muhammad Shafiq, "A secure data sharing platform using blockchain and interplanetary file system," *Sustainability*, vol. 11, no. 24, pp. 7054, 2019.

[31] Javier Alonso-Núñez, Daniel López-Martínez, and Diego R. Llanos, "Arquitectura segura para la trazabilidad basada en IoT y blockchain," June 2025.

[32] National Institute of Standards and Technology (NIST), "FIPS PUB 180-4: Secure Hash Standard (SHS)," Federal Information Processing Standards Publication 180-4, U.S. Department of Commerce, Mar. 2015, Fecha de publicación: 4 de agosto de 2015.

[33] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design patterns: elements of reusable object-oriented software*, Pearson Deutschland GmbH, 1995.

[34] UC3M, "C3-UC3M specification," https://c3-uc3m.github.io/c3-web. Accedido el 07-04-2026. [Online], 2026.

[35] Dongyan Huang, Xiaoli Ma, and Shengli Zhang, "Performance analysis of the raft consensus algorithm for private blockchains," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 1, pp. 172–181, 2020.

[36] Leslie Lamport, "Paxos made simple," *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pp. 51–58, 2001.

[37] Miguel Castro and Barbara Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, Nov. 2002.

[38] Spyridon Samonas and David Coss, "The cia strikes back: Redefining confidentiality, integrity and availability in security.," *Journal of Information System Security*, vol. 10, no. 3, 2014.

Aplicaciones de la computación de altas prestaciones (1/3)

Combinando Escalado en Frecuencia, Procesamiento Multihilo y Vectorización para una Filogenética más Sostenible

Sergio Santander-Jiménez[1] y Miguel A. Vega-Rodríguez[1]

Resumen— **La definición de aproximaciones paralelas energéticamente eficientes para la resolución de problemas reales representa uno de los intereses fundamentales hoy en día en la investigación en computación paralela. En ocasiones, el uso de tecnologías hardware como el ajuste de frecuencias no es suficiente para conseguir entornos de ejecución que alcancen buenos resultados desde la perspectiva conjunta de rendimiento y consumo energético, por lo que se precisa el diseño de estrategias algorítmicas paralelas adecuadas. Este trabajo explora la combinación de escalado en frecuencia, estrategias de procesamiento multihilo con OpenMP y vectorización basada en intrínsecas AVX512 para mejorar el consumo energético de un problema importante de bioinformática, la inferencia filogenética. Los resultados experimentales obtenidos en cuatro conjuntos de datos reales dan cuenta de la consecución de mejoras energéticas de hasta 20× al abordar tamaños del problema complejos en CPU. Las comparativas con otras aproximaciones paralelas y métodos de la literatura demuestran el interés de las estrategias investigadas para mejorar la sostenibilidad de los análisis filogenéticos.**

Palabras clave— **Computación paralela, procesamiento multihilo, vectorización, escalado en frecuencia, eficiencia energética, bioinformática.**

I. Introducción

La consecución de ganancias en rendimiento paralelo representa uno de los principales hilos conductores de la Computación de Altas Prestaciones. No obstante, la presencia de cargas de trabajo cada vez más complejas y con mayores requisitos energéticos ha motivado un interés creciente en el desarrollo de nuevas soluciones orientadas a la consecución de objetivos de eficiencia energética [1]. Una de las prácticas más habituales para mejorar el consumo energético reside en el empleo de técnicas de escalado en frecuencia [2], las cuales permiten ajustar las frecuencias operativas de las arquitecturas como medio para gestionar su uso de potencia. En este sentido, es preciso considerar que las características de las aplicaciones imponen limitaciones en las mejoras energéticas que se pueden obtener con esta técnica [3], [4]. Por ello, maximizar el rendimiento por vatio exige combinar estas estrategias hardware con mejoras algorítmicas en el lado de la aplicación [5]. La definición de optimizaciones algorítmicas eficientes, que incluyen el desarrollo de esquemas paralelos y vectoriales adecuados, resulta clave para alcanzar soluciones sostenibles de alto rendimiento [6].

Uno de los dominios de aplicación caracterizados por consumos energéticos elevados y cargas de trabajo complejas es la bioinformática. Dentro de ella, destaca el problema de la inferencia filogenética, que persigue el estudio de relaciones evolutivas entre organismos [7]. Dada la complejidad del problema, la investigación en métodos filogenéticos se ha centrado en la definición de algoritmos que únicamente perseguían acelerar su resolución sin considerar aspectos energéticos, dando lugar así a emisiones de hasta 3.565 kgCO2e en escenarios experimentales reales [8]. El estudio de métodos energéticamente eficientes en este problema se ha limitado principalmente a la propuesta de soluciones basadas en FPGAs [9], [10], quedando pues abierta esta cuestión investigadora en el caso de dispositivos de propósito general como las CPUs. Para cumplir objetivos de sostenibilidad en este ámbito, es por tanto necesario abordar la propuesta de estrategias para permitir análisis filogenéticos energéticamente eficientes.

Este trabajo explora la combinación de aproximaciones algorítmicas y escalado en frecuencia en CPUs para mejorar la eficiencia energética de los cálculos filogenéticos. Como caso de estudio, nos centraremos en el algoritmo de Fitch [11], uno de los métodos principales empleados en filogenética para evaluar la calidad de las soluciones. Concretamente, planteamos un diseño paralelo de este algoritmo que integra estrategias multihilo con OpenMP y vectorización basada en intrínsecas para explotar las capacidades de cómputo de las CPUs. Se estudia el escalado en frecuencia en distintos escenarios de ejecución a fin de identificar configuraciones que, conjugadas con el diseño propuesto, permitan alcanzar mejoras energéticas efectivas. Para evaluar la propuesta se emplearán cuatro conjuntos de datos reales, examinando los resultados obtenidos atendiendo a distintas métricas, tales como el tiempo de ejecución, la potencia, el consumo energético y la eficiencia energética.

El resto de este artículo está organizado del siguiente modo. La Sección II explica las bases del problema y los fundamentos del algoritmo de Fitch. La Sección III describe las estrategias algorítmicas y de escalado en frecuencia planteadas para resolver el problema. La Sección IV presenta los resultados experimentales obtenidos. Finalmente, la Sección V aporta conclusiones y líneas de trabajo futuro.

II. Formulación del problema

Los árboles filogenéticos se emplean para representar las relaciones evolutivas de un conjunto de

[1]Universidad de Extremadura (https://ror.org/0174shg90), Departamento de Tecnología de los Computadores y de las Comunicaciones, Campus Universitario s/n, Cáceres 10003, España, e-mail: sesaji@unex.es, mavega@unex.es.

N organismos de entrada, cada uno caracterizado por una secuencia biológica de M caracteres. Cada carácter en estas secuencias toma un valor de estado conforme a un alfabeto λ (por ejemplo, el alfabeto de nucleótidos $\lambda = \{A, C, G, T\}$ extendido con estados combinados, desconocidos y gaps de alineamiento). El árbol filogenético $T = (V, E)$ presenta en su conjunto de nodos V dos tipos: nodos hoja y nodos internos. Los nodos hoja representan los organismos caracterizados en las secuencias de entrada, mientras que los nodos internos denotan ancestros hipotéticos. Así, las ramas del conjunto E representan las relaciones ancestro-descendiente entre nodos.

En filogenética se suelen emplear criterios de optimalidad para distinguir la calidad de las filogenias. Entre ellos, la función de parsimonia, que busca el árbol filogenético que minimice el número de mutaciones entre organismos emparentados, representa uno de los criterios más empleados en distintos contextos biológicos, como es el caso del estudio de variantes del virus SARS-CoV-2 [12]. Para identificar las mutaciones asociadas a una filogenia, uno de los métodos más utilizados es el algoritmo de Fitch [11], el cual procesa el árbol desde las hojas a la raíz, calculando para cada nodo su conjunto de estados F. En el caso de un nodo hoja l, su conjunto $F(l)$ coincide con la secuencia que la caracteriza (esto es, la l-ésima secuencia de entrada). En el caso de un nodo interno a con hijos b y c, se calculan sus estados ancestrales para cada carácter i, $F_i(a)$, usando los valores observados en sus hijos para dicho carácter, tal y como se muestra en la Ecuación 1:

$$F_i(a) = \begin{cases} F_i(b) \cap F_i(c) & \text{si } F_i(b) \cap F_i(c) \neq \emptyset, \\ F_i(b) \cup F_i(c) & \text{en caso contrario.} \end{cases} \quad (1)$$

Estos cálculos se efectúan para cada carácter $i = 1$ hasta M. En términos biológicos, aquellas situaciones donde $F_i(b) \cap F_i(c) \neq \emptyset$ denotan que los nodos hijos b y c han heredado el estado ancestral i-ésimo del nodo padre a en el evento de especiación. Sin embargo, cuando $F_i(b) \cap F_i(c) = \emptyset$, esto indica que al menos uno de los hijos ha sufrido una mutación, debiéndose incrementar en ese caso el valor de parsimonia del árbol. En consecuencia, el valor de parsimonia $P(T)$ puede calcularse como indica la Ecuación 2:

$$P(T) = \sum_{i=1}^{M} \sum_{a \in V} \textit{Mutación}(F_i(a)), \quad (2)$$

donde $\textit{Mutación}(F_i(a))$ es 0 si a es un nodo hoja o si F_i procede de una intersección no vacía. En cambio, $\textit{Mutación}(F_i(a))$ será 1 en caso de que F_i haya sido generado mediante una unión, lo cual denota la existencia de mutación.

La Figura 1 muestra un ejemplo gráfico del algoritmo de Fitch. En este algoritmo, es preciso tener en cuenta que, para calcular los estados asociados a un determinado nodo, se requiere calcular previamente los estados de los hijos de dicho nodo. Esto es, el algoritmo presenta dependencias de datos a nivel de

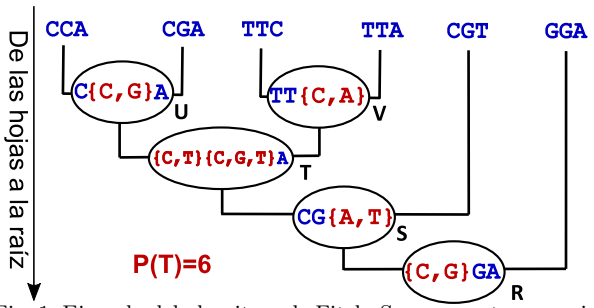


Fig. 1: Ejemplo del algoritmo de Fitch. Se representan en rojo las operaciones de unión que denotan la existencia de mutación e implican un aumento de la parsimonia $P(T)$.

procesamiento del árbol. Por otra parte, el cálculo de los estados asociados a un determinado carácter i se realiza de manera independiente al resto de caracteres, al no mostrar el procedimiento dependencias con respecto a otros caracteres de la misma secuencia. Estos cálculos pueden ser consecuentemente paralelizados, de tal manera que la carga de trabajo puede distribuirse entre los hilos de ejecución disponibles. Es más, cada hilo puede procesar simultáneamente distintos grupos o *chunks* de caracteres almacenados mediante tipos vectoriales. Así, se puede definir un esquema multihilo y vectorizado donde cada hilo sea responsable del cálculo de valores parciales de parsimonia para los *chunks* que le correspondan.

III. Descripción del método

A continuación, se detallan las estrategias paralelas y de escalado en frecuencia consideradas para mejorar los cálculos de parsimonia mediante el algoritmo de Fitch. Antes de proceder con la explicación del diseño multihilo vectorizado, es necesario tener en cuenta que nuestra implementación considera una serie de optimizaciones generales. En primer lugar, las operaciones de intersección y unión contempladas en el algoritmo se llevarán a cabo mediante operaciones AND y OR a nivel de bit, respectivamente. Para ello, consideramos una representación hexadecimal para codificar los valores de estado de carácter. Esta representación reserva un bit separado a cada posible valor de nucleótido, codificando 'A' como 0x08, 'C' como 0x04, 'G' como 0x02 y 'T' como 0x01. En el caso de combinaciones de nucleótidos, habilitamos a 1 los bits asociados a cada nucleótido que forma parte de la combinación (por ejemplo, el estado 'M', que representa 'A' o 'C', se codifica como 0x0C). Los estados desconocidos '?' se codifican como 0x0F y los gaps del alineamiento '-' como 0x10, representando un quinto estado diferente del resto.

Bajo esta codificación, almacenaremos las secuencias de entrada utilizando un array de caracteres compartido por todos los hilos, organizado por columnas (*column-major*) para facilitar la vectorización y mejorar las transferencias de datos en la CPU. Además, se emplea un array de caracteres privado a cada hilo para almacenar los estados ancestrales F calculados durante la ejecución del algoritmo, de tal manera que la posición i-ésima de dicho array contendrá los estados calculados por el hilo para el

Algoritmo 1 Diseño OpenMP+AVX512 del Algoritmo de Fitch

Entrada: *árbol*: estructura array del árbol filogenético, *n_nodos*: número de nodos internos, *secuencias*: array de secuencias, *n_secuencias*: número de secuencias, *n_chunks*: número de *chunks* de caracteres de secuencia, *n_hilos*: número de hilos.
Salida: $P(T)_{acc}$: valor de parsimonia.

```
1:  #pragma omp parallel num_threads (n_hilos) {
2:  tid ← omp_get_thread_num()
3:  P(T)_tid ← 0, F_tid ← 0
4:  ceros ← _mm512_setzero_si512()
5:  unos ← _mm512_set1_epi8(0x01)
6:  #pragma omp for schedule(guided)
7:  para i = 1 hasta n_chunks hacer
8:      para j = 1 hasta n_nodos hacer
9:          valor_nodo ← _mm512_set1_epi8(0xFF) /* inicializando valor de estado */
10:         secuencias_ptr ← &secuencias[n_secuencias×i]
11:         para k = 1 hasta árbol[j].n_hijos hacer
12:             hijo_tipo ← árbol[j].hijo_id[k] & 0x80000000
13:             hijo_id ← árbol[j].hijo_id[k] & 0x7FFFFFFF
14:             si hijo_tipo = 0x80000000 entonces
15:                 /* Hijo interno, leer valores de F_tid */
16:                 valor_hijo ← _mm512_load_si512(F_tid[hijo_id])
17:             si no
18:                 /* Hijo hoja, leer valores de las secuencias de entrada */
19:                 valor_hijo ← _mm512_load_si512(secuencias_ptr[hijo_id])
20:             fin si
21:             /* Realizar operaciones de Fitch */
22:             intersecciones ← _mm512_and_si512(valor_nodo, valor_hijo)
23:             uniones ← _mm512_or_si512(valor_nodo, valor_hijo)
24:             /* Determinar caracteres con intersecciones vacías */
25:             mask_cmp ← _mm512_cmpeq_epi8_mask(intersecciones, ceros)
26:             si_cero ← _mm512_movm_epi8(mask_cmp)
27:             pos_vacías ← _mm512_and_si512(si_cero, unos)
28:             /* Cada intersección vacía aporta +1 a P(T)_tid */
29:             pos_vacías ← _mm512_sad_epu8(ceros, pos_vacías)
30:             P(T)_tid ← P(T)_tid + _mm512_reduce_add_epi32(pos_vacías)
31:             /* Calcular valores de estado combinando intersecciones y uniones */
32:             uniones ← _mm512_and_si512(si_cero, uniones)
33:             valor_nodo ← _mm512_or_si512(intersecciones, uniones)
34:         fin para
35:         F_tid[j] ← _mm512_store_si512(valor_nodo)
36:     fin para
37: fin para
38: P(T)_acc ← Acumular valores parciales de parsimonia (P(T)_tid) /* para cada hilo tid */
39: }
40: return P(T)_acc
```

i-ésimo nodo interno.

En lo que respecta a la estructura del árbol filogenético, se usa un array denotado como *árbol* para almacenar la información necesaria para procesar la topología de la filogenia. Este array concretamente codifica los nodos internos del árbol de acuerdo a su recorrido postorden, almacenando para cada nodo: 1) cuántos nodos hijos tiene (*n_hijos*) y 2) los identificadores de sus nodos hijos (*hijo_id*). Estos identificadores de tipo `int` actuarán como indexadores para leer los estados de los hijos desde las secuencias de entrada (en caso de que el hijo sea un nodo hoja) o bien desde el array F (en caso de que el hijo sea un nodo interno). La distinción entre hijos hojas o internos se aborda asignando un 0 (hoja) o un 1 (interno) al bit más significativo de sus identificadores.

A. Diseño multihilo vectorizado

El procesamiento paralelo en el diseño propuesto se gestiona mediante la combinación de directivas multihilo de OpenMP e intrínsecas SIMD para vectorizar cálculos de grano fino. En concreto, empleamos estrategias de vectorización AVX512, lo cual permite el procesamiento de las operaciones de Fitch considerando 64 nucleótidos cargados en un tipo vectorial `__m512i`. Así, las secuencias de entrada se organizarán en términos de *chunks*, cada uno agrupando en un tipo vectorial 64 caracteres de secuencia consecutivos.

El Algoritmo 1 muestra el diseño paralelo propuesto. Los primeros pasos del algoritmo involucran la definición de una región paralela OpenMP mediante la directiva `#pragma omp parallel` y la inicialización de las variables privadas de cada hilo *tid* correspondientes a su parsimonia parcial $P(T)_{tid}$ y a su array de estados ancestrales F_{tid} (líneas 1–3 en el Algoritmo 1). Dado que el algoritmo se basa en la identificación de intersecciones vacías para incrementar el valor de parsimonia, se definen dos vectores de tipo `__m512i`, etiquetados como *ceros* y *unos*, que serán utilizados en los cálculos SIMD de Fitch (líneas 4–5). Tras ello, aplicamos la directiva `#pragma omp for` para paralelizar los cálculos asociados a diferentes *chunks* de caracteres, usando la política de planificación `guided` para facilitar el equilibrado de carga entre hilos (líneas 6–7).

Dentro del bucle paralelo, cada hilo lleva a cabo el procesamiento del array de nodos internos que representa al árbol, calculando para cada nodo su conjunto de estados a partir de los estados de sus hijos (líneas 8–36). Dado un nodo j, el algoritmo realiza la inicialización de un vector *valor_nodo* de tipo `__m512i`, que servirá para depositar los estados a calcular para j sobre el *chunk* asignado al hilo, mediante la intrínseca `_mm512_set1_epi8` (línea 9). Tras ello, se realiza la lectura de los estados del hijo k-ésimo de j desde el array de secuencias o desde F_{tid}, según el tipo de hijo tratado (hoja o interno). Estos estados son cargados en un vector *valor_hijo* de tipo `__m512i`

mediante `_mm512_load_si512` (líneas 10–20).

El algoritmo calcula las intersecciones y uniones de Fitch sobre los estados actuales del nodo y los cargados del hijo mediante `_mm512_and_si512` y `_mm512_or_si512`, respectivamente (líneas 22–23). De cara a poder identificar qué caracteres están asociados a intersecciones vacías, se realiza una comparación con cero sobre cada elemento de 8 bits del vector de intersecciones usando `_mm512_cmpeq_epi8_mask` (línea 25). Esta intrínseca devuelve como resultado una máscara de 64 bits que contendrá un 1 en aquellas posiciones con intersección vacía, permitiendo identificar cuáles caracteres motivan un incremento de la parsimonia parcial $P(T)_{tid}$. Se hace uso de intrínsecas de baja latencia como `_mm512_movm_epi8` y `_mm512_and_si512` para convertir esta máscara en un vector `__m512i` etiquetado como *pos_vacías* (líneas 26–27), el cual puede utilizarse para actualizar $P(T)_{tid}$ con una reducción. Con este fin, aplicamos `_mm512_sad_epu8` para realizar una primera suma horizontal sobre *pos_vacías*, empaquetando los resultados en la parte baja (16 bits) de ocho elementos de 64 bits (línea 29). Esta disposición es equivalente a 16 elementos de 32 bits, con los elementos pares conteniendo el valor 0. Así, podemos aplicar la intrínseca AVX512 de reducción `_mm512_reduce_add_epi32` y utilizar su resultado para actualizar $P(T)_{tid}$ (línea 30). Finalmente, los estados del nodo son actualizados extrayendo las uniones para las posiciones con intersección vacía mediante `_mm512_and_si512` y combinándolas con el vector de intersecciones usando `_mm512_or_si512` (líneas 32–33).

Los pasos anteriores se repiten para cada nodo hijo, almacenando los valores de estado finales del nodo en F_{tid} a través de `_mm512_store_si512` (línea 35). Una vez todos los nodos internos han sido procesados para el *chunk* actual, el hilo procede con los cálculos asociados a su siguiente *chunk*. Al terminar, los valores parciales de parsimonia $P(T)_{tid}$ calculados por cada hilo se suman y se devuelven en la salida del algoritmo (líneas 38–40).

B. Escalado en frecuencia

En lo que respecta a las técnicas de escalado en frecuencia, en este trabajo gestionamos las frecuencias en entornos de ejecución CPU utilizando el software `cpupower`. Esta herramienta de Linux proporciona un conjunto de utilidades para fijar las frecuencias operativas de la CPU bajo distintos algoritmos de escalado (`performance`, `schedutil`, etc.). En nuestro caso, el software interactúa con el driver de escalado propietario de Intel, `intel_pstates`. Las frecuencias deseadas se establecen utilizando el comando `cpupower frequency-set`. El algoritmo de escalado utilizado es `performance` con el objetivo de promover que los cores trabajen a la frecuencia establecida por el comando anterior durante la ejecución del método paralelo propuesto, cuando sea posible.

Para medir el consumo energético de la CPU, empleamos la tecnología *Running Average Power Limit* (RAPL) [13]. RAPL permite la monitorización de consumo energético en CPU mediante la combinación de contadores del procesador y modelos software de estimación. También facilita datos de consumo medidos mediante reguladores de voltaje on-chip en el caso de CPUs modernas. Es posible acceder a los ficheros generados por RAPL a través de la interfaz `sysfs powercap`, leyendo de ellos la energía en microjulios consumida por distintos componentes (dominios) del sistema. En este trabajo, la monitorización energética se ha realizado sobre el socket completo (dominio *package*), junto con la memoria principal (dominio *dram*).

IV. Resultados experimentales

A fin de realizar un estudio experimental detallado sobre un rango diverso de tamaños del problema, se han considerado en la evaluación del método los siguientes conjuntos de datos reales:

- M50×42456: 50 secuencias de *Enterobacteriaceae*, con 42456 caracteres por secuencia [14].
- M156×119750: 156 secuencias de *Salmonella enterica*, con 119750 caracteres por secuencia [15].
- M169×24251: 169 secuencias de mamíferos, con 24251 caracteres por secuencia [16].
- M1459×8610: 1459 secuencias del virus VIH1, con 8610 caracteres por secuencia [17].

El sistema hardware empleado en los experimentos incluye un procesador Intel Xeon Gold 5218 de 16 cores con 64 GB de RAM DDR4. Las frecuencias consideradas en esta CPU varían desde 2,3 GHz hasta 1,0 GHz. El software instalado en el sistema incluye Ubuntu 18.04LTS, GCC 8.4.0 y OpenMP 4.5. Las opciones de compilación utilizadas incluyen las banderas `-O2`, `-fopenmp`, `-mavx512f` y `-mavx512bw`.

Para obtener resultados estadísticamente fiables, se realizaron 11 ejecuciones independientes por experimento. Cada experimento consistió en la ejecución del algoritmo paralelo de Fitch sobre 2000 árboles filogenéticos con distintas características topológicas, utilizando 16 hilos de acuerdo al número de cores físicos en el sistema. Así, consideraremos los resultados medianos observados en dichos experimentos. Para evaluar el efecto del escalado en frecuencia, se examinaron 10 configuraciones de frecuencias, incluyendo las frecuencias máxima, mínima y otras 8 frecuencias uniformemente distribuidas en los rangos soportados (en saltos de 150 MHz). El tiempo de muestreo para la monitorización energética con RAPL fue de 10 ms.

La evaluación de los resultados obtenidos se ha efectuado considerando cuatro métricas: 1) el tiempo de ejecución (*Tejec*) en segundos, 2) la potencia pico (*Ppico*) en vatios, 3) el consumo energético (*Energía*) en julios y 4) la eficiencia energética (*E-Ef*), expresada como una medida de rendimiento por vatio. Para cuantificar el rendimiento, se ha aplicado la siguiente métrica orientada al problema tratado:

$$\text{Rend(en GigaOp/s)} = \frac{\sum_{i=1}^{NumArboles} FitchOps(T_i)}{Tejec * 10^9}, \quad (3)$$

Tabla I: Resultados del escalado en frecuencia para el diseño paralelo propuesto. Se indican, para cada configuración de frecuencias, los valores medianos observados en los experimentos para cada métrica considerada.

	M50×42456				M156×119750			
Frecuencia	*Tejec* (s)	*Ppico* (W)	*Energía* (J)	*E-Ef* (Rend/W)	*Tejec* (s)	*Ppico* (W)	*Energía* (J)	*E-Ef* (Rend/W)
2,30 GHz	**0,136**	102,563	12,286	0,399	**0,803**	111,544	86,203	0,441
2,15 GHz	0,187	69,715	11,883	0,413	1,008	80,131	74,752	0,508
2,00 GHz	0,192	68,073	11,472	0,427	1,051	79,431	73,782	0,515
1,85 GHz	0,202	63,936	11,360	0,432	1,134	69,623	72,915	0,521
1,70 GHz	0,209	59,454	**11,201**	**0,438**	1,178	68,830	**72,331**	**0,525**
1,55 GHz	0,226	58,873	11,695	0,419	1,298	63,047	75,087	0,506
1,40 GHz	0,235	55,417	11,714	0,419	1,349	62,153	76,389	0,497
1,25 GHz	0,256	52,275	12,102	0,405	1,519	56,159	78,330	0,485
1,10 GHz	0,271	50,602	12,262	0,400	1,611	54,333	80,374	0,473
1,00 GHz	0,288	**50,061**	12,578	0,390	1,756	**53,647**	85,910	0,442
	M169×24251				M1459×8610			
2,30 GHz	**0,281**	102,523	27,157	0,320	**1,454**	104,335	140,089	0,190
2,15 GHz	0,368	73,166	24,298	0,358	1,845	73,371	123,887	0,214
2,00 GHz	0,378	71,937	23,857	0,365	1,888	73,066	122,595	0,217
1,85 GHz	0,398	61,934	23,400	0,372	2,029	64,122	**120,904**	**0,220**
1,70 GHz	0,412	59,940	**22,760**	**0,382**	2,194	62,412	121,933	0,218
1,55 GHz	0,449	56,643	23,848	0,365	2,285	60,613	122,778	0,216
1,40 GHz	0,469	55,342	24,439	0,356	2,484	56,885	128,568	0,206
1,25 GHz	0,517	52,549	24,735	0,352	2,739	51,695	130,688	0,203
1,10 GHz	0,551	51,073	25,736	0,338	2,826	51,435	131,320	0,202
1,00 GHz	0,592	**48,763**	26,553	0,328	3,037	**49,994**	137,818	0,193

donde *NumArboles* es el número de árboles filogenéticos procesados y *FitchOps(T_i)* el número de operaciones de Fitch para el árbol i-ésimo. De acuerdo a la definición original del algoritmo, el valor de *FitchOps* para un árbol T_i viene dado por el número de intersecciones más el número de uniones efectivas:

$$\text{FitchOps}(T_i) = \left(\sum_{v \in V_i} (Grado(v) - 1) * M \right) + P(T_i), \tag{4}$$

donde *Grado(v)* es el grado de un nodo interno v del conjunto de nodos V_i de T_i, M la longitud de secuencia y $P(T_i)$ el valor de parsimonia de T_i. En esta expresión, $\sum_{v \in V_i}(Grado(v) - 1) * M$ representa la cantidad de operaciones de intersección, mientras que $P(T_i)$ denota el número de uniones efectivas.

A. Análisis de resultados

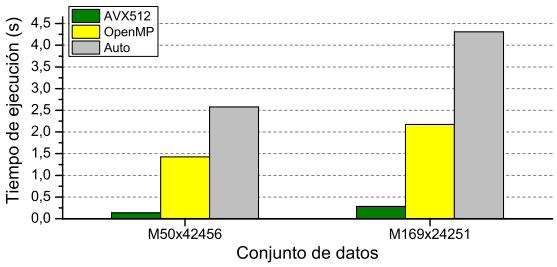
La Tabla I muestra los resultados obtenidos por el diseño propuesto bajo distintas configuraciones de frecuencias. Puede observarse que las configuraciones de 2,30 GHz y 1,00 GHz dieron lugar a los mejores resultados de tiempo de ejecución y potencia pico, respectivamente. Sin embargo, estas configuraciones no representaron las más satisfactorias atendiendo al consumo energético.

Por su parte, los mejores resultados de energía y eficiencia energética fueron obtenidos por la configuración de 1,70 GHz en los datasets M50×42456, M156×119750 y M169×24251 y por la de 1,85 GHz para M1459×8610. En este último conjunto de datos, que contiene el mayor número de secuencias, la configuración de las frecuencias permitió reducir los requisitos energéticos para su resolución de 140,1 a 120,9 julios. Esta técnica también tiene efectos positivos en el conjunto de datos con mayor longitud de secuencia, M156×119750, e incluso en instancias dominadas por bajas latencias, como M50×42456. Con respecto a la configuración referencia de 2,30 GHz, el uso de frecuencias intermedias como la de 1,70 GHz desemboca en mejoras de energía de hasta un 16,2 %, llegando a alcanzar eficiencias energéticas de hasta 0,5 Giga-operaciones por segundo y por vatio.

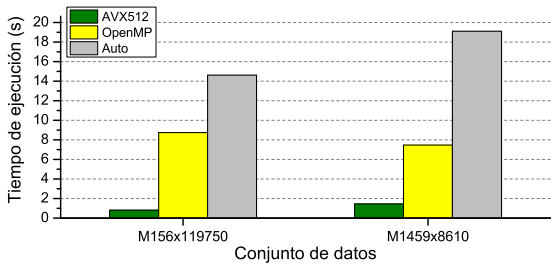
A continuación, examinaremos el efecto de las estrategias paralelas planteadas en comparación con otras aproximaciones. En concreto, compararemos el diseño multihilo basado en AVX512 con 1) un diseño que emplea directivas OpenMP tanto para la gestión multihilo como para la vectorización y 2) una implementación multihilo que gestiona los cálculos SIMD mediante autovectorización basada en compilador.

La Figura 2 ilustra los tiempos de ejecución y potencias pico para cada conjunto de datos, usando las frecuencias que obtuvieron mejores resultados en estas métricas (2,30 GHz para el tiempo y 1,00 GHz para la potencia). El diseño propuesto, con intrínsecas AVX512, obtiene aceleraciones de 10,5× (M50×42456), 10,9× (M156×119750), 7,7× (M169×24251) y 5,1× (M1459×8610) sobre la implementación con directivas SIMD de OpenMP. Estos factores de aceleración se incrementan hasta 18,9× en comparación al código con autovectorización.

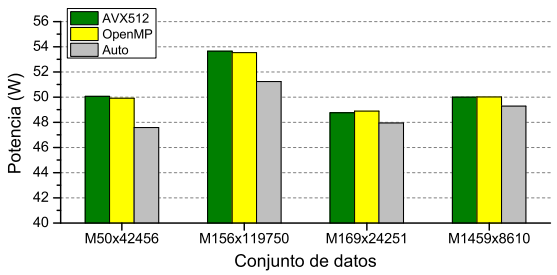
No obstante, la implementación AVX512 presenta un *overhead* de potencia adicional de un 3,3 % en término medio, en comparación a la autovectorización. Este requisito adicional de potencia es más notorio en conjuntos de datos como el M156×119750,



(a) *Tiempo de ejecución (1/2)*

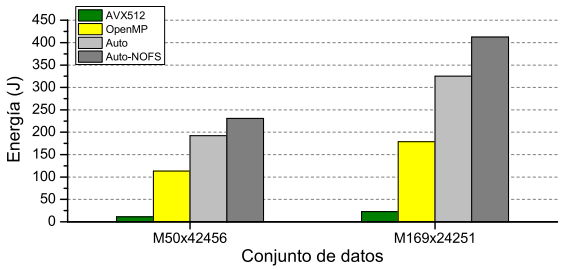


(b) *Tiempo de ejecución (2/2)*

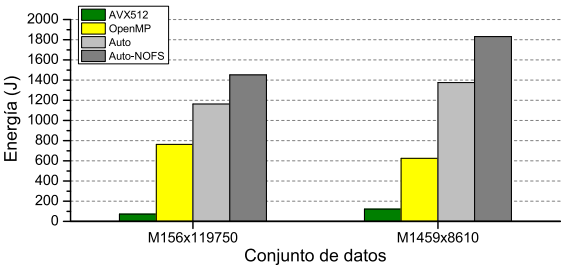


(c) *Potencia pico*

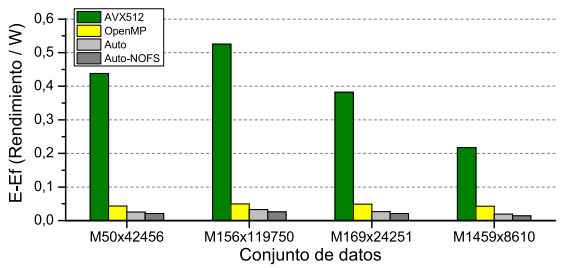
Fig. 2: Comparativas de tiempo de ejecución y potencia pico entre AVX512, directivas OpenMP y autovectorización (Auto).



(a) *Energía consumida (1/2)*



(b) *Energía consumida (2/2)*



(c) *Eficiencia energética*

Fig. 3: Comparativas de consumo y eficiencia energética entre AVX512, directivas OpenMP, autovectorización (Auto) y autovectorización sin escalado en frecuencia (Auto-NOFS).

el cual implica un procesamiento intensivo de datos vectoriales debido a su longitud de secuencia. Así, en este conjunto de datos, el *overhead* de potencia crece hasta un 4,7 %. Por su parte, los resultados de potencia para la versión con directivas SIMD de OpenMP son muy similares a los del diseño AVX512.

En lo que respecta a los resultados de energía, la Figura 3 presenta la comparativa de consumo y eficiencia energética. En estas comparativas se han considerado las configuraciones de frecuencias que han obtenido los mejores resultados promedio para cada diseño (1,70 GHz para la versión con intrínsecas AVX512 y para la versión con directivas OpenMP y 2,00 GHz para el código con autovectorización).

Puede observarse que el diseño AVX512 consigue mejoras energéticas relevantes en todos los conjuntos de datos considerados. En concreto, se observan mejoras en el consumo de hasta 10,5× sobre la versión con directivas OpenMP y 17,2× sobre el código autovectorizado. De hecho, AVX512 representa la opción más satisfactoria atendiendo a la eficiencia energética, a pesar de sus requisitos adicionales de potencia.

Para visualizar el efecto combinado de las estrategias adoptadas, la Figura 3 incluye también resultados del código con autovectorización ejecutándose a una frecuencia de reloj de 2,30 GHz. Al usar esta versión, la ejecución del algoritmo puede alcanzar consumos energéticos de hasta 1830,6 julios. Combinando escalado en frecuencia y estrategias algorítmicas, es posible reducir significativamente estos requisitos y obtener mejoras energéticas de 20,6× (M50×42456), 20,1× (M156×119750), 18,1× (M169×24251) y 15,0× (M1459×8610). Puede observarse que las mejoras más significativas se dan en aquellos escenarios de procesamiento con un uso intensivo de cálculos vectorizados (es decir, en los conjuntos de datos con mayor longitud de secuencia). Estos resultados dan cuenta de la valía de combinar tecnologías hardware y estrategias paralelas precisas para acometer de manera eficiente el procesamiento de este tipo de cargas de trabajo bioinformáticas.

B. Comparativas con otras aproximaciones

Con objeto de comparar el diseño propuesto con aproximaciones de otros autores, se han realizado experimentos adicionales con el método Parsimonator, una implementación alternativa del algoritmo de Fitch para arquitecturas multicore basada en AVX y que está incluida en el software filogenético del estado del arte RAxML [18]. Para realizar la comparativa, se ha seguido la misma metodología experimental que en la evaluación de nuestro diseño, aplicando a su vez un estudio de frecuencias de reloj. Las mejores configuraciones de frecuencia identificadas para

Tabla II: Comparativas de resultados medianos con el método Parsimonator.

Instancia	Propuesta	Parsimonator
Tejec (s)		
M50×42456	**0,136**	0,412
M156×119750	**0,803**	1,440
M169×24251	**0,281**	1,425
M1459×8610	**1,454**	10,760
Ppico (W)		
M50×42456	50,061	**49,190**
M156×119750	53,647	**49,737**
M169×24251	**48,763**	49,559
M1459×8610	**49,994**	49,997
Energía (J)		
M50×42456	**11,201**	36,308
M156×119750	**72,331**	132,537
M169×24251	**22,760**	133,578
M1459×8610	**121,933**	1010,697
E-Ef (Rend/W)		
M50×42456	**0,438**	0,135
M156×119750	**0,525**	0,287
M169×24251	**0,382**	0,065
M1459×8610	**0,218**	0,026

Parsimonator fueron 2,30 GHz (para tiempo de ejecución), 1,00 GHz (para potencia) y 1,40 GHz (para el consumo y la eficiencia energética).

La Tabla II presenta los resultados obtenidos y la comparativa con nuestro diseño propuesto. Nuestra aproximación consigue mejores tiempos de ejecución que Parsimonator, manteniendo valores similares de potencia (sobre 50 vs. 49 vatios de promedio). En términos de consumo y eficiencia energética, el diseño propuesto alcanza un comportamiento más satisfactorio en todos los conjuntos de datos estudiados, observándose mejoras energéticas de hasta 8,3× sobre Parsimonator. Estos resultados por tanto muestran la valía del diseño paralelo propuesto en comparación con otras aproximaciones ampliamente utilizadas en escenarios biológicos reales.

V. Conclusiones

Este trabajo ha investigado la combinación de escalado en frecuencia y aproximaciones algorítmicas paralelas para mejorar la eficiencia energética de los cálculos asociados a un problema bioinformático importante: la inferencia filogenética. Considerando como caso de estudio el algoritmo de Fitch, se ha definido un diseño paralelo que emplea estrategias de procesamiento multihilo con OpenMP y vectorización mediante intrínsecas AVX512. A su vez, se han utilizado herramientas de escalado en frecuencia para apoyar a las estrategias paralelas en la consecución de entornos de ejecución eficientes orientados a la reducción del consumo energético.

Los resultados obtenidos en cuatro conjuntos de datos reales denotan la consecución de mejoras energéticas notables con respecto a otras aproximaciones, como es el caso de una implementación OpenMP pura con directivas SIMD, una aproximación multihilo del estado del arte basada en AVX y una versión del código que adoptaba autovectorización a través del compilador. Como resultado, se redujeron los requisitos energéticos del conjunto de datos más complejo de 1830,6 a 120,9 julios.

Como trabajo futuro, se plantea la exploración de entornos de ejecución eficientes en otras arquitecturas, como es el caso de ARM y RISC-V. También se investigará el potencial de nuevas implementaciones SIMD, como AMX, y su aplicación a otros problemas como la inferencia de redes filogenéticas.

Agradecimientos

Este trabajo, respaldado por la AEI (Agencia Estatal de Investigación, España), el MICIU (Ministerio de Ciencia, Innovación y Universidades, España) y el FEDER (Fondo Europeo de Desarrollo Regional, Unión Europea), es parte del proyecto PID2022-137275NA-I00 (proyecto X-BIO) financiado por MICIU/AEI/10.13039/501100011033 y por FEDER, UE. Este trabajo también ha sido parcialmente financiado por la Junta de Extremadura y el FEDER mediante la ayuda GR24017.

Referencias

[1] L. Lannelongue, J. Grealey, and M. Inouye, "Green Algorithms: Quantifying the Carbon Footprint of Computation," *Advanced Science*, vol. 8, no. 12, article no. 2100707, pp. 1–10, 2021.

[2] S. M. Nabavinejad, S. Reda, and M. Ebrahimi, "Coordinated Batching and DVFS for DNN Inference on GPU Accelerators," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 10, pp. 2496–2508, 2022.

[3] J. Guerreiro, A. Ilic, N. Roma, and P. Tomás, "DVFS-aware application classification to improve GPGPUs energy efficiency," *Parallel Computing*, vol. 83, pp. 93–117, 2019.

[4] R. Muralidhar, R. Borovica-Gajic, and R. Buyya, "Energy Efficient Computing Systems: Architectures, Abstractions and Modeling to Techniques and Standards," *ACM Computing Surveys*, vol. 54, no. 11s, article no. 236, pp. 1–37, 2022.

[5] D. D'Agostino, I. Merelli, M. Aldinucci, and D. Cesini, "Hardware and Software Solutions for Energy-Efficient Computing in Scientific Programming," *Scientific Programming*, vol. 2021, article no. 514284, pp. 1–9, 2021.

[6] L. Szustak, R. Wyrzykowski, T. Olas, and V. Mele, "Correlation of Performance Optimizations and Energy Consumption for Stencil-Based Application on Intel Xeon Scalable Processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 11, pp. 2582–2593, 2020.

[7] T. Warnow, *Computational Phylogenetics: An Introduction to Designing Methods for Phylogeny Estimation*, Cambridge University Press, UK, 2017.

[8] J. Grealey, L. Lannelongue, W. Y. Saw, J. Marten, G. Méric, et al., "The Carbon Footprint of Bioinformatics," *Molecular Biology and Evolution*, vol. 39, no. 3, article no. msac034, pp. 1–15, 2022.

[9] P. Malakonakis, A. Brokalakis, N. Alachiotis, E. Sotiriades, and A. Dollas, "Exploring Modern FPGA Platforms for Faster Phylogeny Reconstruction with RAxML," in *Proc. of IEEE BIBE 2020*, 2020, pp. 97–104.

[10] N. Alachiotis, P. Skrimponis, M. Pissadakis, and D. Pnevmatikatos, "Scalable Phylogeny Reconstruction with Disaggregated Near-Memory Processing," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 3, article no. 25, pp. 1–32, 2022.

[11] X. Gu, *Statistical Analysis of Molecular and Genomic Evolution*, Oxford University Press, UK, 2024.

[12] Y. Turakhia, B. Thornlow, A. Hinrichs, J. McBroome, N. Ayala, et al., "Pandemic-scale phylogenomics reveals the SARS-CoV-2 recombination landscape," *Nature*, vol. 609, pp. 994–997, 2022.

[13] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "RAPL in Action: Experiences in Using RAPL

for Power Measurements," *ACM Trans. Model. Perform. Eval. Comput.*, vol. 3, no. 2, article no. 9, pp. 1–26, 2018.

[14] F. Husník, T. Chrudimský, and V. Hypša, "Multiple origins of endosymbiosis within the Enterobacteriaceae (γ-Proteobacteria): convergence of complex phylogenetic approaches," *BMC Biology*, vol. 9, no. 87, pp. 1–17, 2011.

[15] R. E. Timme, J. B. Pettengill, M. W. Allard, E. Strain, R. Barrangou, et al., "Phylogenetic diversity of the enteric pathogen Salmonella enterica subsp. enterica inferred from genome-wide reference-free SNP characters," *Genome Biol. Evol.*, vol. 5, no. 11, pp. 2109–2123, 2013.

[16] R. W. Meredith, J. E. Janečka, J. Gatesy, O. A. Ryder, C. A. Fisher, et al., "Impacts of the Cretaceous Terrestrial Revolution and KPg Extinction on Mammal Diversification," *Science*, vol. 334, no. 6055, pp. 521–524, 2011.

[17] "HIV Sequence Database," `https://www.hiv.lanl.gov/`, 2005, Last accessed: 11-February-2026.

[18] A. M. Kozlov, D. Darriba, T. Flouri, B. Morel, and A. Stamatakis, "RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference," *Bioinformatics*, vol. 35, no. 21, pp. 4453–4455, 2019.

Hacia la identificación textil robusta mediante detección de inconsistencias del diccionario en separación espectral

Adrián Sarrías[1], Miguel Martínez-Rach[1], Otoniel López-Granado[1] y Héctor Migallón[1]

Resumen— **Los sistemas de identificación textil mediante imagen hiperespectral suelen basarse en separación espectral (*spectral unmixing*) lineal bajo la hipótesis implícita de disponer de un diccionario completo de *endmembers*. En entornos industriales reales, esta hipótesis rara vez se cumple debido a la presencia de fibras no declaradas, la degradación del material o la aparición de mezclas inesperadas, lo que puede dar lugar a resultados aparentemente coherentes desde el punto de vista físico, pero que no reflejan la composición real del material. En este trabajo se analiza el comportamiento de los métodos Fully Constrained Least Squares (FCLS) y Non-negative Constrained Least Squares (NCLS), utilizados para estimar la composición de materiales, tanto en condiciones consistentes como en situaciones de inconsistencia, incluyendo configuraciones en las que el conjunto de materiales considerados se reduce de forma deliberada. A partir de los errores observados en estas estimaciones, especialmente aquellos derivados de la presencia de materiales no representados en el diccionario, proponemos un método de detección de inconsistencia espectral que actúa como una capa de validación sobre las estimaciones de NCLS. El método introduce un mecanismo explícito de rechazo que distingue entre casos Consistentes, de Inconsistencia Parcial y de Inconsistencia Completa, preservando al mismo tiempo la identificación del material dominante siempre que sea factible. Esta propuesta proporciona una estrategia práctica para una identificación textil robusta, al permitir detectar errores asociados a inconsistencias de diccionario en escenarios industriales reales.**

Palabras clave— **Imagen hiperespectral, separación espectral, identificación textil, inconsistencia de diccionario, reciclaje textil industrial.**

I. Introducción y motivación

La imagen hiperespectral (HSI – HyperSpectral Imaging) es una tecnología avanzada de sensado que combina imagen y espectroscopia para adquirir información espacial y espectral sobre un área objetivo. A diferencia de los sistemas convencionales de imagen RGB, que capturan información en tres bandas espectrales amplias (rojo, verde y azul), los sensores hiperespectrales registran valores de reflectancia en cientos de bandas espectrales estrechas y contiguas. Esto da lugar a una estructura de datos tridimensional comúnmente conocida como cubo hiperespectral o hipercubo, donde dos dimensiones corresponden a las coordenadas espaciales y la tercera representa la firma espectral de cada píxel [1].

La imagen hiperespectral, por tanto, contiene información en forma de huellas espectrales detalladas de los materiales. Dado que distintos materiales interactúan con la radiación electromagnética de manera diferente en función de su composición química y su estructura física, sus firmas espectrales proporcionan información que puede ser altamente discriminativa. Esta capacidad permite identificar, clasificar y cuantificar materiales incluso cuando resultan indistinguibles a simple vista en imágenes de color estándar [2].

Desarrollada originalmente para aplicaciones de teledetección (remote sensing) [1], la imagen hiperespectral se ha extendido gradualmente a múltiples ámbitos industriales. En los últimos años, su potencial para caracterizar y clasificar materiales ha despertado un interés creciente en ámbitos como la economía circular y la gestión sostenible de recursos [3], [4].

En entornos industriales de reciclaje, el despliegue práctico de sistemas de imagen hiperespectral debe considerar varios retos operativos. Entre ellos se encuentran la elevada variabilidad y heterogeneidad de los flujos de materiales reales [5], la iluminación no uniforme y las condiciones de adquisición en sistemas basados en cintas transportadoras [6], la alta dimensionalidad inherente de los datos hiperespectrales y las estrictas restricciones de tiempo real impuestas por los procesos de clasificación en línea [7].

La imagen hiperespectral se ha aplicado con éxito a la clasificación e identificación de fibras textiles, permitiendo discriminar entre materiales naturales y sintéticos, así como entre tejidos mezclados [8], ámbito de trabajo de esta propuesta. La clasificación automática de textiles y las implicaciones medioambientales de la reutilización y el reciclaje textil han sido ampliamente analizadas en la literatura. Por ejemplo en [9] se ofrece una revisión exhaustiva de los impactos ambientales asociados a las estrategias de reutilización y reciclaje textil, destacando su potencial para reducir significativamente las emisiones de gases de efecto invernadero, el consumo de agua y la extracción de recursos primarios frente a la producción de fibra virgen. En este análisis se subraya que los sistemas de reciclaje eficaces son esenciales para la transición del sector textil hacia modelos productivos más sostenibles y circulares. Sin embargo, los beneficios ambientales descritos en [9] dependen en gran medida de la calidad y pureza de los flujos de material recuperado.

La aplicación de técnicas espectroscópicas al reciclaje textil se ha investigado como medio para permitir la clasificación automática de flujos de residuos

[1]Departamento de Ingeniería de Computadores, Universidad Miguel Hernández de Elche, Av. de la Universidad s/n, Elche 03202, España. Autor de correspondencia: {asarrias,mmrach,otoniel,hmigallon}@umh.es.

postconsumo. Diversos estudios han demostrado la viabilidad de la espectroscopia en el infrarrojo cercano (NIR) para identificar y separar materiales textiles como algodón y poliéster en contextos industriales de reciclaje [8], [10]. Estos trabajos han puesto de manifiesto el potencial de las tecnologías de detección para mejorar la eficiencia de clasificación, reducir la mano de obra manual y aumentar la pureza de los residuos textiles recuperados, contribuyendo así a cadenas de reciclaje más sostenibles.

No obstante, aunque el estudio presentado en [10] demostró la viabilidad técnica, también puso de relieve limitaciones estructurales que siguen siendo muy relevantes en los entornos industriales actuales. Los flujos de residuos textiles son inherentemente heterogéneos y suelen contener tejidos mezclados, aditivos químicos, colorantes, contaminantes y distintos niveles de degradación del material, todos ellos factores que alteran significativamente las respuestas espectrales [8]. Además, las prendas rara vez están compuestas por una única fibra pura; por el contrario, las mezclas de fibras, como por ejemplo la combinación de algodón y poliéster, son habituales en la producción textil moderna [10]. Como consecuencia, la firma espectral medida de una prenda textil suele representar una combinación de múltiples materiales constituyentes y no una única clase homogénea.

Los sistemas industriales de clasificación suelen diseñarse bajo la hipótesis de un conjunto predefinido y cerrado de categorías de materiales, con modelos entrenados para clasificar las muestras dentro de un conjunto de tipos textiles conocidos [10], [11]. Sin embargo, los residuos textiles del mundo real evolucionan continuamente debido a la aparición de nuevas composiciones y mezclas de fibras, lo que cuestiona los enfoques basados en conjuntos cerrados de materiales puros (*endmembers*) y motiva el desarrollo de métodos de análisis espectral capaces de detectar la presencia de *endmembers* no representados en dicho conjunto, es decir, fuera del diccionario.

Estas limitaciones ponen de manifiesto la necesidad de métodos de análisis espectral capaces de operar en condiciones industriales más realistas, donde los materiales mezclados y las composiciones textiles cambiantes son la norma y no la excepción. En particular, los enfoques convencionales de separación suelen asumir un conjunto fijo y completamente conocido de *endmembers*, una hipótesis que no siempre se cumple en entornos dinámicos de reciclaje. Este trabajo aborda esta limitación analizando estrategias de separación espectral que tienen en cuenta explícitamente la posible presencia de *endmembers* fuera del diccionario, lo que denominamos inconsistencia parcial o completa del diccionario.

El resto del artículo se organiza de la siguiente manera. En la Sección II se introducen los fundamentos de la separación espectral y se describen los métodos FCLS y NCLS para la estimación de abundancias. En la Sección III se analiza el comportamiento de ambos métodos bajo condiciones de consistencia e inconsistencia de diccionario, utilizando distintas configuraciones de *endmembers*. En la Sección IV se presenta el método propuesto de detección de inconsistencia espectral, basado en una capa de validación sobre las estimaciones de NCLS, y se muestran los resultados de su validación experimental. Finalmente, la Sección V recoge las principales conclusiones del trabajo y las posibles líneas futuras.

II. Separación espectral

La separación espectral es un problema fundamental del análisis de imágenes hiperespectrales cuyo objetivo es descomponer el espectro observado de cada píxel en un conjunto de firmas de materiales puros constituyentes, o *endmembers*, y sus proporciones asociadas [12]. A diferencia de los enfoques convencionales de clasificación, que asignan una única etiqueta discreta a cada píxel (clasificación pura), la separación espectral permite representar la contribución simultánea de múltiples *endmembers* en un mismo píxel.

La separación espectral suele formularse mediante el *Modelo Lineal de Mezcla* (LMM, por sus siglas en inglés, *Linear Mixing Model*), una de las aproximaciones más utilizadas en imágenes hiperespectrales por su sencillez y su interpretación física. La idea básica del modelo es que la respuesta espectral registrada en un píxel no procede necesariamente de un único material puro, sino que puede ser el resultado de la contribución conjunta de varios materiales presentes dentro de la misma celda espacial. En este contexto, cada material puro se representa mediante su firma espectral característica, denominada *endmember*, mientras que la contribución relativa de cada uno de ellos al espectro observado se describe mediante un conjunto de coeficientes de abundancia.

Bajo esta hipótesis, el espectro medido en un píxel se modela como una combinación lineal de las firmas espectrales de los materiales presentes. Esta formulación resulta especialmente adecuada cuando la mezcla puede interpretarse como una superposición macroscópica de componentes, de modo que la señal total observada puede aproximarse como la suma ponderada de las contribuciones individuales de cada material.

Formalmente, bajo el Modelo Lineal de Mezcla, el vector espectral observado $\mathbf{y} \in \mathbb{R}^{\mathbb{L}}$ se expresa como

$$\mathbf{y} = \mathbf{M}\mathbf{a} + \mathbf{n}, \tag{1}$$

donde $\mathbb{L}$ representa el número de bandas espectrales de la imagen, $\mathbf{y}$ es el espectro observado en el píxel analizado, $\mathbf{M} \in \mathbb{R}^{\mathbb{L}\times \mathrm{p}}$ es la matriz que contiene las p firmas espectrales de referencia o *endmembers*, $\mathbf{a} \in \mathbb{R}^{\mathrm{p}}$ es el vector de abundancias asociado a dichos materiales y $\mathbf{n} \in \mathbb{R}^{\mathbb{L}}$ recoge tanto el ruido de medida como posibles errores de modelado.

Por tanto, desde el punto de vista físico, las abundancias indican cuánto contribuye cada *endmember* al espectro observado. Por ello, suele exigirse que dichas abundancias sean no negativas, es decir, $a_i \geq 0$, ya que una contribución negativa carece de significado físico. Además, en muchos métodos se impone

también la restricción de suma a uno, $\sum_{i=1}^{p} a_i = 1$, lo que permite interpretar las abundancias como proporciones relativas de los materiales presentes en la mezcla [12].

En el contexto del reciclaje textil, el LMM proporciona un método natural y físicamente interpretable para describir tejidos mezclados, como las mezclas de algodón y poliéster, donde la respuesta espectral medida corresponde a la contribución combinada de múltiples fibras constituyentes. A diferencia de la clasificación que asigna a cada píxel una única etiqueta de *endmember*, la separación espectral modela explícitamente las proporciones de la mezcla, lo que la hace especialmente adecuada para analizar flujos heterogéneos de residuos textiles.

Sin embargo, como se ha mencionado anteriormente, los enfoques clásicos de separación suelen asumir que el conjunto de *endmembers* $\mathbf{M}$ se conoce a priori o puede estimarse de forma fiable a partir de los datos obtenidos. Esta hipótesis define implícitamente un diccionario de *endmembers* cerrado y completo, en el que todos los endmembers presentes en los datos están representados dentro del conjunto predefinido. En la práctica, no obstante, esta condición puede no cumplirse, y pueden aparecer en los datos observados *endmembers* no incluidos en el diccionario predefinido, referidos como *endmembers* fuera del diccionario o inconsistencia del diccionario.

Con el fin de analizar las implicaciones prácticas de esta limitación, se lleva a cabo un estudio comparativo entre dos estrategias ampliamente utilizadas para la estimación de abundancias, Fully Constrained Least Squares (FCLS) [13] y Non-negative Constrained Least Squares (NCLS) [14]. Ambos métodos operan bajo el modelo lineal de mezcla, pero difieren en la formulación de sus restricciones, especialmente en lo relativo a la condición de suma igual a uno. Evaluar su comportamiento tanto en escenarios consistentes como inconsistentes con el diccionario permite valorar su robustez e identificar posibles modos de fallo en presencia de *endmembers* fuera del diccionario.

A. Fully Constrained Least Squares (FCLS)

Fully Constrained Least Squares (FCLS) es uno de los métodos de estimación de abundancias más utilizados dentro del método LMM, ya que incorpora simultáneamente las dos restricciones físicas sobre las abundancias comentadas anteriormente: no negatividad y condición de suma igual a uno. Al imponer ambas restricciones, FCLS garantiza que las abundancias estimadas permanecen dentro de valores físicamente plausibles, evitando tanto valores negativos como desviaciones respecto a una contribución total unitaria. En consecuencia, las soluciones pueden interpretarse directamente como proporciones físicas de endmembers dentro del píxel, lo que convierte a FCLS en un método de referencia en separación espectral.

Por tanto, el problema de estimación de abundancias consiste en determinar, para cada píxel, la contribución relativa de cada uno de los *endmembers* al espectro observado. Bajo el método LMM, esta tarea se formula como un problema de ajuste en el que se busca encontrar el vector de abundancias $\mathbf{a}$ que mejor reproduzca el espectro medido a partir de una combinación lineal de las firmas espectrales disponibles.

De forma natural, este problema se puede plantear como un problema de mínimos cuadrados, en el que se minimiza el error de reconstrucción entre el espectro observado $\mathbf{y}$ y su aproximación $\mathbf{Ma}$. Sin embargo, a diferencia de un ajuste clásico de mínimos cuadrados, en este contexto es necesario imponer restricciones adicionales que garanticen la interpretabilidad física de la solución.

En particular, las abundancias deben ser no negativas, $a_i \geq 0$, ya que, como se ha dicho, no tiene sentido físico que un material contribuya negativamente a la señal observada. Además, se impone la restricción de suma a uno, $\sum_{i=1}^{p} a_i = 1$, lo que permite interpretar las abundancias como proporciones relativas de los materiales presentes en la mezcla, de toda la mezcla, de modo que su suma representa la totalidad de la misma.

Bajo estas consideraciones, el problema de estimación de abundancias se formula como el siguiente problema de optimización:

$$\min_{\mathbf{a}} \ \|\mathbf{y} - \mathbf{Ma}\|_{\mathbf{2}}^{\mathbf{2}} \quad \text{s.a.} \quad \mathbf{a_i} \geq \mathbf{0}, \ \sum_{\mathbf{i=1}}^{\mathbf{p}} \mathbf{a_i} = \mathbf{1} \qquad (2)$$

Esta formulación se conoce como *Fully Constrained Least Squares* (FCLS), ya que incorpora simultáneamente las dos restricciones físicas fundamentales del problema: no negatividad y suma a uno. La primera evita soluciones no físicas, mientras que la segunda normaliza las abundancias, facilitando su interpretación como proporciones relativas.

Desde el punto de vista computacional, el problema corresponde a un caso particular de programación cuadrática con restricciones lineales. Esto implica que la función objetivo es convexa y las restricciones definen un conjunto factible también convexo, lo que garantiza la existencia de una solución global óptima. Gracias a esta estructura, el problema puede resolverse de manera eficiente mediante algoritmos estándar de optimización numérica.

La formulación y resolución eficiente de este problema fue abordada en [13], cuya implementación ha sido ampliamente adoptada en la literatura. Como resultado, FCLS se ha consolidado como uno de los métodos de referencia en separación espectral, especialmente en aquellos escenarios donde las hipótesis del modelo lineal y las restricciones físicas son razonables.

B. Non-negative Constrained Least Squares (NCLS)

El algoritmo NCLS (*Non-negative Constrained Least Squares*) puede entenderse como una relajación del problema de estimación de abundancias definido en el método LMM. En este caso, se mantiene la

restricción de no negatividad sobre las abundancias, pero se elimina la condición de suma a uno.

Esta modificación responde a la necesidad de abordar escenarios en los que la hipótesis de que el conjunto de *endmembers* disponibles explica completamente el espectro observado puede no cumplirse. En particular, cuando el diccionario de materiales es incompleto o existen componentes no modelados, imponer que las abundancias sumen exactamente uno puede forzar una redistribución artificial de la contribución espectral entre los *endmembers* disponibles, dando lugar a estimaciones erróneas.

Al eliminar la restricción de suma a uno, NCLS permite que el modelo ajuste únicamente la parte del espectro que puede ser explicada por los *endmembers* considerados, sin imponer que toda la señal deba estar representada. De este modo, las abundancias dejan de interpretarse estrictamente como proporciones normalizadas y pasan a reflejar contribuciones relativas no necesariamente completas.

Bajo esta formulación, el problema de estimación de abundancias se expresa como un problema de mínimos cuadrados con restricción de no negatividad:

$$\min_{\mathbf{a}} \ \|\mathbf{y} - \mathbf{M}\mathbf{a}\|_2^2 \quad \text{s.a.} \quad \mathbf{a_i} \geq \mathbf{0} \tag{3}$$

Como consecuencia, las abundancias estimadas no se corresponden necesariamente con proporciones físicas estrictas, pero proporcionan una mayor sensibilidad a firmas de *endmembers* débiles o poco frecuentes. Esta característica hace que NCLS sea especialmente adecuado para tareas orientadas a la detección más que a la estimación cuantitativa precisa. Esta estructura de restricciones relajada puede resultar especialmente ventajosa en escenas hiperespectrales complejas que contienen *endmembers* espectralmente similares.

III. FCLS y NCLS bajo consistencia de diccionario y condiciones fuera del diccionario

La estimación de abundancias se sitúa en el núcleo de la separación espectral, ya que determina la contribución relativa de cada *endmember* en cada píxel, de este modo se pueden obtener mapas de abundancia que describen su distribución espacial. Los algoritmos de separación espectral abordan este problema de estimación bajo diferentes hipótesis físicas y distintas estrategias de regularización, como se ha mencionada en la sección anterior.

A. Conjunto de datos y selección de endmembers

Los experimentos presentados se han llevado a cabo utilizando cuatro muestras textiles industriales cuya composición está garantizada por el fabricante. Los identificadores de estas muestras son 58PES42PAM, 65COT35PES, 82PAM18ELA y 90VIS10PES, con composiciones nominales declaradas de: (i) 58 % de poliéster y 42 % de poliamida, (ii) 65 % de algodón y 35 % de poliéster, (iii) 82 % de poliamida y 18 % de elastano (ELA), y (iv) 90 % de viscosa y 10 % de poliéster, respectivamente. Estas combinaciones permiten evaluar el comportamiento de las estimaciones de abundancia en distintos tipos de mezclas binarias, incluidas configuraciones que involucran componentes elásticos como el elastano, que no está representado explícitamente en el diccionario espectral considerado en este estudio. Además, se consideran escenarios en los que el diccionario se modifica eliminando determinados componentes para analizar otras situaciones de inconsistencia.

Para la adquisición de los datos hiperespectrales se ha utilizado una cámara push-broom de infrarrojo cercano (NIR), modelo FX17 del fabricante Specim, que opera en el rango espectral 900–1700 nm, con una resolución espacial de 640 píxeles y 224 bandas espectrales. Para adaptar la adquisición a las condiciones de laboratorio y a la geometría del sistema fijada por la cinta transportadora, se realizaron recortes tanto espaciales como espectrales. En particular, el análisis se restringió al rango espectral 1365–1660 nm, correspondiente a la región con mayor contenido discriminativo para las fibras consideradas.

El diccionario espectral empleado en este estudio está compuesto por los *endmembers* correspondientes a poliéster (PES), poliamida (PAM), lana (WOO), algodón (COT) y viscosa (VIS), obtenidos a partir de muestras puras certificadas mediante el cálculo del espectro medio en regiones espacialmente homogéneas. Este diccionario se ha evaluado tanto en su forma completa como en configuraciones intencionadamente reducidas, en las que se elimina algún *endmember* con el fin de simular condiciones controladas de inconsistencia.

B. Resultados comparativos

La Tabla I resume los resultados obtenidos mediante FCLS y NCLS en las distintas configuraciones de diccionario consideradas. Para cada muestra textil se evalúan tres tipos de escenarios: un diccionario consistente, condiciones de inconsistencia parcial en las que se elimina deliberadamente uno de los *endmembers* reales y situaciones de inconsistencia completa en las que ninguno de los *endmembers* presentes en la muestra está representado en el diccionario.

Dicha tabla recoge las abundancias estimadas para cada fibra junto con tres métricas de reconstrucción: el error cuadrático medio (RMSE), el error absoluto medio (MAE) y el *Spectral Angle Mapper* (SAM, en grados).

El RMSE cuantifica el error de reconstrucción en términos de norma cuadrática y se define como:

$$\mathrm{RMSE} = \sqrt{\frac{1}{\mathrm{L}}\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2} \tag{4}$$

El MAE mide la desviación absoluta media entre el espectro observado y el reconstruido:

$$\mathrm{MAE} = \frac{1}{\mathrm{L}}\|\mathbf{y} - \hat{\mathbf{y}}\|_1 \tag{5}$$

Por su parte, el SAM evalúa la discrepancia angular entre ambos espectros, independientemente de su

escala:

$$\mathrm{SAM} = \arccos\left(\frac{\mathbf{y}\cdot\hat{\mathbf{y}}}{\|\mathbf{y}\|_2\,\|\hat{\mathbf{y}}\|_2}\right) \tag{6}$$

donde $\mathbf{y}$ representa el espectro observado, $\hat{\mathbf{y}} = \mathbf{Ma}$ el espectro reconstruido y L el número de bandas espectrales.

El objetivo de este análisis comparativo es evaluar el comportamiento de ambos métodos bajo condiciones de consistencia e inconsistencia de diccionario.

Los resultados recogidos en la Tabla I revelan diferencias consistentes entre FCLS y NCLS en todas las configuraciones evaluadas. Ni siquiera bajo condiciones de diccionario completo ambos métodos identifican correctamente los materiales dominantes en cada muestra. El método NCLS sí obtiene resultados coherentes y menores valores de RMSE, MAE y SAM, lo que indica un mejor rendimiento en términos de reconstrucción espectral. El método FCLS, al imponer la condición de suma a uno, puede inducir errores de identificación incluso en condiciones de diccionario consistente, al forzar una redistribución de la energía entre los *endmembers* disponibles. Estos resultados indican que los enfoques clásicos de separación carecen intrínsecamente de un mecanismo que señale de forma explícita la inconsistencia de diccionario. En consecuencia, la estimación de abundancias por sí sola resulta insuficiente en entornos industriales donde pueden estar presentes materiales desconocidos o existen firmas espectrales muy similares.

IV. Método de detección propuesto

Esta sección presenta el método de detección propuesto, diseñado para evaluar la consistencia espectral de las abundancias estimadas.

A. Propuesta de detección de consistencia espectral

Los resultados presentados en la Sección III indican que los algoritmos clásicos de separación no disponen de un mecanismo explícito para rechazar inconsistencias del diccionario. Entre los métodos evaluados, NCLS demuestra mayor robustez y refleja con más claridad la inconsistencia del modelo cuando aparecen materiales no representados. En entornos industriales reales, donde tales condiciones son probables, la estimación de abundancias debe complementarse con un mecanismo de validación que evite asignaciones erróneas. Por esta razón, el método propuesto se construye sobre NCLS como estimador base, incorporando una capa adicional de validación de consistencia espectral.

Tal como se describe en el Algoritmo 1, el método propuesto evalúa de forma secuencial distintas condiciones de consistencia, basadas en el soporte global de las abundancias, la concordancia entre el material dominante y la forma espectral, y el contraste entre los principales candidatos. Tras el análisis de estos criterios, cada muestra se clasifica como Consistente, Inconsistencia Parcial o Inconsistencia Completa. El procedimiento no modifica las abundancias estimadas, sino que introduce una capa adicional de validación para detectar posibles inconsistencias del diccionario.

Algoritmo 1 Método de detección de consistencia espectral

Entrada: Y, A, $\mathcal{F}_{\mathrm{nat}}$
Salida: Estado, F_1, (E_{abs}, SAM)
Normalizar Y, A
$x \leftarrow \mathrm{NCLS(A, Y)}$
$W \leftarrow \mathrm{mean(x)}$, $M_{total} \leftarrow \sum W$
$(F_1, F_2), (P_1, P_2) \leftarrow$ dos fibras principales y sus pesos
$E_{abs} \leftarrow \|Y - Ax\|_2 / \|Y\|_2$
$SAM \leftarrow \arccos\left(\frac{Y \cdot Ax}{\|Y\|\|Ax\|}\right)$
if $M_{total} < 0{,}20$ **then**
  return INCONSISTENCIA COMPLETA
end if
if $P_1/M_{total} > 0{,}85$ **and** $SAM > 0{,}50$ **then**
  return INCONSISTENCIA COMPLETA
end if
$Thr \leftarrow \begin{cases} 1{,}3 & \text{si } M_{total} > 1{,}0 \\ 2{,}0 & \text{en otro caso} \end{cases}$
if $E_{abs} > 0{,}50$ **and** $(P_1/P_2) < Thr$ **then**
  return INCONSISTENCIA COMPLETA
end if
$(thr_E, thr_S) \leftarrow \begin{cases} (0{,}44, 0{,}40) & \text{si } F_1 \in \mathcal{F}_{\mathrm{nat}} \\ (0{,}25, 0{,}20) & \text{en otro caso} \end{cases}$
if $E_{abs} > thr_E \vee SAM > thr_S$ **then**
  return INCONSISTENCIA PARCIAL, Dominante=F_1
else
  return CONSISTENTE, Dominante=F_1
end if

El proceso toma como entrada el espectro de la muestra (Y), un diccionario de referencia (A) y el conjunto predefinido de fibras naturales ($\mathcal{F}_{\mathrm{nat}}$). En primer lugar, se aplica NCLS a la imagen hiperespectral, produciendo un vector de abundancias x por píxel. A continuación, se obtiene un vector global de abundancias W promediando las abundancias sobre todos los píxeles analizados. A partir de esta solución se calcula la suma total de abundancias (M_{total}), que proporciona una medida inicial de cuánto de la señal espectral es explicado por el modelo. Un valor significativamente bajo indica que el espectro de la muestra no está adecuadamente representado en el diccionario de referencia.

Una vez estimadas las abundancias globales, el algoritmo valida la reconstrucción utilizando dos métricas complementarias: el residuo absoluto relativo (E_{abs}), calculado como el residuo L2 normalizado por la energía del espectro original, y el Spectral Angle Mapper (SAM). Mientras que E_{abs} cuantifica la magnitud de la señal no explicada, SAM evalúa la fidelidad geométrica midiendo la similitud de la forma espectral. Esta validación dual permite distinguir entre variaciones de intensidad y desajustes espectrales reales, ya que las formas espectrales incorrectas producen valores más altos de SAM. En consecuencia,

Tabla I: Rendimiento comparativo de FCLS y NCLS bajo configuraciones de diccionario completas y reducidas.

Muestra	Escenario de diccionario	Método	PES (%)	PAM (%)	WOO (%)	COT (%)	VIS (%)	RMSE	MAE	SAM (°)
58PES42PAM	Diccionario consistente	FCLS	1.17	0.46	0.39	0.03	97.95	0.2866	0.2794	3.4967
		NCLS	57.75	39.92	0.03	2.14	0.17	0.0084	0.0073	0.5159
	Sin PES	FCLS	–	1.57	0.36	0.01	98.05	0.2867	0.2795	3.5120
		NCLS	–	99.90	0.00	0.03	0.07	0.0326	0.0217	1.8340
	Sin PAM	FCLS	1.33	–	0.69	0.03	97.95	0.2866	0.2794	3.4978
		NCLS	69.57	–	27.98	2.29	0.17	0.0140	0.0124	0.8294
	Inconsistencia completa de diccionario	FCLS	–	–	1.86	0.01	98.13	0.2868	0.2796	3.5207
		NCLS	–	–	99.94	0.01	0.06	0.0497	0.0410	2.8269
65COT35PES	Diccionario consistente	FCLS	1.52	0.07	0.87	4.81	92.72	0.1279	0.1218	2.5398
		NCLS	33.60	0.04	0.31	65.96	0.09	0.0124	0.0094	0.8468
	Sin PES	FCLS	–	1.29	1.05	4.73	92.92	0.1281	0.1219	2.5553
		NCLS	–	2.23	50.23	47.46	0.08	0.0172	0.0112	1.1720
	Sin COT	FCLS	0.30	0.01	2.50	–	97.19	0.1290	0.1229	2.6496
		NCLS	0.12	0.00	65.87	–	34.01	0.0383	0.0326	2.5854
	Inconsistencia completa de diccionario	FCLS	–	1.57	0.36	–	98.07	0.2867	0.2795	3.5104
		NCLS	–	99.90	0.03	–	0.07	0.0320	0.0217	1.8342
82PAM18ELA	Sin elastano	FCLS	8.91	0.57	0.02	0.78	89.72	0.0390	0.0311	2.8323
		NCLS	9.40	86.03	0.00	4.56	0.00	0.0106	0.0094	0.8676
	Sin PAM	FCLS	9.22	–	0.29	0.82	89.68	0.0390	0.0311	2.8360
		NCLS	30.65	–	66.54	2.80	0.00	0.0189	0.0168	1.5279
90VIS10PES	Diccionario consistente	FCLS	7.53	3.26	1.21	1.59	86.81	0.0066	0.0057	0.4301
		NCLS	13.75	10.14	0.45	4.37	71.29	0.0039	0.0032	0.3363
	Sin PES	FCLS	–	10.25	1.26	0.73	87.76	0.0072	0.0062	0.4788
		NCLS	–	24.44	0.75	1.42	73.39	0.0049	0.0039	0.4187
	Sin VIS	FCLS	4.93	0.17	0.14	94.76	–	0.0467	0.0409	3.2478
		NCLS	12.94	0.12	77.47	9.77	–	0.0145	0.0116	1.2078
	Inconsistencia completa de diccionario	FCLS	–	4.60	0.05	95.35	–	0.0474	0.0416	3.3319
		NCLS	–	1.62	93.78	4.60	–	0.0154	0.0123	1.2867

la identificación de fibras se basa en una verdadera concordancia espectral y no únicamente en un ajuste numérico.

El algoritmo aplica después un conjunto jerárquico de condiciones de seguridad diseñadas para garantizar la fiabilidad de la clasificación final. La primera condición evalúa la suma total de abundancias (M_{total}); si este valor cae por debajo de 0.20, el sistema determina que la información espectral explicada es insuficiente para sostener una decisión, dando lugar al estado de Inconsistencia Completa. Posteriormente, se aplica una condición de consistencia forma–masa: si una fibra presenta una dominancia extrema (superior al 85 % de la abundancia total) pero el ajuste espectral es pobre ($SAM > 0{,}50$), el algoritmo detecta una inconsistencia lógica. Este paso resulta esencial para detectar falsos positivos, donde el modelo intenta asignar una fibra conocida a una muestra con distinta composición química pero características de intensidad similares, imponiendo así una salida de salvaguarda que evita identificaciones erróneas.

Una característica distintiva del método propuesto es la implementación de un umbral dinámico de contraste (Thr_{conf}), que ajusta la confianza de identificación en función de la energía espectral recuperada. El filtro evalúa la competitividad entre la fibra dominante (P_1) y la segunda contribución más relevante (P_2) para valorar si su separación es estadísticamente significativa. Su severidad se adapta a la calidad de reconstrucción: cuando la abundancia total es elevada ($M_{total} > 1{,}0$), se aplica un umbral de 1.3; sin embargo, cuando la energía explicada es menor ($M_{total} \leq 1{,}0$), condición normalmente asociada a inconsistencia de diccionario, el requisito aumenta hasta un factor de 2.0. Si el error residual E_{abs} sigue siendo alto y el contraste entre P_1 y P_2 no supera este umbral adaptativo, la muestra se etiqueta como Inconsistencia Completa. Este mecanismo evita asignaciones arbitrarias bajo ambigüedad espectral, asegurando que una identificación solo sea validada cuando la fibra dominante destaque claramente frente a contribuciones secundarias o ruido residual.

Una vez satisfechos los filtros de integridad y contraste, el algoritmo realiza la etapa final de clasificación utilizando criterios de validación adaptados a la naturaleza química de la fibra dominante. Se definen umbrales específicos de tolerancia para las fibras naturales ($\mathcal{F}_{\text{nat}}$), permitiendo márgenes de error ligeramente superiores ($E_{abs} \leq 0{,}44$, $SAM \leq 0{,}40$) debido

a su variabilidad estructural intrínseca. En cambio, para las fibras sintéticas se imponen límites más estrictos ($E_{abs} \leq 0{,}25$, $SAM \leq 0{,}20$), reflejando la mayor homogeneidad de sus firmas espectrales industriales. Si la reconstrucción se encuentra dentro de estos márgenes, la muestra se etiqueta como Consistente, confirmando la identidad de la fibra dominante F_1. Si las métricas superan los umbrales correspondientes pese a haber superado los filtros anteriores, el sistema asigna el estado de Inconsistencia Parcial, indicando discrepancias espectrales potencialmente causadas por anomalías, mezclas complejas o variaciones químicas no capturadas completamente por el modelo de referencia. Cabe señalar que todos los umbrales fueron determinados empíricamente a partir de experimentos de validación.

B. Validación experimental del método propuesto

El método propuesto se ha evaluado utilizando el mismo conjunto de datos y las mismas configuraciones de diccionario descritas en la Sección III. Los resultados correspondientes se muestran en la Tabla II. En particular, la validación se llevó a cabo tanto bajo condiciones de diccionario completo como en escenarios con omisión deliberada de determinados *endmembers*, con el fin de evaluar la capacidad del método para detectar *endmembers* fuera del diccionario sin comprometer la identificación fiable del componente dominante.

Los resultados mostrados en la Tabla II demuestran que el método propuesto se comporta de manera consistente en los escenarios evaluados. Bajo condiciones de diccionario completo, el sistema identifica correctamente el estado Consistente y determina coherentemente el *endmember* dominante en todos los casos analizados. Cuando uno de los *endmembers* reales se elimina deliberadamente, el método detecta con éxito la inconsistencia parcial del diccionario y clasifica la muestra como Inconsistencia Parcial, manteniendo al mismo tiempo la identificación correcta del componente dominante respaldado por el diccionario disponible. Finalmente, en escenarios donde ninguno de los *endmembers* relevantes está representado, el sistema produce el estado de Inconsistencia Completa, evitando así asignaciones forzadas o físicamente inconsistentes. Estos resultados confirman que el mecanismo de decisión propuesto detecta eficazmente *endmembers* fuera del diccionario sin comprometer la identificación fiable del *endmember* dominante siempre que dicha identificación siga siendo factible.

Cabe señalar, respecto a los resultados mostrados en la Tabla II, que, dado que el método propuesto se basa en NCLS como estimador de abundancias, los valores de abundancia obtenidos coinciden con los ya presentados en la Tabla I para dicho método. Por este motivo, en la Tabla II se omiten estas métricas y se presentan únicamente los resultados de validación del estado de consistencia y la identificación del material dominante, ya que el marco de detección actúa como una capa de validación sobre dichas estimaciones sin modificar las abundancias resultantes.

V. Conclusiones

En este trabajo se ha analizado el comportamiento de FCLS y NCLS bajo distintos escenarios de consistencia del diccionario espectral en aplicaciones industriales NIR sobre textiles. Sólo el método NCLS funciona adecuadamente tanto con un diccionario consistente como cuando aparecen *endmembers* fuera del diccionario, es decir, en situaciones de diccionario inconsistente. A partir de este análisis, se ha desarrollado un método de decisión basado en la consistencia espectral, que incorpora un mecanismo explícito de rechazo. El sistema va más allá de la mera estimación de abundancias al evaluar si la solución obtenida es coherente con el diccionario disponible. De esta forma, es capaz de distinguir entre casos de diccionario consistente, de diccionario totalmente inconsistente y situaciones de inconsistencia parcial en los que la muestra no puede explicarse adecuadamente mediante los *endmembers* disponibles, preservando la identificación fiable del *endmember* dominante siempre que dicha identificación esté justificada. La eficacia del método de detección propuesto abre de forma natural la puerta a su integración con algoritmos de separación más avanzados orientados a una estimación precisa de proporciones bajo inconsistencias de diccionario. En particular, este trabajo sienta una base para el desarrollo de métodos que incorporen regularización espectral y espacial, permitiendo no solo la detección de *endmembers* fuera del diccionario, sino también la estimación estable de fracciones de componentes en entornos industriales complejos. Tales avances resultan especialmente relevantes para sistemas automatizados de reciclaje textil, donde la identificación fiable de materiales y la evaluación consistente de la composición son críticas para asegurar la fiabilidad del proceso y apoyar operaciones de reciclaje técnica y económicamente viables.

Agradecimientos

Este trabajo está parcialmente financiado por los proyectos PID2021-123627OB-C55 y PID2024-158682OB-C33, financiados por MCIN/AEI/10.13039/501100011033 y por "ERDF A way of making Europe".

Referencias

[1] Alexander Goetz, Gregg Vane, Jerry Solomon, and Barrett Rock, "Imaging spectrometry for earth remote sensing," *Science (New York, N.Y.)*, vol. 228, pp. 1147–53, 07 1985.

[2] Jaleed Khan, Hamid Khan, Adeel Yousaf, Khurram Khurshid, and Asad Abbas, "Modern trends in hyperspectral image analysis: A review," *IEEE Access*, vol. 6, pp. 14118–14129, 03 2018.

[3] Jacintha Menezes, Nadeesha Hemachandra, and Kate Isidro, "Role of big data analytics and hyperspectral imaging in waste management for circular economy," *Discover Sustainability*, vol. 5, no. 1, pp. 298, 2024.

[4] Mariangeles Salas, Simran Singh, Raman Rao, Raghul Thiyagarajan, Ashutosh Mittal, John Yarbrough, Anand Singh, Lucian Lucia, and Lokendra Pal, "Hyperspectral imaging for real-time waste materials characterization and recovery using endmember extraction and abundance detection," *Matter*, vol. 8, no. 10, pp. 102365, 2025.

Tabla II: Resultados de validación experimental del método de detección de inconsistencia de diccionario.

Muestra	Escenario	Resultado	Material dominante
58PES42PAM	Completo	Consistente	PES
	Sin PES	Inconsistencia parcial	PAM
	Sin PAM	Inconsistencia parcial	PES
	Inconsistencia completa	Inconsistencia completa	NINGUNO
65COT35PES	Completo	Consistente	COT
	Sin PES	Inconsistencia parcial	COT
	Sin COT	Inconsistencia parcial	PES
	Inconsistencia completa	Inconsistencia completa	NINGUNO
82PAM18ELA	Sin ELA	Inconsistencia parcial	PAM
	Sin PAM	Inconsistencia completo	NINGUNO
90VIS10PES	Completo	Consistente	VIS
	Sin PES	Inconsistencia parcial	VIS
	Sin VIS	Inconsistencia parcial	PES
	Inconsistencia completa	Inconsistencia completa	NINGUNO

[5] Ming-Fang Cheng, Arvind Mukundan, Riya Karmakar, Muhamed Adil Edavana Valappil, Jumana Jouhar, and Hsiang-Chen Wang, "Modern trends and recent applications of hyperspectral imaging: A review," *Technologies*, vol. 13, no. 5, 2025.

[6] Thomas Arnold, Tibor Bereczki, and Dirk Balthasar, "Development of a new line illumination for industrial hyperspectral imaging systems," in *Sensing Technology. ICST 2022*, N. K. Suryadevara, B. George, K. P. Jayasundera, and S. C. Mukhopadhyay, Eds., Cham, 2023, vol. 1035 of *Lecture Notes in Electrical Engineering*, Springer.

[7] E. R. Kristensen, J. Dornonville de la Cour, T. Warburg, R. L. Eriksen, B. Jørgensen, J. E. Avery, and M. Hinge, "High-speed processing of hyperspectral images for enabling demanding industrial applications," *Chemometrics and Intelligent Laboratory Systems*, vol. 267, pp. 105531, 2025.

[8] Fariborz Eghtedari, Leszek Pecyna, Rhys Evans, Alan Pestell, Stuart McLeod, and Shan Dulanty, "Identification of textile fibres using a near infra-red (nir) camera," *Journal of Imaging*, vol. 11, no. 4, 2025.

[9] G. Sandin and G. M. Peters, "Environmental impact of textile reuse and recycling – a review," *Journal of Cleaner Production*, vol. 184, pp. 353–365, 2018.

[10] Jordi-Roger Riba, Rosa Cantero, Pol Riba-Mosoll, and Rita Puig, "Post-consumer textile waste classification through near-infrared spectroscopy, using an advanced deep learning approach," *Polymers*, vol. 14, no. 12, 2022.

[11] Maria Kainz, Johannes K. Krondorfer, Malte Jaschik, Maria Jernej, and Harald Ganster, "Supervised and unsupervised textile classification via near-infrared hyperspectral imaging and deep learning," in *Proceedings of the 7th International Conference on Optical Characterization of Materials (OCM 2025)*, Jürgen Beyerer, Thomas Längle, and Michael Heizmann, Eds., Karlsruhe, Germany, 2025, pp. 309–318, KIT Scientific Publishing.

[12] José M. Bioucas-Dias, Antonio Plaza, Nicolas Dobigeon, Mario Parente, Qian Du, Paul Gader, and Jocelyn Chanussot, "Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 2, pp. 354–379, 2012.

[13] Daniel C. Heinz and Chein-I Chang, "Fully constrained least squares linear spectral mixture analysis method for material quantification in hyperspectral imagery," in *IEEE International Geoscience and Remote Sensing Symposium*, 2001.

[14] Chein-I Chang and Daniel C. Heinz, "Constrained subpixel target detection for remotely sensed imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 38, 2000.

Arquitectura software para el análisis hiperespectral en tiempo real en sistemas industriales de clasificación de materiales

Adrián Sarrías[1], Miguel Martínez-Rach[1], Otoniel López-Granado[1] y Héctor Migallón[1]

Resumen— **La integración de la imagen hiperespectral en sistemas industriales de clasificación ha permitido alcanzar una clasificación de materiales de alta precisión, incluso cuando presentan características visuales similares pero composiciones químicas diferentes. Sin embargo, las exigencias de procesamiento en tiempo real asociadas a la adquisición de datos HSI, caracterizada por su elevada resolución espectral y espacial, requieren estrategias computacionales avanzadas. En este trabajo se presenta una arquitectura software escalable y eficiente, diseñada para el análisis hiperespectral en tiempo real en líneas automatizadas de clasificación de materiales. La arquitectura explota paralelismo heterogéneo y homogéneo para distribuir las tareas de preprocesado, clasificación y segmentación entre múltiples hilos y núcleos de procesamiento. Se desarrollan y evalúan dos métodos de clasificación, basados en *Spectral Angle Mapper* y en redes neuronales artificiales. Ambos alcanzan una alta precisión en la identificación de materiales, aunque afectan de manera diferente a la escalabilidad del sistema. El diseño modular del sistema garantiza su adaptabilidad a distintas configuraciones hardware y tipos de materiales, favoreciendo su escalabilidad y su integración en diversos entornos industriales. La restricción de tiempo real impuesta por la frecuencia máxima de adquisición de la cámara es de $1,493$ ms. Gracias a las optimizaciones aplicadas, los procesos críticos, concretamente el preprocesado y la clasificación, se han reducido a poco más de $30\mu s$ cada uno, consumiendo únicamente alrededor del $5\,\%$ del tiempo disponible y dejando casi el $95\,\%$ libre para operaciones adicionales o mejoras de rendimiento.**

Palabras clave— **Imagen hiperespectral, tiempo real, arquitectura software, clasificación de materiales.**

I. Introducción y motivación

En los últimos años, la creciente preocupación por la sostenibilidad ambiental ha impulsado la transición hacia modelos de economía circular. En este contexto, la acumulación de residuos, especialmente de materiales como plásticos y textiles, plantea un reto significativo que requiere soluciones innovadoras para su recuperación y reutilización eficiente.

La clasificación de residuos constituye una etapa fundamental previa a su tratamiento, ya que permite recuperar materias primas secundarias de alto valor y pureza. Tradicionalmente, se han empleado diversas tecnologías, como separación magnética, sistemas de aire, cribado, rayos X, inducción o visión artificial. Sin embargo, los métodos basados en imágenes digitales presentan limitaciones cuando materiales visualmente similares difieren en su composición química, y los modelos basados en inteligencia artificial requieren reentrenamiento ante nuevas condiciones, además de no poder discriminar adecuadamente utilizando únicamente el espectro visible.

En este contexto, la imagen hiperespectral (HSI – HyperSpectral Imaging), que combina imagen digital y espectroscopía [1], permite capturar la firma espectral de cada píxel en diferentes rangos de longitud de onda, posibilitando la identificación fiable de materiales incluso cuando presentan una apariencia visual similar. Esta capacidad ha favorecido su integración en sistemas industriales de clasificación automatizada, donde robots separan materiales en función de su composición química.

Actualmente, los sensores hiperespectrales industriales ofrecen resolución espacial del orden de milímetros y se utilizan en múltiples ámbitos, como biomedicina, agricultura, control de calidad alimentaria o clasificación de plásticos. Su aplicación en el sector de la gestión de residuos, incluyendo residuos de construcción, electrónicos o urbanos, está experimentando un crecimiento significativo.

El coste computacional del procesamiento de datos hiperespectrales sigue siendo un obstáculo para su despliegue industrial en determinados entornos. Además, la heterogeneidad de los pipelines de procesamiento y de los diferentes entornos industriales requiere el diseño de arquitecturas software modulares, facilitando la disponibilidad de soluciones flexibles y adaptables.

Desde el punto de vista computacional, el principal factor es el volumen de datos hiperespectrales generado por unidad de tiempo. En escenarios industriales exigentes, la combinación de alta resolución espectral y espacial, junto con elevadas tasas de adquisición dificultan el procesamiento en tiempo real mediante implementaciones secuenciales en CPU. En consecuencia, el rendimiento del sistema se degrada si no se emplean estrategias de paralelización explícitas y un diseño eficiente del pipeline.

Asimismo, la evolución de los sensores hiperespectrales ha incrementado notablemente el volumen de datos generado, lo que ha impulsado el uso de interfaces de comunicación de alto ancho de banda como USB3 Vision, 10/25/100 GigE Vision o CoaXPress 2.0. Este crecimiento exige soluciones arquitectónicas capaces de evitar que tanto la comunicación como el procesamiento se conviertan en cuellos de botella.

Para abordar estos retos, en este trabajo se propone una arquitectura software que distribuye las principales etapas del pipeline mediante paralelización heterogénea entre hilos independientes. Poste-

[1]Departamento de Ingeniería de Computadores, Universidad Miguel Hernández de Elche, Av. de la Universidad s/n, Elche 03202, España. Autor de correspondencia: {asarrias,mmrach,otoniel,hmigallon}@umh.es.

riormente, se aplican técnicas de paralelización homogénea mediante estrategias de computación de alto rendimiento en sistemas CPU multinúcleo, centrándose especialmente en aquellas etapas críticas que pueden comprometer el rendimiento en tiempo real o limitar la escalabilidad del sistema.

El resto del artículo se organiza de la siguiente manera: en la Sección II se describe el sistema hardware desarrollado y los aspectos clave de la adquisición hiperespectral; en la Sección III se presenta la arquitectura propuesta; las Secciones IV y V analizan las optimizaciones software y el rendimiento del sistema; la Sección VI discute los resultados obtenidos; y finalmente, en la Sección VII se presentan las conclusiones y líneas futuras.

II. Sistema y configuración de adquisición

El prototipo utilizado integra un sistema de clasificación hiperespectral en tiempo real compuesto por una cinta transportadora, un sistema de adquisición HSI, un robot de selección y una plataforma de computación.

El sistema de adquisición se basa en una cámara Specim FX17, que proporciona 640 píxeles espaciales y 224 bandas espectrales en el rango 900–1700 nm. La cámara opera en modo *line-scan* y se sincroniza con el movimiento de la cinta mediante un encoder, garantizando la adquisición de píxeles cuadrados y evitando distorsiones espaciales. Bajo estas condiciones, cada línea adquirida debe procesarse antes de la llegada de la siguiente, lo que impone una restricción temporal máxima de $1,493$ ms a 670 fps.

El tamaño del píxel sobre la cinta (*Ground Sampling Pixel*, GSP) viene dado por:

$$GSP = \frac{2 \cdot h \cdot \tan(\alpha/2)}{p} \tag{1}$$

En la configuración utilizada ($h = 1004$ mm), se obtiene $GSP = 1{,}08$ mm, lo que determina la frecuencia de adquisición necesaria en función de la velocidad de la cinta. La relación entre velocidad y frecuencia se define como:

$$fps = \frac{v}{GSP} \tag{2}$$

La iluminación se realiza mediante lámparas halógenas que cubren el espectro visible y NIR, configuradas para garantizar una respuesta espectral estable sin saturación del sensor.

El sistema de clasificación se valida utilizando cuatro materiales plásticos (acrilonitrilo butadieno estireno (ABS), polipropileno (PP), polietileno de alta densidad (HDPE) y poliestireno (PS)), manipulados mediante un robot industrial con ventosas.

III. Arquitectura software propuesta

En esta sección se describe el diseño de la arquitectura software, así como las optimizaciones realizadas para cumplir los requisitos de tiempo real en un sistema industrial de clasificación de materiales.

El framework de desarrollo seleccionado es Qt [2], debido a su carácter multiplataforma, que permite ejecutar la aplicación en Windows, Linux y MacOS sin modificaciones significativas. Qt ofrece herramientas robustas para el desarrollo de interfaces gráficas, así como un modelo eficiente de comunicación entre componentes basado en el mecanismo de señales y *slots*, que facilita la modularidad y el desacoplamiento entre módulos.

La cámara Specim FX17 se conecta al sistema mediante Camera Link, interfaz que requiere una tarjeta capturadora dedicada, habitualmente soportada en sistemas Windows.

La arquitectura de la aplicación, mostrada en la Figura 1, se basa en la ejecución concurrente de cuatro hilos principales: *main*, *camera*, *segment* y *sender*. El hilo *main* gestiona la interfaz gráfica y la inicialización del sistema.

El hilo *camera* es responsable de la adquisición, preprocesado y clasificación de las líneas hiperespectrales. Cada línea capturada se recibe a través de la interfaz Camera Link y se almacena en un buffer gestionado por la tarjeta capturadora.

Con el fin de asegurar la continuidad temporal, se implementa un mecanismo de verificación basado en identificadores de línea proporcionados por la cámara. Cualquier discrepancia se interpreta como pérdida de datos. En caso de pérdida de líneas, se replica la clasificación válida más reciente para mantener la continuidad espacial. Además, se monitoriza el número de líneas perdidas y se generan alertas si se superan umbrales predefinidos.

Las líneas clasificadas se almacenan en un buffer lineal de tamaño doble respecto a la ventana de visualización (1600 líneas), utilizando una estrategia de duplicación con desplazamiento. Este enfoque garantiza que la ventana activa de datos sea siempre contigua en memoria, evitando la complejidad asociada a buffers circulares y mejorando la localidad de memoria y el rendimiento de caché.

Cuando se completa la clasificación de la ventana activa, se activa el hilo *segment*, que realiza la detección de objetos de forma periódica permitiendo la detección progresiva de objetos sin interrumpir la adquisición y clasificación.

El proceso de segmentación genera los centroides de los objetos detectados. Para evitar duplicidades, se aplica un procedimiento de validación que compara cada nuevo centroide con los previamente detectados, considerando coincidencias exactas o desplazamientos dentro de un umbral. Los centroides válidos se almacenan en una lista global de objetos disponibles para manipulación robótica. Los objetos asignados o fuera de alcance se eliminan de la lista.

El hilo *sender* gestiona la interacción con los robots. Cuando un robot solicita una nueva tarea, se selecciona el objeto más adecuado y se envía la orden de recogida correspondiente. Paralelamente, los resultados de segmentación se envían al hilo *main* para su visualización.

La ejecución secuencial de estas etapas no permi-

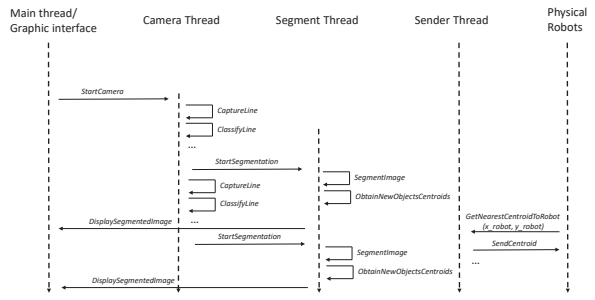


Fig. 1: Arquitectura de hilos de la aplicación.

tiría cumplir las restricciones temporales impuestas por la frecuencia de adquisición de la cámara. Por ello, la arquitectura se basa en una primera capa de paralelización heterogénea, que distribuye las tareas entre hilos independientes, evitando bloqueos y desacoplando operaciones críticas.

Cuando la carga computacional de una etapa lo requiere, se aplica una segunda capa de paralelización homogénea, distribuyendo el procesamiento interno entre múltiples cores. Esta estrategia jerárquica permite escalar el sistema a medida que aumentan la frecuencia de adquisición o la dimensionalidad espectral.

Adicionalmente, se emplea un hilo basado en ROS para la comunicación con el PLC, encargado del control de la cinta, la iluminación, el robot y otros subsistemas industriales. Este hilo tiene un coste computacional reducido, ya que se limita a la gestión de eventos.

El modelo de concurrencia sigue una estrategia de un único escritor y múltiples lectores, donde cada recurso compartido tiene un propietario bien definido. Las modificaciones estructurales se restringen a hilos específicos, mientras que el resto realiza accesos controlados de solo lectura. Este diseño evita condiciones de carrera y garantiza un comportamiento determinista en tiempo real. La comunicación entre hilos se basa en el mecanismo asíncrono de señales y *slots* de Qt, asegurando que las operaciones no críticas no interfieran con las etapas críticas.

*A. Hilo de adquisición (*camera *thread)*

Este hilo es responsable de la adquisición de líneas de la cámara, el preprocesado y la clasificación. Todas estas operaciones deben completarse antes de la llegada de la siguiente línea, debido a las restricciones de tiempo real del sistema. Dado que este hilo constituye el principal cuello de botella potencial, su eficiencia es crítica para garantizar la escalabilidad del sistema, especialmente al aumentar la frecuencia de adquisición, la resolución o la complejidad de clasificación.

El mayor coste computacional del sistema se concentra en el preprocesado espectral y la clasificación. Por ello, el hilo *camera* es el principal objetivo de paralelización homogénea, utilizando regiones paralelas de OpenMP para distribuir los cálculos espectrales entre múltiples núcleos, manteniendo un comportamiento determinista.

A.1 Preprocesado

El preprocesado espectral es fundamental para mejorar la calidad de los datos antes de la clasificación. En este trabajo se emplea una cadena de preprocesado compacta y eficiente, diseñada para ejecución en tiempo real, basada en dos etapas de normalización. En primer lugar, se aplica una corrección mediante referencias blanca y negra (calibración de reflectancia) para compensar variaciones del sensor y de la iluminación. A continuación, se realiza una normalización min–max por banda, tras la cual los valores se escalan y convierten a formato entero, permitiendo un procesamiento eficiente en las etapas posteriores.

Las referencias blanca y negra se obtienen a partir de múltiples capturas de una superficie uniforme y con el obturador cerrado, respectivamente, calculándose de forma individual por píxel para tener en cuenta efectos de iluminación no uniforme y distorsiones ópticas. Además, en el método basado en SAM se incorpora un filtro heurístico que descarta píxeles pertenecientes a la cinta transportadora, reduciendo la carga computacional. Este filtrado no se aplica en el método basado en ANN, donde la cinta se considera una clase adicional.

A.2 Clasificación

Una vez preprocesados los datos, se realiza la clasificación. En este trabajo se evalúan dos métodos: uno basado en Spectral Angle Mapper (SAM) y otro en redes neuronales artificiales (ANN). El objetivo es clasificar cuatro materiales: ABS, PP, HDPE y PS, cuyas respuestas espectrales son similares.

Las firmas espectrales de referencia se obtienen a partir de múltiples regiones de interés (ROIs) en diferentes muestras y posiciones. Tras aplicar el mismo preprocesado, los espectros se promedian para generar un patrón representativo por material, lo que permite incorporar nuevos patrones sin aumentar la complejidad del sistema.

Clasificación basada en SAM: El método SAM calcula el ángulo entre el espectro de cada píxel y los patrones de referencia en un espacio vectorial de dimensión n. En este trabajo se realiza un recorte espectral, utilizando únicamente las bandas 50 a 198, eliminando regiones poco discriminativas. El ángulo espectral θ (menor valor indica mayor similitud) se define como:

$$\theta = \arccos\left(\frac{\mathbf{Spectrum}\cdot\mathbf{Pattern}}{\|\mathbf{Spectrum}\|\,\|\mathbf{Pattern}\|}\right) \quad (3)$$

Clasificación basada en ANN: El segundo método de clasificación analizado se basa en redes neuronales artificiales. La arquitectura del modelo ANN empleado se muestra en la Figura 2. El modelo está compuesto por una capa de entrada correspondiente a las 224 bandas espectrales de cada píxel, tres capas densas con función de activación *ReLU*, una capa de *dropout* para reducir el sobreajuste [3], y una capa de salida con activación *softmax*.

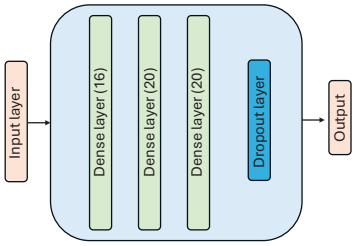


Fig. 2: Arquitectura del modelo ANN.

Al igual que en el método basado en SAM, se aplica un recorte espectral. El modelo se entrena con un conjunto de aproximadamente 240000 espectros de píxeles correspondientes a los distintos materiales y a la cinta transportadora. El conjunto de datos se divide en un 80 % para entrenamiento y un 20 % para validación.

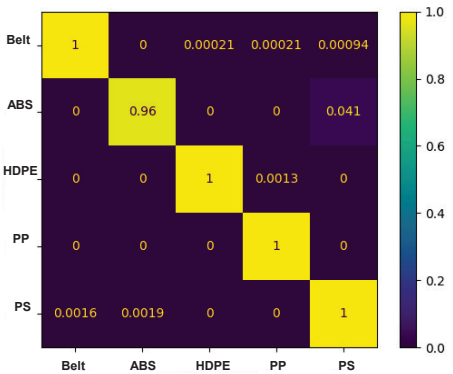


Fig. 3: Matriz de confusión del modelo ANN.

El modelo entrenado alcanza una precisión de 0,9903. Como se muestra en la Figura 3, la clasificación es prácticamente perfecta para todos los materiales, excepto para ABS, que presenta una precisión del 96 %.

B. Segment thread

Una vez completada la adquisición, el preprocesado y la clasificación de una ventana de líneas (1600 en nuestro caso), se activa el hilo *segment*. En esta etapa, para cada material se genera una máscara binaria y, mediante funciones de OpenCV, se obtienen los contornos, centroides y áreas de los objetos detectados.

Los objetos se almacenan en una lista accedida con exclusión mutua por el hilo *segment*, que añade nuevos objetos, y por el hilo *sender*, que extrae objetos para su manipulación.

En entornos industriales estructurados, esta etapa puede resolverse mediante técnicas ligeras. No obstante, en escenarios más complejos, pueden emplearse métodos basados en aprendizaje profundo para detección y segmentación de objetos [4], así como arquitecturas que combinan información espectral y espacial [5], [6].

C. Sender thread

El hilo *sender* es responsable de la comunicación entre la aplicación y multiples robots. Al iniciarse, el hilo *sender* establece comunicación mediante sockets con los robots activos.

El sistema gestiona tres tipos de mensajes: `BUSY`, `ACK` y `NEW`. El mensaje `BUSY` indica que el robot está ocupado, mientras que `ACK` se utiliza para mantener la conexión activa.

IV. Métodos de optimización a nivel software

El sistema utilizado se basa en un procesador AMD Ryzen 9 7950X (arquitectura Zen 4), con 16 núcleos físicos y 32 hilos. Aunque el procesador soporta *Simultaneous Multithreading* (SMT), en este trabajo se emplea un máximo de 16 hilos, uno por núcleo físico. Esta decisión evita la compartición de recursos entre hilos lógicos, reduce la contención en caché y memoria, y proporciona un comportamiento más predecible y de menor latencia, fundamental en sistemas en tiempo real.

En sistemas Windows, el planificador del sistema operativo puede migrar dinámicamente los hilos entre núcleos, lo que introduce latencias adicionales debido a la pérdida de localidad de caché. Para evitar este efecto, se emplea asignación de hilos (*thread placement*), fijando cada hilo a un núcleo específico [7].

El *thread placement* se implementa utilizando *SetThreadAffinityMask* sobre los manejadores nativos de los hilos. Además, para evitar interferencias con procesos del sistema, los hilos de la aplicación se asignan a partir del núcleo lógico 8, dejando los primeros núcleos para el sistema operativo. Asimismo, se utilizan únicamente núcleos lógicos pares, garantizando que cada hilo se ejecute en un núcleo físico independiente.

En particular, los hilos *main* y *ROS* se asignan al núcleo 8, el hilo *segment* al núcleo 10, el hilo *sender* al núcleo 12, y el hilo crítico *camera* al núcleo 14. Los núcleos restantes se emplean para paralelizar las tareas más costosas, principalmente el preprocesado y la clasificación.

A. Preprocesado espectral

El Algoritmo 1 describe el preprocesado aplicado a cada píxel de una línea hiperespectral. Para acelerar su ejecución, se emplea OpenMP, paralelizando el procesamiento a nivel de píxel mediante un bucle **parallel for**.

Algoritmo 1 Preprocesado paralelo

Entrada: Línea hiperespectral, referencias blanco-negro, umbrales
Salida: Línea normalizada, máscara cinta/material
parallel for $i = 0$ to `spatialSize` $- 1$ **do**
    `Spectrum` $\leftarrow$ espectro bruto del píxel i
    `darkSpectrum` $\leftarrow$ referencia oscura
    `whiteMinusDark` $\leftarrow$ `whiteSpectrum`$[i]$ $-$ `darkSpectrum`$[i]$
    (`Spectrum`$[i]$, `maxSpectrum`) $\leftarrow$ DoubleNormalisation (véase Sección III-A.1)
    `isConveyorBelt`$[i]$ $\leftarrow$ ConveyorBeltHeuristic
end parallel for

El rendimiento de la paralelización depende de parámetros como el tipo de planificador (*static* o *dynamic*) y el tamaño de bloque (*chunk size*), que influyen en el equilibrio de carga y la localidad de memoria.

B. Clasificación

Uno de los métodos de clasificación estaba basado en ANN, el cual depende de bibliotecas externas (TensorFlow), que abstraen los detalles de bajo nivel pero limitan el control sobre la gestión de hilos y la optimización de memoria. Por el contrario, el método basado en SAM ha sido implementado completamente en el proyecto, lo que permite aplicar estrategias de paralelización específicas.

ANN: El modelo ANN se integra en la aplicación y se ejecuta sobre cada línea hiperespectral, generando una etiqueta de clasificación para cada píxel, incluyendo aquellos correspondientes a la cinta transportadora. La inferencia se realiza íntegramente en CPU, ya que los requisitos de rendimiento se satisfacen sin necesidad de aceleradores como GPU.

Por defecto, TensorFlow utiliza todos los recursos disponibles del sistema, incluso en modo inferencia. Este comportamiento puede ajustarse mediante su API de configuración (*TF_SetConfig*), que permite controlar parámetros como el número de hilos, la visibilidad de dispositivos o el paralelismo interno. Sin embargo, TensorFlow no permite aplicar *thread placement* a nivel de núcleo físico, lo que limita la optimización fina del sistema.

SAM: En el Algoritmo 2 el coste computacional por píxel está dominado por el cálculo del ángulo espectral, que se repite para cada clase de material. No obstante, el algoritmo incorpora una etapa previa que distingue entre píxeles de material y de cinta transportadora. Si un píxel se clasifica como cinta, el coste es mínimo; en caso contrario, se realiza la comparación completa con todos los patrones.

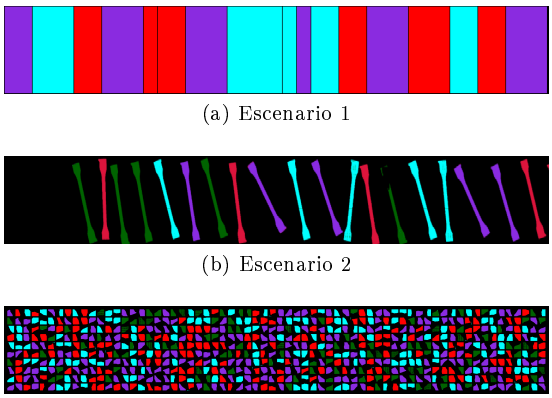
Algoritmo 2 Clasificación paralela por línea mediante SAM

```
Entrada: Línea normalizada (Spectrum)
         Patrones de materiales (Materials)
parallel for i = 0 to spatialSize − 1 do
    if isConveyorBelt[i] then
        Asignar conveyor_label
    else
        min_angle ← ∞
        best_class ← −1
        for all pattern ∈ Materials do
            θ ← SAM(pixel, pattern)
            if θ < min_angle then
                min_angle ← θ
                best_class ← índice de pattern
            end if
        end for
        Asignar best_class
    end if
end parallel for
```

V. Resultados experimentales

A. Configuración experimental

La plataforma empleada se basa en un procesador AMD Ryzen 9 7950X, con 16 núcleos físicos y 32 hilos, y un total de 81 MB de caché (1 MB L1, 16 MB L2 y 64 MB L3). La memoria principal es de 64 GB DDR5 a 6000 MT/s. El sistema se completa con una unidad NVMe de 1 TB. No se usa GPU en ninguna configuración.



(a) Escenario 1

(b) Escenario 2

(c) Escenario 3

Fig. 4: Escenarios experimentales considerados.

El sistema de adquisición se basa en una cámara SPECIM FX17, que opera en el rango SWIR (900–1700 nm) y proporciona 224 bandas espectrales y 640 píxeles espaciales por línea.

A máxima resolución espectral, la cámara alcanza una frecuencia de adquisición de 670 fps, lo que implica un tiempo disponible de 1,493 ms por línea. Cada línea capturada corresponde a una sección espacial, y la sucesión de líneas permite reconstruir el cubo hiperespectral en tiempo real.

Para evaluar el comportamiento del sistema bajo distintas condiciones operativas, se han definido tres escenarios (Figura 4):

- **Escenario 1 (carga completa):** La cinta transportadora está completamente cubierta por un único material. Este caso representa la situación de mayor carga computacional.
- **Escenario 2 (escenario real):** Se transportan fragmentos de los cuatro materiales (ABS, PP, HDPE y PS), con presencia significativa de fondo. Este escenario corresponde a condiciones industriales reales.
- **Escenario 3 (alta fragmentación):** Material triturado distribuido densamente sin solapamiento. Este escenario estresa principalmente la etapa de segmentación y gestión de objetos.

Cada escenario contiene 12000 líneas, con 640 píxeles por línea y 224 bandas, lo que equivale a 7,68 millones de espectros por escenario.

B. Rendimiento de clasificación

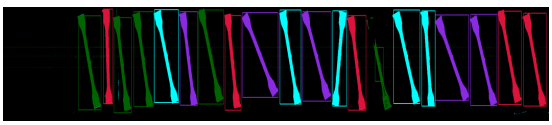
En primer lugar, se evalúa la precisión de clasificación de ambos métodos. Para ello, se calcula el porcentaje de píxeles correctamente clasificados, así como el porcentaje de objetos correctamente identificados tras la segmentación.

Como se muestra en la Tabla I, el método basado en ANN alcanza una precisión del 99,9 %, superior al 98,0 % obtenido por el método SAM. En términos de objetos segmentados, ambos métodos alcanzan una tasa de acierto del 100 %.

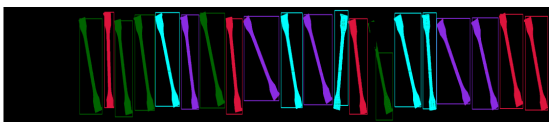
La Figura 5 muestra los resultados de clasificación y segmentación en el escenario 2. El método basado en ANN presenta un rendimiento ligeramente superior. Las únicas discrepancias observadas en ambos

Tabla I: Precisión de clasificación para los métodos SAM y ANN.

	SAM	ANN
Precisión (%)	98,0	99,9
Tasa de acierto (%)	100,0	100,0



(a) Método SAM



(b) Método ANN

Fig. 5: Resultados de segmentación y clasificación en el escenario 2.

métodos corresponden al material PS blanco, cuya naturaleza semitransparente provoca una detección incompleta del contorno, afectando al cálculo del centroide.

C. Rendimiento computacional

El hilo *camera* es el más crítico desde el punto de vista computacional, ya que debe completar el preprocesado y la clasificación de cada línea dentro del intervalo de 1,493 ms impuesto por la cámara a máxima frecuencia (670 fps).

C.1 Evaluación computacional de la etapa de preprocesado

Como se explicó en la Sección IV-A, el preprocesado basado en doble normalización difiere ligeramente según se utilice el método de clasificación basado en SAM o el basado en ANN. A diferencia del primero, el método ANN no requiere una clasificación previa de los píxeles para distinguir la cinta transportadora. No obstante, todos los resultados computacionales presentados en esta sección incluyen la etapa de preprocesado correspondiente en cada caso.

La Tabla II muestra los tiempos de preprocesado, incluyendo la etapa de preclasificación descrita anteriormente, en función del número de hilos OpenMP utilizados, empleando planificación *static* y un tamaño de bloque máximo, de modo que todos los hilos reciben la misma carga de trabajo. En esta configuración, el sistema operativo (Windows 11) se encarga de asignar los hilos a los núcleos y puede migrarlos dinámicamente si lo considera necesario. La tabla, correspondiente al escenario 1, incluye además la desviación estándar, el *speed-up* y la eficiencia paralela. Todos los valores representan el tiempo medio calculado sobre las 12000 líneas de dicho escenario.

Como puede observarse, la paralelización ofrece un buen rendimiento, aunque los tiempos de cómputo se sitúan ya en el orden de microsegundos. El objetivo no es únicamente cumplir los requisitos de tiempo real, sino hacerlo de forma escalable. La desviación estándar calculada sobre las 12000 líneas constituye además un indicador relevante de estabilidad.

Tabla II: Tiempo de preprocesado frente al número de hilos (*static*, sin *thread placement*, escenario 1).

	Hilos				
	1	**2**	**4**	**6**	**8**
Tiempo (μs)	191,9	104,6	57,0	44,2	41,2
Desv.	19,6	23,1	21,2	22,9	24,8
Speed-up	–	1,8	3,4	4,3	4,7
Efic.	–	92 %	84 %	72 %	58 %

A continuación, se analiza el comportamiento al emplear planificación *dynamic* en lugar de *static*. La Tabla III muestra los tiempos de ejecución del preprocesado con planificación dinámica y distintos tamaños de bloque.

Tabla III: Tiempo de preprocesado frente al número de hilos (*dynamic*, sin *thread placement*, escenario 1).

	Hilos				
	1	**2**	**4**	**6**	**8**
Dynamic (chunk = 5)					
Tiempo (μs)	195,06	110,04	100,08	50,89	52,82
Desv.	24,96	27,62	37,11	34,37	35,62
Speed-up	–	1,8	1,9	3,8	3,7
Efic.	–	89 %	49 %	64 %	46 %
Dynamic (chunk = 10)					
Tiempo (μs)	198,20	105,54	74,00	71,80	57,20
Desv.	33,80	19,79	38,71	39,53	42,83
Speed-up	–	1,9	2,7	2,8	3,5
Efic.	–	94 %	67 %	46 %	43 %

Las tablas II y III muestran que la sobrecarga introducida por la planificación *dynamic*, frente a *static*, no se justifica en esta etapa. Aunque en teoría la planificación dinámica puede corregir posibles desequilibrios de carga, en este caso no produce mejoras de rendimiento. Además, también se incrementa la variabilidad temporal, como reflejan los mayores valores de desviación estándar. Este comportamiento se explica por la naturaleza uniforme del preprocesado, donde el coste computacional por píxel es esencialmente idéntico y apenas existen divergencias dependientes de los datos.

Como se discutió en la Sección IV, otra estrategia de optimización consiste en fijar los hilos a núcleos físicos concretos. En esta configuración, cada hilo, ya sea creado explícitamente por el hilo *main* o generado implícitamente dentro de una región paralela OpenMP, se asigna a un núcleo fijo, evitando su migración por parte del sistema operativo.

Debe señalarse que el hilo que inicia una región paralela OpenMP actúa como hilo maestro dentro de dicha región y conserva su asignación original, mientras que el resto de hilos requiere una asignación explícita. La Tabla IV muestra los tiempos de preprocesado cuando se aplica esta fijación de hilos. La comparación con la Tabla II muestra una mejora significativa tanto en tiempo de ejecución como en eficiencia paralela, especialmente cuando aumenta el número de hilos. Esta mejora se debe, principalmente, a la reducción de la contención por recursos entre hilos de aplicación y procesos del sistema.

Los resultados obtenidos en el escenario 1 son con-

Tabla IV: Tiempo de preprocesado (μs) frente al número de hilos (*static*, con *thread placement*, escenario 1).

	Hilos				
	1	**2**	**4**	**6**	**8**
Tiempo (μs)	193,4	103,8	55,8	41,2	31,4
Desv.	19,6	23,1	21,2	22,9	24,8
Speed-up	–	1,9	3,5	4,7	6,2
Efic.	–	93 %	87 %	78 %	77 %

sistentes con los observados en los escenarios 2 y 3, tanto en tendencia como en magnitud. La Tabla V compara los tiempos de preprocesado para los tres escenarios utilizando la configuración óptima identificada en el escenario 1. Las diferencias observadas son mínimas y pueden atribuirse a condiciones transitorias del sistema, como la planificación del procesador o la actividad en segundo plano.

Tabla V: Tiempo de preprocesado (μs) frente al número de hilos (*static*, con *thread placement*).

	Hilos				
	1	**2**	**4**	**6**	**8**
Esc. 1	193,4	103,8	55,8	41,2	31,4
Esc. 2	199,4	109,6	58,9	42,8	32,5
Esc. 3	201,4	104,5	55,1	39,9	31,0

C.2 Evaluación computacional del clasificador basado en ANN

Como se ha dicho no se usa GPU y TensorFlow emplea todos los recursos disponibles del sistema, pero este comportamiento puede modificarse a nivel de sesión mediante la función *TF_SetConfig*, controlando parámetros como *intra_op_parallelism_threads* e *inter_op_parallelism_threads*.

El parámetro *intra_op_parallelism_threads* determina el número máximo de hilos utilizados dentro de una operación individual.

Como muestra la Tabla VI, el proceso de inferencia no presenta escalabilidad: aumentar el número de hilos no reduce el tiempo de ejecución.

Tabla VI: Tiempo de clasificación (μs) del método ANN frente al número de hilos.

Tiempo (μs)	Hilos				
	1	**2**	**4**	**8**	**32**
Esc. 1	227,95	220,18	230,86	305,07	336,10
Esc. 2	227,93	220,17	230,84	305,07	336,09
Esc. 3	227,93	220,16	230,84	305,07	336,09

Los tiempos de inferencia no difieren significativamente entre escenarios, lo que indica que el coste computacional es estable. Aunque el uso de dos hilos reduce ligeramente el tiempo de ejecución, la mejora es limitada. Si bien la configuración actual cumple las restricciones de tiempo real, no es escalable sin el uso de GPU.

C.3 Evaluación computacional del clasificador basado en SAM

A diferencia del caso anterior, el clasificador basado en SAM se ha implementado completamente dentro del proyecto, sin depender de bibliotecas externas. Aunque parte de los datos se almacenan en estructuras OpenCV, esta biblioteca no se utiliza en la ejecución del algoritmo. La implementación ha sido optimizada experimentalmente y complementada con las técnicas descritas en la Sección IV.

Tabla VII: Tiempo de clasificación (μs) del método SAM según hilos y planificación.

Esc.	Planif.	1	2	4	6	8
1	static	179,16	96,10	52,44	35,64	26,55
	dyn. 5	181,95	100,63	53,78	37,68	29,37
	dyn. 10	180,45	99,79	54,48	38,57	30,80
2	static	100,74	58,51	42,75	28,95	22,84
	dyn. 5	102,55	60,93	36,21	27,78	23,54
	dyn. 10	99,86	59,58	33,96	26,48	21,73
3	static	144,05	83,51	49,05	37,25	28,99
	dyn. 5	147,23	88,01	48,84	35,54	28,53
	dyn. 10	144,76	82,85	46,29	33,76	27,15

Tabla VIII: Tiempo de segmentación (μs) por escenario.

Escenario	**Tiempo 400 líneas (μs)**	**Tiempo medio (μs)**
Esc. 1	50588,0	126,47
Esc. 2	47893,0	119,73
Esc. 3	53984,0	134,96

En todos los experimentos siguientes se utiliza *thread placement*, ya que, como se observó en el análisis del preprocesado, esta técnica mejora significativamente la eficiencia, sobre todo al aumentar el número de hilos en las regiones paralelas de OpenMP.

La Tabla VII presenta los tiempos de clasificación del método SAM acelerado para tres configuraciones: planificación *static* con tamaño de bloque máximo, y planificación *dynamic* con tamaños de bloque 5 y 10.

El coste secuencial difiere entre escenarios debido a la preclasificación de píxeles pertenecientes a la cinta transportadora. En el escenario 1, donde todos los píxeles deben analizarse, puede observarse con claridad que el clasificador SAM supera al método ANN en tiempo de ejecución. Además, la implementación basada en SAM permite una paralelización efectiva, alcanzando tiempos aproximadamente diez veces inferiores a los del método ANN. Dado que la planificación *dynamic* con tamaño de bloque 10 ofrece mejores resultados en los escenarios 2 y 3, esta configuración puede considerarse la estrategia por defecto.

C.4 Evaluación computacional de la etapa de segmentación

Esta operación se ejecuta, como mínimo, cada 400 líneas clasificadas, aunque este valor es configurable. La Tabla VIII muestra los tiempos de ejecución de la segmentación para los distintos escenarios. En el peor caso (escenario 3), el tiempo medio por línea es de 134,96 μs. Esto indica que esta etapa no requiere paralelización adicional, ya que dispone de un amplio margen respecto al límite máximo de 1,493 ms.

VI. Discusión

En este trabajo se ha desarrollado y evaluado un sistema computacional para el control de una máquina destinada a la clasificación de materiales reciclables. En particular, se ha implementado una arquitectura software modular con dos objetivos principales: por un lado, cumplir los requisitos de tiempo

real del sistema y, por otro, garantizar que la solución sea escalable y adaptable a nuevas configuraciones con mayores exigencias computacionales, como incrementos en resolución, caudal de procesamiento, número de materiales a clasificar o complejidad espectral.

Dada la naturaleza del proyecto, la plataforma de cómputo integrada en la máquina industrial se dedica exclusivamente a su funcionamiento. En consecuencia, la propuesta parte de la hipótesis de uso completo y exclusivo de dicha plataforma. No obstante, la máquina no es un sistema aislado, sino que dispone de conectividad y capacidades de comunicación. Por ello, todos los resultados presentados se han obtenido con el sistema completamente conectado y ejecutando el sistema operativo. Además, una de las restricciones impuestas por el socio industrial es que el desarrollo debe realizarse bajo Windows.

La plataforma de cómputo utilizada puede considerarse un sistema de gama alta dentro del ámbito de escritorio, con una configuración potente para su categoría, pero sin incorporar aceleradores específicos como GPU.

El uso exclusivo de esta plataforma dentro del sistema industrial ha permitido desarrollar un software altamente eficiente y escalable. Un aspecto clave del diseño es la estrategia de distribución de carga en dos niveles: en un primer nivel, se asignan hilos independientes a los principales componentes software; en un segundo nivel, los cálculos más críticos se aceleran mediante técnicas de paralelización para minimizar el tiempo de ejecución.

Los resultados muestran que ambos niveles de distribución del trabajo ofrecen un rendimiento significativamente mejor cuando la arquitectura software gestiona explícitamente el *thread placement*, en lugar de delegar completamente esta tarea al planificador del sistema operativo. No se utilizan los procesadores lógicos proporcionados por SMT, de modo que solo se emplean núcleos físicos. Como resultado, el número máximo de procesos generados por el software se limita a 12, de los 16 núcleos físicos disponibles, reservando 4 núcleos para el sistema operativo y procesos auxiliares.

En las técnicas de paralelización empleadas en el segundo nivel de distribución de carga, se analizaron dos tipos de planificación: *static* y *dynamic*. En la etapa de preprocesado, donde no existe desequilibrio de carga, la distribución estática resulta preferible. En la etapa de clasificación con el método basado en SAM, donde sí aparece cierto desequilibrio debido a la preclasificación previa, la distribución dinámica mejora ligeramente los tiempos de cómputo.

Los resultados experimentales se obtuvieron sobre tres escenarios representativos de distintas condiciones de carga computacional. Cada escenario consta de 12000 líneas hiperespectrales. Estos tres escenarios representan, por tanto, diferentes regímenes de carga, desde una carga máxima de procesamiento espectral (escenario 1) hasta una configuración industrial real (escenario 2) y una situación de alta fragmentación de objetos (escenario 3). En consecuencia, los resultados obtenidos permiten evaluar el rendimiento del sistema bajo distintas condiciones computacionales sin modificar el hardware ni los parámetros de adquisición.

Se han presentado dos métodos de clasificación, ambos con muy buen rendimiento en la tarea de separación de materiales, aunque la precisión del método basado en ANN es ligeramente superior a la del método SAM. Sin embargo, los tiempos de cómputo acelerados del método SAM son considerablemente menores que los del método basado en ANN. Debe destacarse que el método ANN se ejecutó utilizando la versión CPU de TensorFlow, ya que el sistema no recurre a aceleración por GPU. En el caso del método ANN, el coste computacional aumenta con el crecimiento del tamaño de la red, aunque dicho incremento no es proporcional al número de patrones de referencia, como sí ocurre en el método SAM. Además, en el método SAM se introdujo una etapa previa de filtrado para evitar clasificar los píxeles pertenecientes a la cinta transportadora.

Por otro lado, la incorporación de nuevos patrones en el método SAM no requiere cambios software, mientras que el método basado en ANN exige reentrenar la red neuronal. Este último aspecto resulta especialmente relevante, ya que favorece el uso del método SAM en sistemas industriales versátiles y adaptables.

Desde el punto de vista de la escalabilidad, el coste computacional de la etapa de preprocesado escala aproximadamente de forma lineal con la dimensionalidad espectral y con la resolución espacial del sensor, ya que las operaciones se aplican píxel a píxel a lo largo de todas las bandas espectrales.

La escalabilidad computacional de la etapa de clasificación difiere entre ambos enfoques. En el método SAM, el coste crece aproximadamente de forma lineal con el número de clases de material, dado que el ángulo espectral debe calcularse entre cada píxel y cada firma de referencia. En consecuencia, añadir nuevas clases incrementa directamente el número de comparaciones requeridas. En cambio, en el método ANN, el coste de inferencia está determinado principalmente por la arquitectura de la red y la dimensionalidad de entrada. Incrementar el número de clases suele afectar únicamente al tamaño de la capa de salida, provocando un aumento moderado del coste, pero no un crecimiento proporcional al número de patrones.

Bajo la configuración evaluada (150 bandas espectrales y 640 píxeles espaciales por línea), y considerando el peor escenario computacional, ejecutado con 8 hilos de procesamiento, el tiempo conjunto de preprocesado y clasificación basada en SAM fue de 59 μs por línea. Este tiempo de ejecución corresponde a una frecuencia teórica máxima de aproximadamente 17 kHz, más de 25 veces superior a la frecuencia actual de adquisición de 670 fps (1,493 ms por línea). Asumiendo un escalado lineal respecto a la resolución espacial y la dimensionalidad espectral, este margen sugiere que, manteniendo el mismo hardware y la

misma frecuencia de adquisición, el sistema podría soportar del orden de 16000 píxeles espaciales por línea o cerca de 3800 bandas espectrales dentro del presupuesto temporal disponible.

En cuanto al número de clases de material, el tiempo de preprocesado permanece constante, mientras que la etapa de clasificación SAM escala aproximadamente de forma lineal con el número de firmas de referencia. Bajo estas condiciones, el presupuesto de clasificación disponible de 1,461 ms permitiría teóricamente hasta unas 216 clases de material antes de alcanzar el límite de tiempo real. Además, serían necesarias aproximadamente 33 clases para que el método SAM igualase el tiempo total de ejecución de la implementación basada en ANN. Estas proyecciones indican que la arquitectura propuesta conserva un amplio margen computacional para sensores de mayor resolución, modelos alternativos de cámara, bibliotecas de materiales más extensas y configuraciones industriales más exigentes.

Aunque la arquitectura software es, en gran medida, independiente del sensor hiperespectral concreto, salvo por la interfaz de comunicación y la capa de adquisición, los cambios en el tipo de sensor o en sus características espectrales pueden requerir ajustes en calibración y clasificación. En particular, las referencias radiométricas (calibración blanca y negra) y las firmas espectrales de los materiales deben actualizarse cuando cambian las condiciones de iluminación o el sensor utilizado. En el caso del método SAM, este proceso implica redefinir las firmas de referencia, lo que puede realizarse de forma eficiente. En cambio, para el método ANN, cambios en la respuesta espectral pueden requerir un nuevo entrenamiento de la red neuronal.

Asimismo, determinadas etapas de preprocesado pueden depender de condiciones operativas como variaciones en la iluminación, presencia de polvo, deriva del sensor o factores ambientales que afecten a la estabilidad espectral. Por esta razón, el sistema se ha diseñado para operar con un margen temporal amplio entre adquisiciones, permitiendo incorporar estrategias adicionales de compensación o preprocesado sin comprometer el tiempo real.

Aunque la implementación propuesta se basa únicamente en CPU, decisión motivada principalmente por razones de coste, el análisis de escalabilidad demuestra que existe un margen computacional considerable, por lo que la aceleración hardware no resulta necesaria en las condiciones evaluadas.

No obstante, la aceleración basada en GPU podría resultar relevante en futuras ampliaciones. Esto ocurriría, por ejemplo, si se incorporasen tareas no asociadas a la clasificación, como segmentación basada en aprendizaje profundo, o si aumentasen significativamente la resolución espacial, espectral o temporal, o bien la complejidad de los modelos de clasificación. En tales casos, la estrategia de procesamiento debería evolucionar desde el esquema actual basado en líneas hacia un esquema basado en ventanas, reduciendo la sobrecarga debida a transferencias frecuentes de pequeños bloques de datos y permitiendo el uso eficiente del paralelismo masivo de las GPU. Esta aproximación permitiría aprovechar aceleradores hardware preservando la estructura modular de la arquitectura propuesta.

Aunque la validación experimental se ha realizado sobre un prototipo industrial concreto, la arquitectura ha sido diseñada deliberadamente para garantizar su adaptabilidad a diversos entornos industriales. En relación con la iluminación, la configuración actual emplea lámparas halógenas de amplio espectro, adecuadas para un escenario industrial de coste contenido. En instalaciones con sistemas de iluminación más uniformes o espectralmente optimizados, las necesidades de preprocesado se reducirían, disminuyendo el coste computacional y ampliando aún más el margen temporal disponible.

En cuanto a la variabilidad de sensores, la arquitectura software está desacoplada del hardware de adquisición. Como se ha indicado, cambios en la interfaz de comunicación, el rango espectral o la resolución espacial afectan únicamente a la capa de adquisición y a la configuración de parámetros, sin alterar la estructura central del software. Aunque estos cambios pueden modificar las restricciones de tiempo real, el sistema ha demostrado disponer de un margen temporal amplio, garantizando escalabilidad hacia mayores tasas de adquisición o mayor dimensionalidad de datos.

La heterogeneidad de materiales entre instalaciones industriales se aborda mediante el carácter modular de la propuesta. Como se ha señalado, el número de firmas de referencia puede ampliarse sin modificaciones arquitectónicas en el enfoque basado en SAM, mientras que en el método ANN pueden añadirse nuevas clases mediante reentrenamiento. En caso de mayor variabilidad espectral o composiciones materiales más complejas, el sistema puede operar utilizando el conjunto completo de bandas disponibles o incluso sensores de mayor resolución espectral. Los resultados experimentales muestran que existe margen computacional suficiente para acomodar estos incrementos.

Por otro lado, la velocidad de la cinta transportadora está determinada principalmente por parámetros propios de la instalación industrial, como el tamaño del píxel sobre el terreno o la estabilidad del material durante el transporte. Velocidades elevadas pueden provocar desplazamientos indeseados en materiales ligeros o irregulares, afectando a la precisión de la recogida robótica. Por ello, la velocidad debe ajustarse en función de las propiedades del material y de las restricciones mecánicas. La arquitectura propuesta no impone limitaciones intrínsecas más allá de las derivadas del tiempo de adquisición, y el margen de procesamiento demostrado asegura compatibilidad con una amplia gama de requisitos industriales.

VII. Conclusiones

Se ha validado la viabilidad de un sistema industrial versátil y escalable, demostrando su capacidad

para cumplir los requisitos de procesamiento en tiempo real.

En particular, se ha desarrollado una arquitectura software eficiente y escalable para la clasificación industrial de materiales en tiempo real basada en imagen hiperespectral. La propuesta distribuye las distintas etapas del procesamiento en hilos independientes, constituyendo un primer nivel de paralelización heterogénea, mientras que un segundo nivel de paralelización homogénea se aplica sobre las tareas computacionalmente más críticas mediante técnicas de computación de alto rendimiento en sistemas CPU multinúcleo.

La evaluación detallada de los requisitos computacionales y de la efectividad de distintas estrategias de aceleración, incluyendo OpenMP, *thread placement* y planificación estática y dinámica, ha demostrado que el sistema es capaz de satisfacer las restricciones de tiempo real con un margen amplio, garantizando así su escalabilidad frente a cargas de trabajo más exigentes.

La escalabilidad del sistema depende de múltiples factores, entre ellos el número de bandas espectrales procesadas. En el prototipo desarrollado se emplearon 150 de las 224 bandas disponibles con fines de clasificación. El sistema también es escalable respecto a la resolución espacial del sensor, 640 píxeles por línea en nuestro caso, y a la frecuencia de adquisición, que alcanza hasta 670 fps. Otras dimensiones de escalabilidad incluyen la velocidad y dimensiones de la cinta transportadora, el número de brazos robóticos desplegados, la incorporación de cámaras RGB y la configuración del sistema de iluminación, que puede influir en la complejidad del preprocesado y en la carga computacional. Más allá de estos aspectos físicos, el sistema también ha sido diseñado para soportar escalabilidad algorítmica y computacional. Su arquitectura modular permite integrar algoritmos más sofisticados de preprocesado, clasificación o segmentación sin necesidad de rediseñar el pipeline completo.

En el contexto del problema abordado se han desarrollado y analizado dos métodos de clasificación: uno basado en redes neuronales artificiales (ANN) y otro basado en el algoritmo Spectral Angle Mapper (SAM). Ambos alcanzan una buena precisión de clasificación; sin embargo, el método basado en SAM ha demostrado una mayor eficiencia computacional y una adaptación más sencilla a nuevos materiales, lo que lo convierte en una solución más versátil para entornos industriales dinámicos. Por el contrario, el enfoque basado en ANN resulta preferible cuando el número de materiales a clasificar aumenta significativamente, siempre que el modelo haya sido entrenado adecuadamente.

Como líneas futuras de desarrollo se plantea el uso de aceleración hardware mediante GPU para soportar tareas computacionalmente más exigentes, lo que permitiría incorporar técnicas intensivas de preprocesado, como el filtrado Savitzky-Golay, aumentar el intervalo entre recalibraciones y prolongar el funcionamiento continuo del sistema. Asimismo, se prevé la inclusión de algoritmos heurísticos y funciones de optimización específicamente diseñadas para determinar el orden óptimo de recogida de piezas.

En particular, cuando aparezcan dificultades de segmentación debidas a propiedades intrínsecas del material, como la transparencia parcial del PS blanco, podrían considerarse distintas estrategias alternativas. Una de ellas sería la integración de métodos de segmentación de instancias basados en aprendizaje profundo, capaces de explotar conjuntamente información espectral y espacial para mejorar la reconstrucción de contornos en materiales ópticamente complejos. En estos casos se estudiará el uso de aceleración por GPU bajo un esquema de procesamiento por ventanas, con el fin de gestionar eficientemente la carga computacional adicional manteniendo las restricciones de tiempo real.

De forma alternativa, el diseño modular de la arquitectura propuesta permite incorporar una cámara RGB adicional para mejorar la definición de contornos en aquellos casos en los que el contraste hiperespectral no sea suficiente. Dado que el framework separa las etapas de adquisición, preprocesado, clasificación y segmentación, la integración de esta información complementaria no requeriría modificaciones estructurales del núcleo de la arquitectura.

Agradecimientos

Este trabajo está parcialmente financiado por los proyectos PID2021-123627OB-C55 y PID2024-158682OB-C33, financiados por MCIN/AEI/10.13039/501100011033 y por "ERDF A way of making Europe".

Referencias

[1] Paul L. M. Geladi, Hans F. Grahn, and James E. Burger, *Multivariate Images, Hyperspectral Imaging: Background and Equipment*, chapter 1, pp. 1–15, John Wiley & Sons, Ltd, 2007.

[2] QT Group, "QT framework 6.7," `https://www.qt.io/product/framework`, 2024, Accessed: 2026-02-06.

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014, `http://jmlr.org/papers/v15/srivastava14a.html`.

[4] A. Mukundan, R. Karmakar, D. Gupta, and H.-C. Wang, "Deep learning-based toolkit inspection: Object detection and segmentation in assembly lines," *Computers, Materials and Continua*, vol. 86, no. 1, pp. 1–23, 2025.

[5] H. Wu, C. Li, and Y. Li, "Full-scale semantic segmentation of hyperspectral imaging based on spatial-spectral joint network," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. X-1, pp. 267–274, 2024.

[6] J. Li, H. Wang, A. Zhang, and Y. Liu, "Semantic segmentation of hyperspectral remote sensing images based on pse-unet model," *Sensors*, vol. 22, no. 24, pp. 9678, 2022.

[7] C. P. Ribeiro, M. Castro, V. Marangozova-Martin, J.-F. Méhaut, H. C. Freitas, and C. A. P. S. Martins, "Evaluating CPU and memory affinity for numerical scientific multithreaded benchmarks on multi-cores," *IADIS International Journal on Computer Science and Information Systems*, vol. 7, no. 1, pp. 79–93, 2012, `https://www.iadisportal.org/ijcsis/papers/2012140106.pdf`.

Efficient evaluation of video encoding acceleration techniques: methodology and case study

Xiangyu Wan[1], Clara Baltasar[1], Guillermo Vigueras[1], and Antonio Jesús Díaz-Honrubia[1]

Resumen— **The increasing complexity of modern video coding standards, especially Versatile Video Coding (VVC), demands efficient encoding strategies that often use Machine Learning (ML) to make encoder and decoder implementations practical. Conventional evaluation procedures require substantial computational effort, frequently leading to hundreds of hours of encoding, particularly when multiple models are analyzed. This work introduces a novel methodology for the systematic evaluation and comparison of early-stage ML-based solutions aimed at accelerating video encoders, allowing assessments within feasible timeframes while maintaining consistency across comparisons. A case study explores the use of several ML classifiers to speed up the first-level quadtree partitioning in VVC, achieving a 19.89 % reduction in encoding time with only a 1.16 % increase in BD-rate.**

Palabras clave— **Video coding, evaluation, methodology.**

I. Introduction

DUE to the increasing quality of videos and the growing popularity of Ultra-High Definition (UHD), 2K, and 4K formats, new efficiency requirements for video compression have emerged. To address these needs, the Versatile Video Coding (VVC) standard [1] was released by the Joint Video Experts Team (JVET) in 2020 as the latest video encoding technology. Compared to High Efficiency Video Coding (HEVC) [2], VVC achieves around a 50 % bitrate reduction while maintaining similar subjective quality [3]. However, this improvement comes with a significant increase in computational complexity, making encoding very time-consuming.

Like HEVC, VVC uses Coding Tree Units (CTUs), with Coding Units (CUs) up to 128×128 pixels, and extends quadtree partitioning through the Quadtree Plus Multi-Type Tree (QT+MTT) structure. A CU can be split into square blocks (quad split) or rectangular ones using binary or ternary splits in both directions. This flexible partitioning, together with multiple coding modes, greatly increases encoding complexity.

VVC encoders compute the Rate-Distortion (RD) cost for each CU and mode, selecting the optimal configuration. However, the large number of partitioning options and modes significantly increases computational cost.

Many ML-based methods have been proposed to accelerate encoding, but their refinement requires extensive testing under standard conditions, which is computationally expensive. To overcome this limitation, this paper proposes a methodology that enables efficient validation in reduced time, especially when evaluating multiple models.

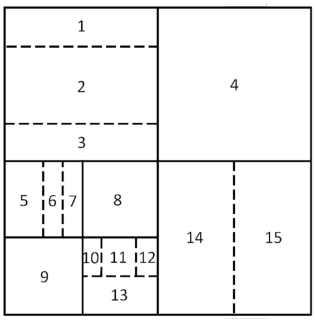


Fig. 1: Example of CTU partitioning in VVC.

The rest of this paper is organized as follows. In Section II, we present a brief technical background and a summary of the state of the art in this field. Section III introduces the proposed methodology, including the definition of the new BTR metric. Then, Section IV presents a use case in which a new fast VVC encoding algorithm is proposed to illustrate the use of the methodology. Finally, the main conclusions are drawn in Section V.

II. Technical background and related work

The partitioning scheme in VVC is a clear evolution over earlier standards. HEVC uses CTUs up to 64×64 pixels with recursive Quad Tree (QT) splitting into CUs as small as 8×8. In contrast, VVC adopts larger 128×128 CTUs and introduces the Multi-Type Tree (MTT) structure [4], improving flexibility through a two-step process: QT splitting into four equal blocks, followed by Binary Tree (BT) and Ternary Tree (TT) splits (with BT only for 128×128 nodes). Figure 1 illustrates this combined partitioning scheme.

Due to recursive splitting and multiple coding modes (inter, intra, affine, skip, merge, etc.) [3], VVC encoding is highly time-consuming. Evaluations under the Common Test Conditions (CTCs) [5] can take months. For example, full RA encoding with four QPs may require up to 117 days on a single CPU. Therefore, a unified methodology is essential to accelerate model development and enable more efficient experimentation. To the best of the authors' know-

[1]Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingenieros Informáticos, Madrid, Spain, e-mail: {xiangyu.w, cm.baltasar, guillermo.vigueras, antoniojesus.diaz}@upm.es.

ledge, no prior work defines a unified methodology for refining fast video coding algorithms. The CTCs represent the only standardization effort, but their high computational cost limits their practical use.

Existing works mainly focus on accelerating VVC encoding. García-Lucas *et al.* [6] proposed a Naïve-Bayes model to predict QT splits of 128×128 blocks, achieving 13 % time reduction with 0.3 % BD-rate increase, later extended in [7]. CNN-based approaches, such as [8], reduce encoding time by 30.63 % with around 3 % BD-rate loss through early CU partition termination.

Random Forest models have also been explored. Amestoy *et al.* [9] introduced risk-controlled predictions, achieving 30.1 % time reduction with 0.61 % BD-rate increase, while Lindino *et al.* [10] reported 34.76 % time savings with 1.03 % loss. Non-ML approaches, such as quadtree parallelization [11], achieve up to 50.98 % time reduction at the cost of higher BD-rate (5.11 %).

III. Proposed methodology for comparing video coding acceleration algorithms

This section presents the proposed methodology for efficiently refining and comparing video coding acceleration algorithms, particularly during validation. It is based on the CTC framework, specifically the widely used Random Access configuration, where the 21 sequences are grouped into five classes:

- Class A1: 4K, 3840×2160 pixels.
- Class A2: 4K, 840×2160 pixels.
- Class B: Full HD, 1920×1080 pixels.
- Class C: WVGA, 832×480 pixels.
- Class D: WQVGA, 416×240 pixels.

From these sequences, a representative subset was selected for validation, ensuring coverage of different resolutions and spatial-temporal characteristics. One sequence per class was chosen based on their Spatial Index (SI) and Temporal Index (TI), as defined in ITU-T P.910 [12], which describe texture and motion complexity. Figure 2 shows their distribution. Sequences with both high SI and TI are rare, as strong motion typically reduces fine details. The selected sequences represent a wide range of content characteristics:

- **Campfire:** Class A1 sequence, low SI, high TI.
- **RollerCoaster2:** Class A2 sequence, low SI, medium TI.
- **ParkScene:** Class B sequence, low SI, low TI.
- **BQMall:** Class C sequence, medium SI, medium TI.
- **BQSquare:** Class D sequence, high SI, low TI.

To compute the BD-rate, four QP values (22, 27, 32, 37) are used. However, full encoding with VTM 20.0 [13] exceeds 525 hours, so in the proposed methodlogy only the first seven frames in coding order are encoded, generating two intra and five inter frames. Despite presentation gaps, only bitrate, PSNR, and encoding time are required, and results

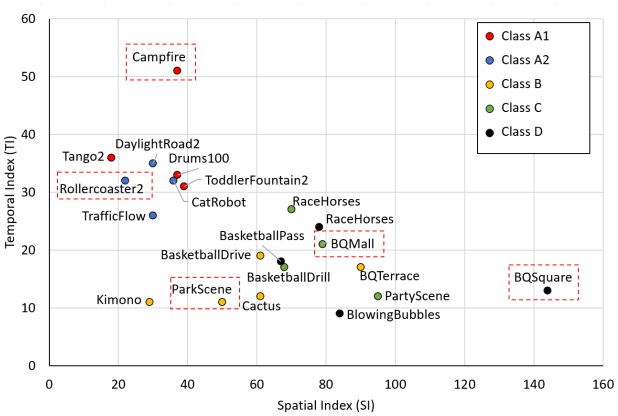


Fig. 2: SI and TI values of the CTC sequences under the *Random Access* configuration. The sequences highlighted in red indicate the selected representative subset.

remain equivalent. This selection ensures all QP offsets for inter frames are represented. Once the subset of sequences and frames is defined, both the reference encoder and the tested encoder are used to encode them at QP values of 22, 27, 32, and 37. From these runs, three metrics are obtained: average bitrate, average PSNR, and total encoding time. The BD-rate is computed from bitrate and PSNR to evaluate compression efficiency, while time savings are determined by comparing the encoding times of both encoders. Many algorithms target either inter- or intra-coding, but not both. Therefore, results from irrelevant frame types can be discarded, focusing the analysis only on the frames affected by the optimized tool.

To validate the chosen subset, the correlations of the proposed subset with the full sequences have been calcuted for the bitrate, the PSNR, and the encoding time, resulting in correlations of 0.9963, 0.9789, and 0.7288, respectively, validating this reduced approach for efficient evaluation.

Finally, Figure 3 illustrates the different steps of the proposed methodology to evaluate various models aimed at accelerating inter-coded frames using the reference VTM 20.0 software for VVC. The steps are as follows:

1. Selection of the proposed representative subset of sequences.
2. Encoding of the first 7 frames from each sequence using the four QP values specified in the CTCs with both, the reference and the modified encoders.
3. Discarding of the two intra-coded frames.
4. Collection of the bitrate, the PSNR, and the encoding time from the encoder's logs.
5. Calculation of the BD-rate.
6. Calculation of the mean of all individual BD-rates and time reductions.

IV. Case study: algorithm to accelerate the first split of the quadtree in VVC

This section presents a case study illustrating the proposed methodology by comparing ML models for accelerating first-level quadtree splitting in VVC inter-coded frames. Models classify 128×128 CUs as

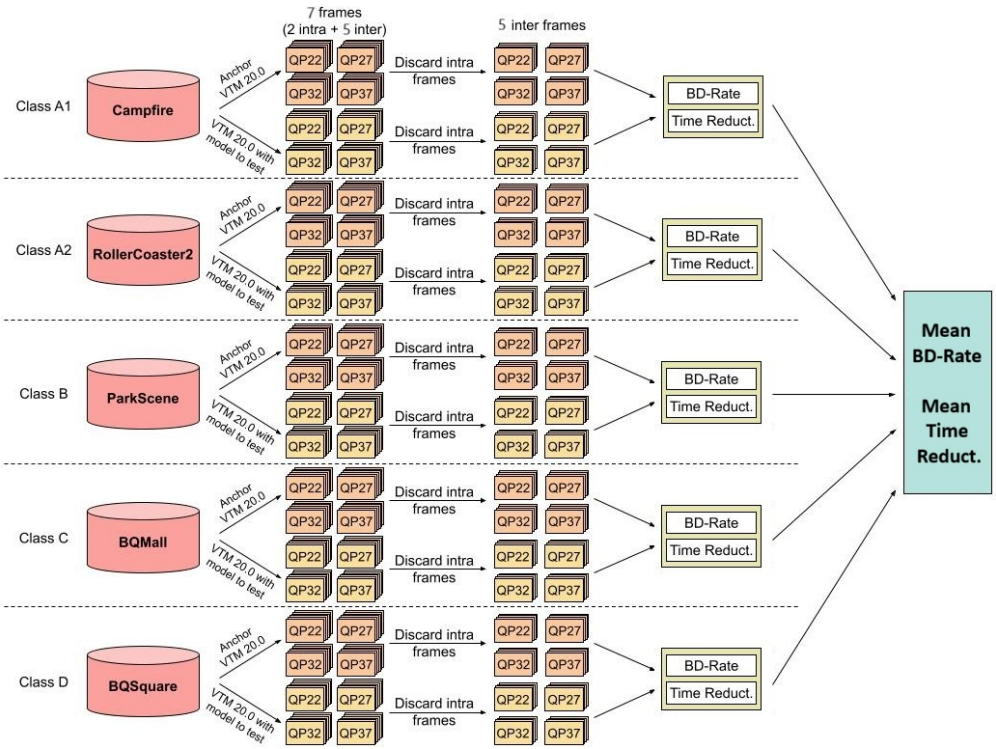


Fig. 3: Proposed methodology for the case of validating inter-coding algorithms using the reference VTM 20.0 encoder for VVC.

split (C_S) or non-split (C_N), either forcing subdivision into 64×64 blocks or evaluating alternative modes.

The dataset generation is first described, followed by the training of 17 models using different ML algorithms. All models are evaluated with the proposed methodology, and the best one is further assessed under CTC conditions, with results compared using the BTR metric.

A. Dataset

A representative dataset is essential for training ML models to predict CU splitting in VVC. The splitting decision is mainly influenced by texture complexity and motion, which can be inferred from luminance information and edge presence. To capture these characteristics, six statistical features are extracted from the CU luminance component:

- **Mean:** average luminance value of the CU.
- **Standard deviation:** measures brightness dispersion, indicating texture complexity.
- **Skewness:** describes the asymmetry of brightness distribution.
- **Kurtosis:** reflects the concentration of values around the mean.
- **Levene's test value:** evaluates variance differences among the four 64×64 sub-CUs.
- **Kruskal-Wallis test value:** assesses differences in mean values across sub-CUs.

To better capture edges and motion, these statistics are also computed after applying the following filters:

- **Sobel filter:** highlights edges using first-order gradients.
- **Laplacian filter:** detects finer details through second-order derivatives.
- **Frame residual:** captures motion by subtracting consecutive frames.

Additionally, the QP is included as a feature, as it affects compression level, image smoothness, and CU size.

The final dataset contains 25 features: six statistics from the original luminance, the same features from three filtered versions, and the QP. Class labels (C_S, C_N) are obtained from VTM 20.0 decisions. Features are denoted as $w_{i,j}$, with w_{QP} for QP and C for the class. All inter-coded CUs from CTC sequences are included as samples.

B. Model learning and hyperparameter tunning

Three ML algorithms were used to train classifiers for predicting the splitting of 128×128 CUs. In total, 17 models were trained using different hyperparameter configurations and evaluated with the proposed methodology.

For the Naïve Bayes (NB) algorithm, only one model was trained, as it has no hyperparameters. To avoid assumptions about feature distributions, all features were discretized using the method in [14]. For Random Forest (RF), combinations of $mss = 2, 5, 10, 15$ and $msl = 5, 10, 25$ were tested. For SVM, both polynomial and RBF kernels were considered.

Feature selection using a forward wrapper [15] was applied to NB and SVM to remove irrelevant features. The selected features for NB were $w_{sobel,std}$, $w_{sobel,kurt}$, $w_{residual,std}$, $w_{residual,kruskal}$, and w_{QP}. For SVM, the selected features were $w_{luma,std}$, $w_{luma,kruskal}$, $w_{sobel,mean}$, $w_{laplace,mean}$, $w_{residual,std}$, $w_{residual,kurt}$, and w_{QP}.

In contrast, RF did not require feature selection, as it inherently performs it during training. Additionally, RF ($mss = 10$, $msl = 25$) and SVM (RBF) were also evaluated on a rebalanced dataset obtained through class weighting proportional to the original class distribution in the dataset.

Overall, 1 NB model, 13 RF models (including one with rebalancing), and 3 SVM models (including one rebalanced) were trained, totaling 17 models.

C. Model selection using the proposed methodology

The proposed methodology was applied using VTM 20.0 as the reference encoder and modified versions incorporating the trained models for CU splitting decisions. Experiments were conducted on an Intel Xeon Silver 4314 CPU at 2.4 GHz.

Table I summarizes the results in terms of BD-rate, and time reduction, averaged over the *Campfire*, *RollerCoaster2*, *ParkScene*, *BQMall*, and *BQSquare* sequences. Each sequence was encoded with seven frames under RA configuration and QPs 22, 27, 32, and 37. Since the models target inter-coded frames, intra-coded frames were excluded. Figure 4 provides detailed heatmaps of BD-rate and time reduction.

According to Table I, the NB model provides the best performance due to its very low BD-rate (1.301 %), despite offering a more moderate time reduction (8.568 %). Other models, such as RF ($mss = 10$, $msl = 25$, rebalanced), SVM with a polynomial kernel, and SVM (RBF, rebalanced), achieve higher time reductions (around 10–15 %), but at the cost of noticeably higher BD-rate values, making them less efficient overall.

Figure 4 shows that most models behave similarly for *RollerCoaster2*, *Campfire*, and *BQMall*, except for the SVM with RBF kernel, which performs poorly in *RollerCoaster2*. Greater variability appears in *BQSquare*, while *ParkScene* only shows good results for the top-performing models. Among them, NB stands out for its more consistent behavior across all sequences.

Considering its minimal BD-rate increase, stable performance, and low complexity (linear, $\mathcal{O}(\mathtt{n})$), the NB model is selected for the proposed fast VVC encoding algorithm.

D. Performance evaluation using the CTCs

The selected Naïve-Bayes model, integrated into the VTM 20.0 encoder, was used for the final evaluation following the full CTC guidelines under the RA configuration. Experiments were performed on the same Intel Xeon Silver 4314 CPU at 2.4 GHz. The results of this fast encoding algorithm are presented in Table II, showing an average time reduction of 19.89 % with a low BD-rate increase of 1.16 %.

These results improve upon those obtained during validation, with lower BD-rate and higher time reduction. This confirms that the proposed validation subset properly represents the full set of sequences. For the subset (*Campfire*, *RollerCoaster2*, *ParkScene*, *BQMall*, *BQSquare*), the BD-rate range is [0,1480, 3,9787], while for all sequences it is [−0,0126, 3,9787], showing strong overlap. Similarly, time reduction ranges are [6,8981, 38,7248] for the subset and [1,8989, 43,6475] overall.

To quantify this similarity, the Jaccard index [16], defined as $J(A, B) = |A \cap B|/|A \cup B|$, is used. The obtained values are 0.96 for BD-rate and 0.76 for time reduction, indicating a high agreement between the reduced subset and the full evaluation.

V. Conclusions

This work presents a methodology to validate video encoding acceleration algorithms without executing the full set of CTC tests. Instead, a reduced subset of 5 sequences and 7 frames per sequence is used, providing a reliable approximation of the overall results. Nevertheless, final evaluation must still follow the complete CTC guidelines.

The methodology is mainly intended for early-stage ML-based approaches, enabling faster validation while maintaining consistency with full evaluations.

A use case is also presented, where an ML-based algorithm is proposed to accelerate VVC encoding. A dataset is constructed from 128×128 CUs, and several models are trained and compared using the proposed methodology. The Naïve-Bayes model achieves the best balance, and its full evaluation shows a 19.89 % reduction in encoding time with a limited BD-rate increase of 1.16 %.

Agradecimientos

This work has been supported by the Spanish Ministry of Science and Innovation, the Agencia Estatal de Investigación (10.13039/501100011033) and the European Commission (FEDER: A way to make Europe) under projects PID2024-161538OB-I00, and PREP2024-003139.

Referencias

[1] ISO/IEC and ITU-T, "Versatile Video Coding (VVC). ITU-T Recommendation H.266 and ISO/IEC 23090-3," Video standard, 2020.

[2] ISO/IEC and ITU-T, "High Efficiency Video Coding (HEVC). ITU-T Recommendation H.265 and ISO/IEC 23008-2," Video standard, 2013.

[3] Benjamin Bross, Ye-Kui Wang, Yan Ye, Shan Liu, Jianle Chen, Gary J Sullivan, and Jens-Rainer Ohm, "Overview of the versatile video coding (VVC) standard and its applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3736–3764, 2021.

[4] Y.-W. Huang, J. An, H. Huang, X. Li, S.-T. Hsiang, K. Zhang, H. Gao, J. Ma, and O. Chubach, "Block partitioning structure in the VVC standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3818–3833, 2021.

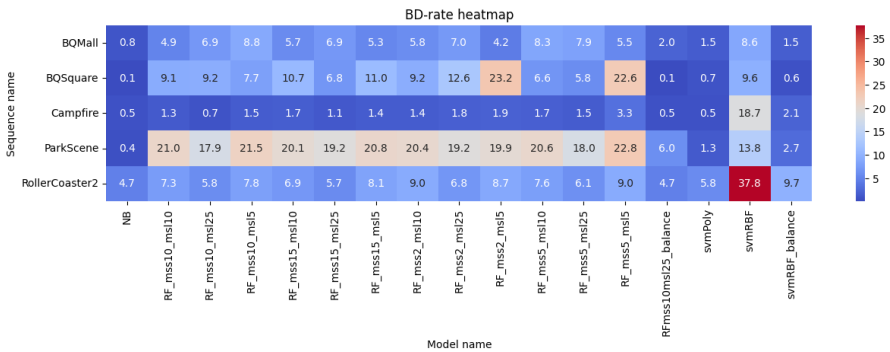
[5] X. Li and K. Suehring, "Common Test Conditions and Software Reference Configurations," in *Technical Report JVET-H1010, Joint Video Experts Team (JVET)*, 2017.

[6] David García-Lucas, Gabriel Cebrián-Márquez, Antonio Jesús Díaz-Honrubia, Thanuja Mallikarachchi, and Pedro Cuenca, "Cost-efficient HEVC-based quadtree splitting (HEQUS) for VVC Video Transcoding," *Signal Processing: Image Communication*, vol. 94, pp. 116199, 2021.

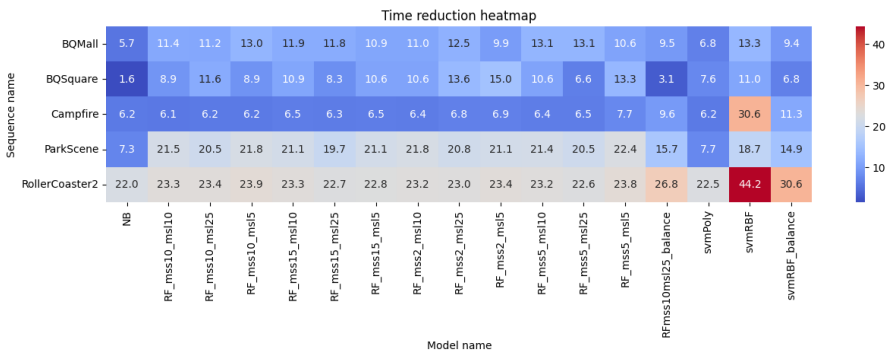
[7] D. García-Lucas, G. Cebrián-Márquez, A.J. Díaz-Honrubia, T. Mallikarachchi, and P. Cuenca, "A fast full

Tabla I: Results of applying the proposed methodology to the 17 models obtained from the derived dataset. They show that the NB-based model offers the best trade-off between BD-rate and time reduction when the BTR metric is used.

Model	BD-rate (%)	Time reduction (%)
NB	1,301	8,568
$RF_{mss=10,msl=25}$	8,099	14,575
$RF_{mss=10,msl=25,rebalanced}$	2,671	12,955
$RF_{mss=10,msl=10}$	8,728	14,242
$RF_{mss=10,msl=5}$	9,459	14,774
$RF_{mss=15,msl=25}$	7,947	13,754
$RF_{mss=15,msl=10}$	9,023	14,738
$RF_{mss=15,msl=5}$	9,320	14,406
$RF_{mss=5,msl=25}$	7,858	13,843
$RF_{mss=5,msl=10}$	8,987	14,937
$RF_{mss=5,msl=5}$	12,632	15,564
$RF_{mss=2,msl=25}$	9,465	15,330
$RF_{mss=2,msl=10}$	9,154	14,593
$RF_{mss=2,msl=5}$	11,593	15,271
$SVM_{polynomial}$	1,954	10,156
SVM_{RBF}	17,686	23,569
$SVM_{RBF,rebabalaced}$	3,345	14,598



(a) BD-rate (%) results of the different ML models across the different sequences.



(b) Time reduction (%) results of the different ML models across the different sequences.

Fig. 4: Comparison of BD-rate and time reduction achieved by the different ML models across the selected sequences in the proposed methodology.

partitioning algorithm for HEVC-to-VVC video transcoding using Bayesian classifiers," *Journal of Visual Communication and Image Representation*, vol. 94, pp. 103829, 2023.

[8] Zhaoqing Pan, Peihan Zhang, Bo Peng, Nam Ling, and Jianjun Lei, "A CNN-based fast inter coding method for VVC," *IEEE Signal Processing Letters*, vol. 28, pp. 1260–1264, 2021.

[9] Thomas Amestoy, Alexandre Mercat, Wassim Hamidouche, Daniel Menard, and Cyril Bergeron, "Tunable VVC Frame Partitioning Based on Lightweight Machine Learning," *IEEE Transactions on Image Processing*, vol. 29, pp. 1313–1328, 2020.

[10] Matheus Lindino, Bruno Zatt, Mateus Grellert, and Guilherme Correa, "Low-Complexity Multi-Type Tree Partitioning for Versatile Video Coding Based on Machine Learning," in *2022 IEEE International Conference on Image Processing (ICIP)*, 2022, pp. 3616–3620.

[11] Alberto González-Ruiz, Antonio Jesús Díaz-Honrubia, Santiago Tapia-Fernández, David García-Lucas, Gabriel Cebrián-Márquez, and Luis Mengual-Galán, "Adaptive quadtree splitting parallelization (AQSP) algorithm for the VVC standard," *The Journal of Supercomputing*, pp. 1–16, 2024.

[12] ITU-T, "Recommendation P.910, Subjective Video Quality Assessment Methods for Multimedia Applications," Tech. Rep., International Telecommunication Union, 1999.

[13] JVET, "VVC Test Model - Version 20.0," `https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_`

Tabla II: Results of the final algorithm including the NB model following the CTCs guidelines.

Class	Sequence	BD-rate (%)	Time reduction (%)
A1	Drums100	0,9812	12,7704
	Tango2	1,7156	41,9871
	Campfire	0,8235	7,0090
	ToddlerFountain2	0,0458	1,8989
A2	CatRobot	2,0687	19,8487
	TrafficFlow	2,3111	17,9235
	RollerCoaster2	3,9787	17,8234
	DaylightRoad2	2,8891	43,6475
B	Kimono	0,6643	36,4227
	ParkScene	0,6915	38,7248
	BasketballDrive	0,9540	13,3003
	Cactus	0,7605	20,1167
	BQTerrace	1,1319	15,7520
C	BasketballDrill	0,2673	5,5769
	BQMall	0,3282	6,8981
	PartyScene	0,1889	4,5926
	RaceHorsesC	−0,0032	33,8261
	Average	1,1618	19,8893

`VTM/-/tree/VTM-20.0`.

[14] U. M. Fayyad and K. B. Irani, "Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning," in *Proceedings of the International Joint Conference on Uncertainty in AI, Chambery, France*, Aug 1993.

[15] I. Guyon and A. Elisseeff, "An Introduction to Variable and Feature Selection," *Journal Machine Learning Research*, vol. 3, pp. 1157–1182, Mar 2003.

[16] Paul Jaccard, "Étude comparative de la distribution florale dans une portion des Alpes et des Jura," *Bull Soc Vaudoise Sci Nat*, vol. 37, pp. 547–579, 1901.

[17] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Transactions on circuits and systems for video technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[18] Gisle Bjøntegaard, "Improvements of the BD-PSNR Model," Tech. Rep. VCEG-AI11, ITU-T SG16 Q6, 2008.

[19] Damián Ruiz, Gerardo Fernández-Escribano, José Luis Martínez, and Pedro Cuenca, "Fast intra mode decision algorithm based on texture orientation detection in HEVC1," *Signal Processing: Image Communication*, vol. 44, pp. 12–28, 2016.

[20] Guilherme Correa, Pedro A. Assuncao, Luciano Volcan Agostini, and Luis A. da Silva Cruz, "Fast HEVC encoding decisions using data mining," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 4, pp. 660–673, 2015.

[21] Behnam Heidari and Mohammadreza Ramezanpour, "Reduction of intra-coding time for HEVC based on temporary direction map," *Journal of Real-Time Image Processing*, vol. 17, no. 3, pp. 567–579, 2020.

[22] Wei-Jung Chien, Li Zhang, Martin Winken, Xiang Li, Ru-Ling Liao, Han Gao, Chih-Wei Hsu, Hongbin Liu, and Chun-Chi Chen, "Motion vector coding and block merging in the versatile video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3848–3861, 2021.

[23] W. Yeo and B.-G. Kim, "CNN-based fast split mode decision algorithm for versatile video coding (VVC) inter prediction," *J. Multimed. Inf. Syst.*, vol. 8, pp. 147–158, 2021.

[24] A. Tissier, W. Hamidouche, J. Vanne, and D. Menard, "Machine Learning Based Efficient QT-MTT Partitioning for VVC Inter Coding," in *2022 IEEE International Conference on Image Processing (ICIP)*, 2022, pp. 1401–1405.

[25] Alexandre Tissier, Wassim Hamidouche, Souhaiel Belhadj Dit Mdalsi, Jarno Vanne, Franck Galpin, and Daniel Menard, "Machine Learning Based Efficient QT-MTT Partitioning Scheme for VVC Intra Encoders," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 33, no. 8, pp. 4279–4293, 2023.

[26] Yingbiao Yao, Jiaojiao Wang, Chenjie Du, Jinghui Zhu, and Xin Xu, "A support vector machine based fast planar prediction mode decision algorithm for versatile video coding," *Multimedia Tools and Applications*, vol. 81, no. 12, pp. 17205–17222, 2022.

[27] Qiuwen Zhang, Yihan Wang, Lixun Huang, Bin Jiang, and Xiao Wang, "Fast CU partition decision for H.266/VVC based on the improved DAG-SVM classifier model," *Multimedia Systems*, vol. 27, no. 1, pp. 14–27, 2021.

[28] Gosala Kulupana, Saverio Blasi, et al., "Fast versatile video coding using specialised decision trees," in *2021 Picture Coding Symposium (PCS)*. IEEE, 2021, pp. 1–5.

[29] C.-M. Lien, M.-J. Chen, C.-M. Yang, R.-H. Hong, Y.-H. Lin, and C.-H. Yeh, "Multi-type tree partition decision for H.266/VVC inter coding based on Random Forest classifiers," in *IET International Conference on Engineering Technologies and Applications (ICETA 2023)*, 2023, vol. 2023, pp. 97–98.

[30] Joint Video Experts Team, "VVC Software VTM-20.0," `https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/-/tree/VTM-20.0?ref_type=tags`, 2024, [Online; accessed 22-July-2024].

[31] Pedro Domingos and Michael Pazzani, "On the optimality of the simple Bayesian classifier under zero-one loss," *Machine learning*, vol. 29, pp. 103–130, 1997.

[32] Jason D Rennie, Lawrence Shih, Jaime Teevan, and David R Karger, "Tackling the poor assumptions of naive bayes text classifiers," in *Proceedings of the 20th international conference on machine learning (ICML-03)*, 2003, pp. 616–623.

[33] Geoffrey I Webb, Eamonn Keogh, and Risto Miikkulainen, "Naïve Bayes," *Encyclopedia of machine learning*, vol. 15, no. 1, pp. 713–714, 2010.

[34] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[35] Nello Cristianini, *An Introduction to Support Vector Machines and other kernel-based learning methods*, Cambridge University Press, 2000.

[36] Rikard Vinge and Tomas McKelvey, "Understanding support vector machines with polynomial kernels," in *2019 27th European signal processing conference (EUSIPCO)*. IEEE, 2019, pp. 1–5.

[37] André E Lazzaretti and David MJ Tax, "An adaptive radial basis function kernel for support vector data description," in *Similarity-Based Pattern Recognition: Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3*. Springer, 2015, pp. 103–116.

[38] Shubhkirti Sharma and Vijay Kumar, "A comprehensive review on multi-objective optimization techniques: Past, present and future," *Archives of Computational Methods in Engineering*, vol. 29, no. 7, pp. 5605–5633, 2022.

Arquitecturas heterogéneas y modelos de programación (2/4)

Evaluación del compromiso área-memoria en FPGAs con HBM para aplicaciones de stencils utilizando OpenCL

Daniel López-Martínez [1], Manuel de Castro [1] y Diego R. Llanos [1]

Resumen— **Las FPGAs han demostrado su utilidad para acelerar kernels de HPC con elevada carga computacional. Sin embargo, se enfrentan a dificultades con los algoritmos limitados por la memoria (memory-bound), debido al ancho de banda comúnmente bajo de los dispositivos disponibles. Las tendencias recientes en diseño de dispositivos buscan mitigar este problema sustituyendo la memoria DDR4 por HBM, que ofrece anchos de banda máximos superiores. Sin embargo, la HBM requiere una gestión explícita de sus canales, lo que conlleva un mayor esfuerzo de programación y un uso adicional de recursos de la FPGA.**

En este artículo, mediante una aplicación de stencils altamente optimizada, estudiamos la sobrecarga de recursos asociada a la gestión de HBM como memoria heterogénea en FPGAs. Evaluamos las predicciones de trabajos anteriores sobre el uso de recursos de la aplicación elegida en un dispositivo FPGA concreto. De la evaluación concluimos que las predicciones fueron excesivamente optimistas. Destacamos que la gestión de la HBM de las FPGAs añade esfuerzos de desarrollo no despreciables, así como sobrecostes significativos de recursos FPGA. Proponemos algunas soluciones, analizando sus ventajas e inconvenientes.

Palabras clave— **HPC, FPGAs, HBM, Ancho de banda, Stencil, OpenCL.**

I. Introducción

La computación de alto rendimiento (HPC) ha encontrado en las FPGAs una plataforma versátil para la aceleración de núcleos computacionales (*kernels*) mediante el uso de arquitecturas a medida y segmentación masiva (*pipelining*) [1]. Sin embargo, a pesar de presentar potenciales beneficios (por ejemplo, una alta eficiencia energética), estas plataformas han lidiado históricamente con una limitación crítica: el ancho de banda de memoria. Mientras que las GPUs han evolucionado rápidamente hacia memorias de alto rendimiento, las FPGAs han mantenido el uso de memoria DDR4 de forma más prolongada, suponiendo un cuello de botella en aplicaciones de tipo memory-bound [2].

La reciente integración de la tecnología High Bandwidth Memory (HBM) en las FPGAs modernas promete mitigar esta limitación, ofreciendo anchos de banda teóricos de órdenes de magnitud superiores a los de las tecnologías de memoria tradicionales. Sin embargo, esta tecnología introduce una capa de complejidad adicional en el diseño, incluso al utilizar herramientas de síntesis de alto nivel (*HLS*) [3]: el HBM requiere una gestión explícita de sus pseudocanales (*PCs*). Esto no solo incrementa el esfuerzo de desarrollo, sino que también impacta significativamente en el consumo de recursos del dispositivo, especialmente en FPGAs de Intel/Altera.

En este artículo, evaluamos la sobrecarga real de recursos que implica la gestión de HBM como memoria heterogénea, utilizando como caso de estudio una implementación de stencil altamente optimizada basada en el trabajo de Zohouri *et al.* [4]. Varias características de los cómputos de stencil los convierten en un caso representativo para el uso de HBM: son cómputos memory-bound, con un gran grado de paralelismo, y un patrón de acceso a memoria regular y conocido en tiempo de compilación. Aunque trabajos previos han propuesto modelos de predicción para el escalado de recursos en FPGAs, nuestros experimentos sugieren que dichas estimaciones tienden a ser excesivamente optimistas e ingenuas respecto de la complejidad de la solución. Las contribuciones principales de este trabajo son:

- Un análisis del impacto en el uso de ALMs y registros, al escalar el número de PCs en FPGAs de Intel con HBM.
- La evaluación y refutación de predicciones previas sobre el uso de recursos, demostrando que la gestión de HBM requiere una estrategia consciente de los recursos disponibles.
- El planteamiento y la discusión de varias soluciones basadas en OpenCL para optimizar el compromiso entre el ancho de banda efectivo y la utilización de la lógica interna.

El resto de este artículo se organiza de la siguiente forma. La Sección II presenta el contexto en el que se desarrolla el trabajo y discute trabajos previos. La Sección III discute la metodología seguida para desarrollar las soluciones FPGA. La Sección IV presenta la experimentación realizada. Por último, la Sección V presenta las conclusiones del trabajo y las líneas futuras de investigación.

II. Contexto y Trabajo Relacionado

A. OpenCL

OpenCL es un estándar de programación abierto definido por el Grupo Khronos [5] para desarrollar software destinado a plataformas heterogéneas, basado en un anfitrión (*host*) y uno o más dispositivos (*devices*). Esto implica crear un código para el host (*host code*), normalmente en C/C++, que se ejecutará en el procesador anfitrión y se encargará de orquestar el trabajo de los dispositivos. Y, por otro

[1]Dpto. de Informática, Universidad de Valladolid, e-mail: {daniel.lopezm,manuel.castro,diego.llanos}@uva.es

lado, un código para los dispositivos (*kernel code*), escrito en una variante de C, que se ejecutará en los distintos dispositivos o aceleradores hardware, y llevará a cabo operaciones computacionalmente intensivas para completar la tarea implementada. El código host se compila con un compilador de C/C++, y los kernels se compilan en tiempo de ejecución con un compilador que soporte el dispositivo seleccionado. OpenCL proporciona APIs para gestionar los dispositivos y abstraer las comunicaciones entre el host y los mismos.

Gracias a la herramienta Intel FPGA SDK para OpenCL, podemos programar FPGAs con OpenCL, un lenguaje de programación de alto nivel. El proceso de compilación del código de kernel para FPGAs implica compilar OpenCL a una representación intermedia en LLVM y, posteriormente, a un lenguaje de descripción de hardware (comúnmente utilizado para programar FPGAs). Sintetizar descripciones de hardware a binarios que puedan programar la FPGA es un proceso considerablemente lento, que toma múltiples horas. Esto hace que el código del kernel tenga que compilarse previamente a la ejecución y por separado, y cargarse en tiempo de ejecución para reconfigurar la FPGA.

Para desarrollar diseños eficientes para FPGA, el Intel FPGA SDK para OpenCL proporciona un modelo de programación de kernels basado en un único *work-item*, la unidad mínima de cómputo de OpenCL, equivalente a un hilo de ejecución. En este modelo no existe paralelismo a nivel de hilo, y en su lugar, el compilador extraerá paralelismo basado en *pipelines* de los bucles del código kernel. Este modelo se aproxima más a la manera tradicional de programar FPGAs basada en pipelines profundas [6].

B. Cómputos de Stencil

Los cómputos de stencil son un tipo de cómputo en el que cada punto de un array multidimensional se actualiza mediante una función fija del elemento leído y sus vecinos. Los cómputos de stencil iterativos aplican esta operación a todos los elementos del array un número de iteraciones (*time-steps*) concreto. El radio y la geometría del stencil determinan el número de vecinos y las direcciones espaciales involucradas en cada operación. En este trabajo se consideran principalmente stencils de dos dimensiones y radio igual a uno. En concreto, nos centramos en el clásico ejemplo del stencil 2D de Jacobi de 5 puntas: cada elemento de la matriz resultado se calcula como la media de su valor anterior y el de sus 4 vecinos directos. Habitualmente, se dispone de dos búferes de datos de igual tamaño; el primero contiene los datos de lectura para la operación de stencil, y en el segundo se guardan los valores actualizados tras cada cómputo realizado. Cuando el cómputo de todos los elementos se ha completado, ambos búferes intercambian sus roles como paso previo a la ejecución de la siguiente iteración del stencil.

Los cómputos de stencil presentan una alta localidad espacial: cada elemento de la matriz se accede múltiples veces por iteración. Para explotar esta localidad, los diseños basados en FPGA utilizan registros de desplazamiento (*shift registers*). Estos funcionan como una caché eficiente que almacena los elementos de la matriz en memoria interna, hasta que ya no se requiera acceder a ellos más, siguiendo un esquema FIFO. Esta técnica permite reducir drásticamente el número de accesos a memoria global (externa), incrementando significativamente el rendimiento de las soluciones. Como los recursos de la FPGA son limitados, y el tamaño de los registros de desplazamiento depende de la anchura de la matriz a computar, las soluciones FPGA aplican *bloqueo espacial* para computar matrices de tamaños arbitrarios: las matrices se subdividen a lo ancho en bloques más pequeños, que son los que la FPGA es capaz de procesar.

Otra optimización habitual para cómputos de stencil iterativos en FPGA es el *bloqueo temporal*. Consiste en computar varias iteraciones de una vez en la FPGA, por cada ejecución del kernel de stencil. Los resultados intermedios se almacenan en memoria interna, de forma que la memoria global solo se accede 2 veces en todo el proceso: una para leer los valores iniciales de la matriz, y otra para escribir los resultados después de las iteraciones que se hayan computado. Esta técnica también reduce drásticamente el número de accesos a memoria global.

Si bien las FPGAs permiten una ejecución eficiente mediante pipelines profundas, permitiendo un paralelismo masivo, el rendimiento de stencils sigue estando limitado por el ancho de banda de la memoria, es decir, son problemas *memory-bound*. El ancho de banda de la memoria externa de las FPGAs es inferior al de las arquitecturas GPU de la misma generación [2], lo que habitualmente las deja en desventaja de rendimiento. No obstante, la implementación de las técnicas mencionadas anteriormente aumenta significativamente el rendimiento de los cómputos de stencil en FPGA, haciéndolas alternativas competitivas. Además, dichas técnicas, aunque podrían adaptarse a CPUs y GPUs, presentan mejoras de rendimiento mucho más limitadas en estas otras arquitecturas. Diversos estudios anteriores [7–9] han demostrado que las FPGAs pueden lograr un nivel de rendimiento igual al de las GPUs mediante implementaciones avanzadas de bloqueo temporal. La aplicación de bloqueo temporal en estos diseños transforma los cómputos de stencil en algoritmos limitados por el cómputo (*compute-bound*), dada la alta tasa de reutilización de datos. Sin embargo, estos diseños presentarían problemas de eficiencia en escenarios con un número reducido de iteraciones, donde la aplicación de bloqueo temporal es limitada, prevaleciendo la limitación por memoria. Otros trabajos, como el de Reggiani *et al.* [10], proponen arquitecturas multi-FPGA que permiten aumentar el ancho de banda total sin necesidad de computación redundante.

En este trabajo tratamos de explorar soluciones que implementen tanto bloqueo espacial como bloqueo temporal, tomando como referencia el trabajo

de Zohouri *et al.* [4], en el que se presenta una solución parametrizada basada en Intel FPGA SDK para OpenCL que consigue un rendimiento alto en la computación de stencils en FPGA.

C. HBM2

High Bandwidth Memory [11] es un estándar de DRAM apilada en 3D, diseñado para proporcionar un alto ancho de banda de memoria. La memoria DRAM HBM se acopla a un chip mediante una interfaz distribuida, la cual se divide en canales completamente independientes entre sí. En HBM2, cada canal tiene un ancho de datos de 128 bits, operando a un ratio de datos DDR (Double Data Rate). El ratio máximo de entrada y salida de datos mejora de 1 Gbps en HBM1 a 2 Gbps en HBM2, en parte gracias al uso de dos pseudocanales (PCs) por canal físico [12, 13]. Cada stack o interfaz de HBM2 cuenta con 8 canales físicos y, por lo tanto, 16 PCs a los que se puede acceder de forma independiente. Esto sitúa el ancho de banda máximo teórico para un único stack de HBM2 en 256 GB/s, cifra que se deriva de la siguiente relación:

$$BW_{stack} = \frac{(8\,canales \times 128\,bits) \times 2\,Gbps/pin}{8\,bits/Byte} = 256\,GB/s$$

Diversas FPGAs modernas utilizan hoy en día la tecnología HBM para incrementar su ancho de banda máximo de memoria y así mitigar el cuello de botella más común de las FPGAs frente a las GPUs. Además, las herramientas HLS actuales permiten desarrollar en aceleradores mediante lenguajes de programación de alto nivel como C/C++ y OpenCL, lo que las hace más accesibles. Trabajos anteriores [3, 14–16] han demostrado resultados notables en aplicaciones específicas gracias al incremento del ancho de banda de la memoria que la tecnología HBM proporciona. Esto se debe al uso simultáneo de varios PCs que proporciona la HBM, los cuales, a pesar de ser más lentos que los canales de DRAM tradicionales, en conjunto proporcionan un ancho de banda mucho mayor [13]. No obstante, la mayoría de estos estudios [14, 16] se centran en soluciones a problemas específicos con FPGAs de AMD/Xilinx, cuya arquitectura y programación con herramientas HLS son significativamente distintas de las de FPGAs de Intel/Altera. Como indican Choi *et al.* en su estudio [3], tanto Vitis de Xilinx como el Intel FPGA SDK para OpenCL, herramientas HLS de los distintos vendedores, requieren un esfuerzo explícito por parte del desarrollador para conectar los PCs a la lógica deseada. A mayores, en dispositivos Intel/Altera, se observa que el consumo de recursos aumenta de forma casi lineal con el número de PCs utilizados, a diferencia de los dispositivos de Xilinx, donde el impacto en la utilización de recursos es relativamente bajo al escalar el número de canales. Esta disparidad subraya la necesidad de realizar estudios en profundidad sobre arquitecturas de memoria específicas en dispositivos de Intel/Altera, donde la gestión eficiente de los pseudocanales resulta crítica para no comprometer el área disponible de las FPGAs.

III. Metodología

A. Descripción del problema

La programación de FPGAs requiere un esfuerzo no solo en términos de complejidad de desarrollo, sino también en el uso del área. En este trabajo nos centraremos en los aceleradores de Intel, específicamente en la Stratix 10 MX 2100, que utiliza la tecnología HBM2. Gracias a la herramienta Intel FPGA SDK para OpenCL, podemos programar la lógica de estos aceleradores en lenguaje OpenCL. Sin embargo, para utilizar eficientemente la tecnología HBM, se requiere la especificación explícita de los PCs a los que se accede mediante atributos de tipo "buffer_location" en los kernels. Esto queda ilustrado en la Figura 1. Al especificar los buffers `f1_1` y `f1_2` de esa manera, estamos diciendo al diseño hardware que configure esos buffers a los canales 0 y 1 de HBM. Por su parte, en el código de host, se deben crear los buffers correspondientes que se utilizarán con la memoria HBM (`f1_1` y `f1_2` en el ejemplo), especificando que son "memoria heterogénea", utilizando la flag `CL_MEM_HETEROGENEOUS_INTELFPGA`, proporcionada por el SDK de Intel. La guía oficial de Intel FPGA SDK para OpenCL [17] recomienda encarecidamente enlazar estos buffers a los argumentos del kernel correspondiente antes de leer o escribir nada en ellos. De esta forma, el runtime de OpenCL es capaz de conocer a qué PC de HBM están enlazados cuanto antes, y realizar las correspondientes lecturas y escrituras de forma correcta. Se deriva de lo anterior una limitación adicional: dado que el vínculo entre los PCs y los argumentos del kernel se define en tiempo de compilación, el programador debe garantizar en el código de host que cada buffer se asocie exclusivamente a argumentos de kernel ligados a un mismo PC. Asociar un mismo buffer a varios argumentos ligados a distintos PCs provocará movimientos de memoria adicionales, opacos al programador, que reducen el rendimiento.

```
                <kernel code>
__kernel void read(
  __global __attribute__((buffer_location("HBM0"
       ))) float *f1_1,
  __global __attribute__((buffer_location("HBM1"
       ))) float *f1_2,
  ...
    ) {}
                 <host code>
for (int h = 0; h < HBMS; h++) {
  f1_d_[h] = clCreateBuffer(context,
       CL_MEM_READ_WRITE |
       CL_MEM_HETEROGENEOUS_INTELFPGA, HBM_SIZE,
        NULL, &err);
  }
```

Fig. 1: Código de ejemplo de uso de pseudocanales de HBM.

Esta manera de programar el uso de memoria, con un nivel de abstracción bajo que requiere la especificación explícita y gestión manual de los pseudocanales a través de distintos buffers, hace que la migración de códigos existentes para su ejecución en este tipo

de dispositivos sea costosa. Adicionalmente, también puede provocar un consumo de recursos de la FPGA mayor de lo esperado, especialmente si la gestión no es óptima. Ambos fenómenos se estudiarán en las siguientes secciones.

B. Implementación y predicciones previas

Para estudiar el uso de recursos de la FPGA Stratix 10 MX 2100 en el ámbito de la computación de stencils, se decidió tomar como referencia la implementación aportada por Zohouri *et al.* [4]. Esta implementación en OpenCL combina técnicas de registros de desplazamiento, bloqueo espacial y temporal, y vectorización manual, junto con otras optimizaciones específicas para FPGA basadas en pipelines profundas. Con el fin de probar diversas configuraciones con distintas cargas de cómputo, la solución proporcionada está parametrizada, lo que nos permite realizar un estudio basado en los parámetros y su relación con el área de recursos de la FPGA.

La implementación de referencia proporcionada por los autores utiliza un modelo de programación basado en un único work-item, tal y como se explica en [4]. Este modelo de programación suele ser preferible para FPGAs, ya que es más capaz de explotar sus ventajas. El diseño planteado utiliza tres kernels principales: uno de lectura de datos de entrada, uno de escritura de resultados y otro de cómputo. El kernel de cómputo está replicado automáticamente, generando múltiples elementos de procesamiento (PEs) hardware. Cada PE computa una iteración del stencil, procesando vectores de elementos en pipeline. Las PEs están diseñadas para producir un vector de resultados por ciclo de reloj (es decir, tienen un intervalo de iniciación de 1 ciclo). Los datos se leen de memoria externa y se pasan a través de los PEs utilizando canales a nivel de chip (elementos de comunicación inter-kernel de alto rendimiento, proporcionados por el Intel FPGA SDK), y se escriben de nuevo en memoria externa cuando terminan. Un esquema del diseño de la implementación de referencia se muestra en la Figura 2.

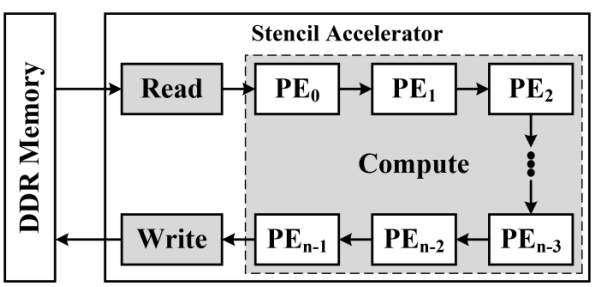


Fig. 2: Diseño de la implementación de referencia, extraído de [4].

Como se comentó previamente, esta solución implementa tanto bloque espacial como temporal. El bloqueo espacial se empleó para evitar la restricción de tamaño a los datos de entrada, dado que la propuesta de Zohouri *et al.* utiliza un registro de desplazamiento [18]. En cada ciclo de reloj se escribe un nuevo vector de elementos de la matriz de entrada en el registro de desplazamiento y se elimina uno antiguo. Por otro lado, el bloqueo temporal se implementa mediante la replicación de los PEs. Cada PE realiza las operaciones de una iteración del stencil para el mismo bloque espacial. Para realizar la replicación automática de los PEs, se utiliza un kernel de tipo *autorun*, proporcionado por el Intel FPGA SDK. Estos kernels se ejecutan automáticamente sin que el host los gestione, y permiten la replicación automática y parametrizada de hardware.

En base al uso del registro de desplazamiento, el bloqueo temporal y la vectorización, la implementación fue parametrizada. Los tres parámetros más relevantes para el estudio del uso de área son los siguientes:

- *bsize*: representa el tamaño (anchura) del bloque espacial, que a su vez determina el tamaño del registro de desplazamiento. Influye en la cantidad de redundancia en los cómputos a realizar.
- par_{vec}: representa el tamaño del vector de los PEs; que es el número de elementos que se procesan simultáneamente en el dispositivo. Valores mayores llevarán a mejoras en el rendimiento, aunque aumentará significativamente el uso de los recursos de cómputo de la FPGA, así como la presión en el ancho de banda de memoria global.
- par_{time}: representa el número de iteraciones a computar en la FPGA, que se traduce en el número de réplicas de PEs a crear. Valores altos llevarán a mejoras en el rendimiento, pero también a un uso multiplicativamente mayor de los recursos de la FPGA, y a una mayor cantidad de operaciones redundantes.

En base a estos parámetros, Zohouri *et al.* presentaron un modelo de optimización del diseño en función de la cantidad de recursos FPGA disponibles. Utilizando este modelo, proporcionaron una serie de predicciones para el uso de recursos y rendimiento de la Stratix 10 MX 2100 con esta aplicación. Este modelo y predicciones se afinaron más en la tesis de Zohouri [19]. La Tabla I muestra las predicciones relativas al stencil *Diffusion 2D* presentadas en la tesis. Estas predicciones muestran que, con tamaños de vector relativamente altos, se espera alcanzar un uso casi óptimo de los recursos de tipo DSP y un uso bastante por debajo del 100 % de los bloques de memoria.

C. Solución propuesta

Tomando como referencia la implementación de Zohouri *et al.*, disponible en el repositorio de GitHub del propio Zohouri [1], específicamente la relativa al kernel *Diffusion 2D*, realizamos cambios para aprovechar toda la capacidad de memoria presente en la HBM. Cabe destacar que, sin realizar ningún cambio, el código de referencia utiliza un único PC de HBM, lo que limita significativamente el tamaño de entrada máximo de la aplicación. Siguiendo los detalles explicados en la sección III-A sobre el uso de

[1] `https://github.com/zohourih/Diffusion_FPGA`

Radio	bsize	$\mathbf{par}_{time}$	$\mathbf{par}_{vec}$	Frecuencia (MHz)	Uso de ancho de banda (GB/s—%)	Memoria (Bits—Blocks)	DSP
1	16320	8	96	450	345.6— 68 %	20 %— 67 %	97 %
2	16308	4	108	450	388.8— 76 %	20 %— 67 %	98 %
3	16340	4	76	450	273.6— 53 %	25 %— 68 %	100 %
4	16356	2	116	450	417.6— 82 %	20 %— 68 %	100 %

Tabla I: Predicciones de características y uso de recursos para el stencil Diffusion 2D, extraídas de la tesis de Zohouri [19].

HBM2 en OpenCL, se adaptaron los kernels de lectura y escritura, y el código host, para poder utilizar hasta 16 PCs para lectura y 16 PCs para escritura. De esta forma, se hace uso del total de 32 PCs de los que dispone HBM2 en la FPGA Stratix 10 MX 2100. Los datos de entrada se reparten entre 16 PCs en el código de host, cuando se desea hacer máximo uso de la HBM. La computación de stencils iterativos requiere que, entre iteraciones, los buffers de lectura y escritura se intercambien. Sin embargo, cambiar los buffers que se pasan como argumento de entrada a los kernels en tiempo de ejecución no es recomendable, como se explicó en la Sección III-A. Es por esto que se optó por incluir los 32 PCs como argumentos de los kernels de lectura y escritura, y, mediante un `switch` dentro de los kernels y una *flag* cuyo valor varía entre iteraciones, elegimos leer/escribir los primeros 16 PCs o los últimos 16 PCs. De esta manera, en las iteraciones pares, el kernel de lectura utiliza los PCs del 0 al 15 y el kernel de escritura los PCs del 16 al 31, intercambiando los PCs entre kernels en las iteraciones impares.La necesidad de dividir los datos de entrada en 16 buffers también implica cambios adicionales en el código de host. Las modificaciones propuestas incrementan la complejidad del código, pero permiten utilizar todos los PCs que HBM2 proporciona en la Stratix 10 MX 2100, en vez de solo uno.

IV. Experimentación

A. Descripción de la plataforma

Los experimentos presentados a continuación se realizaron en el supercomputador MeluXina [20], una infraestructura de clase petascale perteneciente a la red de la Empresa Común de Informática de Alto Rendimiento Europea (EuroHPC JU), y gestionada por LuxProvide. La arquitectura del sistema cuenta con una partición especializada de FPGAs, compuesta por nodos equipados con tarjetas aceleradoras BittWare 520N-MX. Cada una de estas tarjetas integra un dispositivo Intel Stratix 10 MX2100 con memoria de alto ancho de banda (HBM2). Esta FPGA cuenta con dos stacks de HBM2, lo que implica 16 canales físicos y, por lo tanto, 32 PCs. Cada PC tiene una capacidad de 512 MB. El ancho de banda máximo teórico de la memoria de los dispositivos es de 512 GB/s, según lo explicado en la sección II-C. Si bien las especificaciones de la FPGA indican que es capaz de alcanzar frecuencias de reloj de hasta 1 GHz [21], la frecuencia de reloj real obtenida depende de la lógica de usuario implementada. Por tanto, el ancho de banda alcanzable puede variar considerablemente.

El flujo de trabajo en la plataforma se orquesta mediante el gestor de recursos Slurm, que garantiza una asignación eficiente de los nodos de cómputo. Debido a la alta demanda de recursos computacionales y memoria necesaria para la síntesis de hardware, el proceso de compilación de los kernels de FPGA (utilizando la herramienta Intel FPGA SDK para OpenCL) se realiza mediante el envío de trabajos de procesamiento por lotes (batch jobs) a través de Slurm. Una vez generado el archivo binario o bitstream, el mapeo y la ejecución del kernel en el acelerador se gestionan mediante la solicitud de acceso a los nodos de la partición FPGA, lo que permite la interacción directa entre el código del host y el acelerador hardware.

B. Comprobando las predicciones

Para comprender mejor los valores de los parámetros probados, es importante destacar que, en la especificación proporcionada por Zohouri *et al.*, se detallan ciertas restricciones que se han tomado como guía. En primer lugar, se asume que el parámetro *bsize* es una potencia de 2, ya que la implementación está diseñada para calcular los índices de los bloques mediante operaciones de módulo optimizadas para potencias de 2. Además, el valor de *bsize* debe ser divisible por el valor del parámetro par_{vec}. Con respecto al parámetro par_{vec}, se limita su valor a potencias de 2 para evitar malgastar el ancho de banda. Por último, se recomienda tomar valores del parámetro par_{time} que sean múltiplos de cuatro, para evitar accesos a memoria no alineados.

En la Tabla II se presentan 11 configuraciones de parámetros y su consumo estimado del área de la FPGA, según los reportes que genera Intel FPGA SDK para OpenCL durante la compilación. Estos conjuntos de configuración se fueron decidiendo iterativamente, con el objetivo de encontrar una configuración de parámetros cuyo consumo de recursos fuera menor del 100 %. Específicamente, el caso **C11** es la configuración más cercana a la configuración límite predicha por Zohouri en [19], representada en la Tabla I. Por otra parte, el caso **C1** coincide con la configuración propuesta por Zohouri *et al.* en [4]. En ambos casos, **C1** y **C11**, se observa un uso de recursos de memoria mucho mayor que el predicho. Este uso de recursos por encima de los disponibles causó que ninguna de esas configuraciones completara su compilación. Partiendo de estos dos casos como límites superiores, se eligieron configuraciones de

parámetros con valores cada vez más pequeños hasta encontrar una configuración compilable.

Caso	bsize	$\mathbf{par}_{time}$	$\mathbf{par}_{vec}$	Lógica	Bloques RAM	DSP
C1	8192	92	8	114 %	257 %	113 %
C2	8192	48	8	95 %	200 %	79 %
C3	8192	24	8	85 %	169 %	44 %
C4	8192	8	8	78 %	149 %	28 %
C5	**16384**	**8**	**4**	**51.41 %**	**85.02 %**	**23.98 %**
C6	**16384**	**1**	**1**	**32.21 %**	**40.41 %**	**20.07 %**
C7	16384	4	8	76 %	144 %	24 %
C8	16384	8	8	78 %	150 %	28 %
C9	16384	8	16	130 %	385 %	36 %
C10	16384	8	32	235 %	1243 %	52 %
C11	16384	8	64	444 %	5533 %	85 %

Tabla II: Uso de recursos de FPGA en función de la configuración de parámetros de stencil.

En la Figura 3 se representan los datos de la Tabla II, de tal manera que en el eje X se muestran los distintos casos de configuración de parámetros y en el eje Y el porcentaje de uso del área de recursos. Cada recurso se representa con una barra de distinto color, lo que nos permite observar que los recursos de memoria (bloques de RAM) presentan los porcentajes más altos en todas las configuraciones de parámetros. Habiendo determinado que el uso de recursos de memoria es el más problemático para la implementación planteada, se presenta una gráfica en la Figura 4, que representa el uso de recursos de memoria para cada caso de configuración probado.

Con los reportes de los 11 casos de configuraciones de parámetros, se pudo comprobar que tan solo los casos **C5** y **C6** presentan un uso de recursos dentro del límite permitido. En ambos casos, el valor del parámetro *bsize* coincide con el límite óptimo predicho anteriormente. Cabe destacar que el caso **C6** es una configuración de parámetros mínima, que carece de bloqueo temporal (valor de par_{time} igual a 1). Asimismo, el caso **C5** tiene un valor del parámetro par_{vec} mucho más pequeño que el de la predicción planteada. Nótese que los datos de uso del área de recursos para los casos **C5** y **C6** son datos finales obtenidos al completar toda la compilación de los mismos (en vez de las estimaciones iniciales que ofrece el compilador, como en los otros casos).

C. Compromiso área-memoria

A la luz de los resultados relativos al uso de recursos de la FPGA, se puede concluir que las predicciones anteriores no fueron acertadas. Sin embargo, estas predicciones se realizaron sin tener en cuenta la dificultad de implementar la correcta gestión de los PCs de la tecnología HBM.

En los datos expuestos en la Sección IV-B, vemos que valores altos del parámetro par_{vec} aumentan drásticamente el uso de recursos de memoria. También se observa que tomar valores más bajos de este parámetro reduce tanto el uso de memoria como el de DSPs, principales recursos de cómputo en las FPGAs. De esta manera, el caso óptimo compilable presenta un uso de DSPs subóptimo. También se ha podido comprobar, al comparar los casos **C4** y **C8**, que el parámetro *bsize* no parece afectar en gran medida el uso de ninguno de los recursos. Si bien se observó que el parámetro par_{time} tiene una relación directa con el uso de recursos de memoria y DSPs, parece afectar menos que el parámetro par_{vec}.

Debido a limitaciones derivadas de la gran duración de la compilación y de la cuota de tiempo disponible en el superordenador utilizado, no se pudieron llevar a cabo más pruebas sobre el uso de recursos con configuraciones límite. También por el mismo motivo, solo se realizaron pruebas de ejecución para tres de los casos compilados: los casos **C5** y **C6** utilizando el máximo número de PCs de HBM disponibles en la FPGA (16 para lectura y otros 16 para escritura), y el caso **C6** utilizando un único PC de HBM (el mismo para lectura y escritura). Este último caso coincide con la implementación de referencia proporcionada por Zohouri *et al.* sin modificaciones, ya que, por defecto, se utiliza un único PC de HBM, como si fuera un tipo de memoria DDR.

En la Tabla III se muestran los resultados de ejecución de los 3 casos compilados, para distintos tamaños de entrada. Se incluye la frecuencia de reloj de cada diseño, que varía en función de la complejidad de la lógica implementada. Los tamaños de entrada probados están estrechamente ligados a la capacidad máxima de los PCs de HBM. En teoría, para el caso de un único PC para almacenar las matrices de entrada y salida, el tamaño máximo de cada matriz debería ser de 8192^2 elementos de punto flotante (256 MB). Sin embargo, este tamaño es tan cercano a la capacidad máxima de un PC, que la ejecución de un caso de entrada así produce un fallo del tipo "memoria insuficiente". Esto probablemente se deba a que una pequeña parte de la memoria de cada PC esté reservada para estructuras internas del driver, runtime, u OpenCL. Por ello, el único tamaño de entrada probado con uno PC es el que tiene por lado la potencia de dos anterior: 4096^2.

En los tiempos de ejecución del kernel para los distintos tamaños de entrada, se deberían observar incrementos de tiempo proporcionales a la altura de la matriz de entrada, y al número de bloques espaciales que se utilicen (lo cual depende de la anchura de la matriz de entrada). Para tamaños de 4096^2 y 8192^2, dado que el número de columnas es menor que el valor de *bsize*, solo se utiliza un bloque espacial. Por tanto, el incremento del tiempo por incrementar el tamaño de lado debería ser igual al incremento de ese tamaño. Por otro lado, a partir del tamaño 16384 ya se utilizan dos bloques espaciales, debido a los elementos extra derivados de los halos del stencil, que se incluyen dentro de cada bloque temporal. Estos elementos corresponden a $2 \times \text{radio} = 2$ columnas en cada bloque temporal, y son leídos, pero no computados. Para 32768, se requieren tres bloques ($3/2 = 1{,}5$ bloques más que para 16384). Por lo tanto, obtenemos los siguientes incrementos de tiempo teóricos para cada tamaño de entrada:

- Incremento de tiempo teórico de 4096 a 8192:

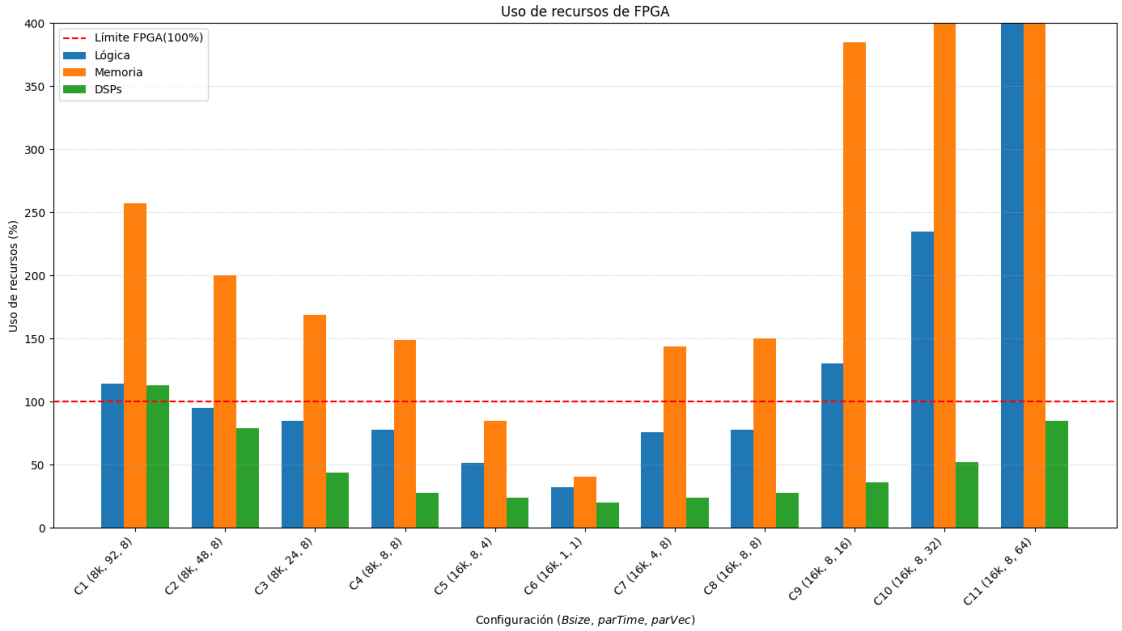


Fig. 3: Gráfica del uso general de recursos en cada caso de la Tabla II.

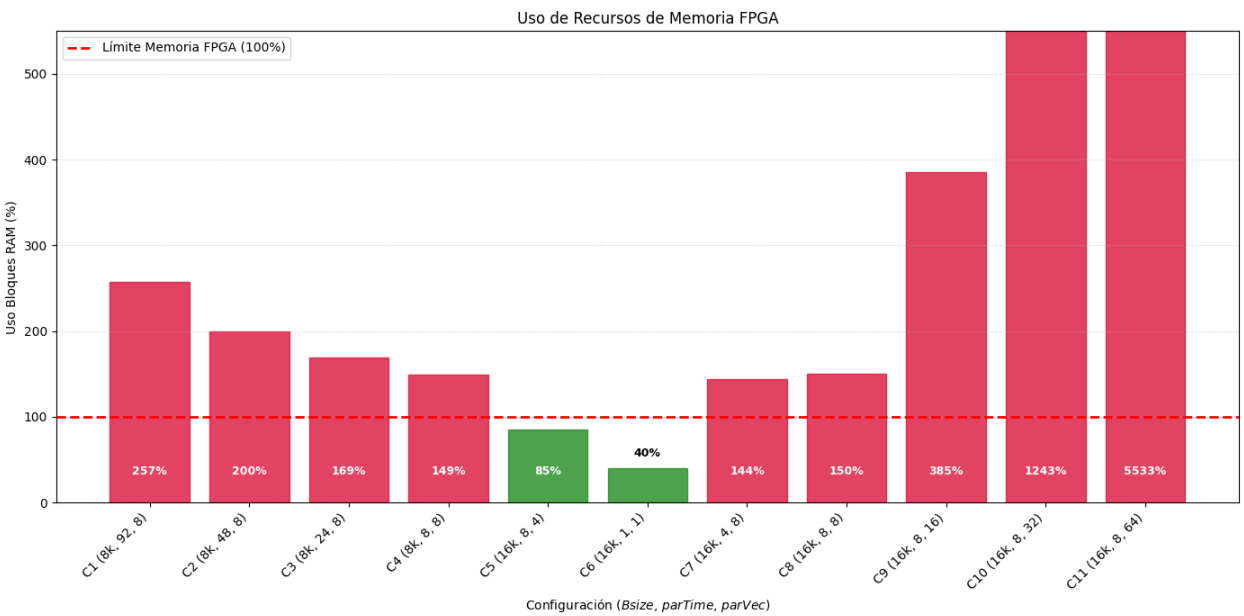


Fig. 4: Gráfica del uso de recursos de memoria en cada caso de la Tabla II.

HBM PCs	Parámetros (bsize, par_{time}, par_{vec})	Frec. reloj (MHz)	Tamaño entrada	Tiempo kernel (s)	GFLOPS	Rendimiento (GB/s)
32	(16384, 8, 4)	353.12	4096	1.29	11.498	10.221
			8192	4.578	12.988	11.545
			16384	20.924	11.376	10.112
			32768	78.252	12.151	10.801
	(16384, 1, 1)	425.00	4096	16.0394	0.941	0.836
			8192	32.0089	1.886	1.677
			16384	127.396	1.892	1.682
			32768	381.948	2.522	2.242
1	(16384, 1, 1)	458.33	4096	14.784	1.018	0.905

Tabla III: Resultados de ejecución de configuraciones del stencil en HBM

$\times 2 \times 1 = \times 2$

- Incremento de tiempo teórico de 8192 a 16384: $\times 2 \times 2 = \times 4$
- Incremento de tiempo teórico de 16384 a 32768: $\times 2 \times 1{,}5 = \times 3$

Los resultados correspondientes a la configuración de parámetros (16384, 1, 1) utilizando 32 PCs se ajustan aproximadamente a estos incrementos de

tiempo teóricos. Sin embargo, la configuración de parámetros (16384, 8, 4) presenta unos incrementos de tiempo destacadamente mayores a los teóricos:

- Incremento de tiempo real de 4096 a 8192: ×3,5488
- Incremento de tiempo real de 8192 a 16384: ×4,5706
- Incremento de tiempo real de 16384 a 32768: ×3,7398

Por otra parte, el modelo teórico planteado por Zohouri en [19] nos indica que, para conjuntos de parámetros con el mismo valor de *bsize*, el incremento teórico del rendimiento (es decir, el decremento en el tiempo de ejecución) es directamente proporcional al incremento en los valores de los otros dos parámetros, par_{time} y par_{vec}, asumiendo que no se sobrepasa el ancho de banda de la memoria. Para hacer una comparación entre las dos configuraciones de parámetros (**C6** con **C5**, para 32 PCs ambos), también hay que tener en cuenta las distintas frecuencias de reloj de cada diseño. Partimos de la fórmula planteada por Zohouri para calcular el tiempo de ejecución teórico, simplificada y representada en la Ecuación 1, donde t_{read}, t_{write} y num_{acc} dependen del tamaño de entrada y del parámetro *bsize*. Si dividimos el tiempo de ejecución teórico de la configuración (16384, 1, 1) entre el de la configuración (16384, 8, 4), obtenemos el resultado presentado en la Ecuación 2. Este resultado indica que la relación entre el speedup teórico y el valor de los parámetros es directa, tomando en cuenta también la frecuencia de reloj.

$$run_time = \frac{[\frac{iter}{par_{time}}] \times (t_{read} \ + \ t_{write})}{f_{max} \times par_{vec} \times num_{acc}} \quad (1)$$

$$\begin{aligned} speedup &= \frac{run_time(16384,1,1)}{run_time(16384,8,4)} = \frac{\frac{1}{1\times425\times1}}{\frac{1}{8\times353{,}12\times4}} \\ &= \frac{353{,}12}{425} \times 8 \times 4 = 26{,}5879 \end{aligned} \quad (2)$$

Si bien el speedup teórico para (16384, 8, 4) con respecto a (16384, 1, 1) es de ×26,5879, experimentalmente se ha obtenido un speedup de tan solo un ×6,99. Esto implica que el modelo teórico planteado por Zohouri no es capaz de explicar eficazmente el rendimiento de este nuevo diseño que utiliza todos los PCs de HBM. Para encontrar una explicación a esta observación, sería necesario realizar un estudio más exhaustivo del efecto de cada parámetro en el rendimiento, y elaborar un nuevo modelo teórico. Aun así, la configuración que utiliza vectorización y bloqueo temporal es la que obtiene el mayor rendimiento, y el mayor uso de los recursos de la FPGA.

Se observa también que los tiempos de ejecución del caso con un único PC son ligeramente inferiores a los del caso con mismos parámetros y 32 PCs. Esta diferencia se explica completamente por la reducción en la frecuencia de reloj del diseño que usa 32 PCs. Esta reducción de frecuencia es consecuencia directa del sobrecoste en uso de recursos derivado del acceso a un mayor número de pseudocanales.

V. Conclusiones y Líneas Futuras

Este estudio analiza el impacto del uso de los pseudocanales (PCs) de la tecnología HBM2 en el uso de recursos FPGA. Para ello, se utilizan kernels de stencil implementados en OpenCL como caso de ejemplo, que se evalúan en FPGAs Intel Stratix 10 MX. Los resultados experimentales demuestran que la gestión explícita de la memoria HBM en el entorno OpenCL acarrea un uso de recursos significativamente mayor al proyectado en trabajos previos. Se refuta la validez de modelos de predicción anteriores, y se muestra que el consumo de bloques de RAM y de lógica escala de forma drástica cuando se usa más de un pseudocanal, especialmente al incrementar el parámetro de vectorización. También se presenta una evaluación experimental del rendimiento de 3 diseños con distintos parámetros de optimización y uso de pseudocanales. Pese a la complejidad de desarrollo y las restricciones de síntesis, las implementaciones que usan 32 PCs logran rendimientos competitivos, y permiten escalar el cómputo a tamaños de entrada significativamente superiores a los admitidos por un único pseudocanal. No obstante, los modelos de rendimiento teóricos propuestos con anterioridad muestran cierta incapacidad para explicar algunos de los resultados obtenidos.

Posterior a la implementación aquí presentada, se planteó una solución para usar de forma más eficiente los pseudocanales (PCs) de HBM mediante un reparto alternado de la matriz de entrada en lugar de bloques contiguos. Esto permitiría que los kernels accedan a 16 PCs en paralelo, multiplicando el ancho de banda efectivo con un uso de recursos en la FPGA similar al diseño actual, como hemos podido comprobar en resultados tempranos. Aunque esta solución plantea una alta complejidad en la lógica del código del host, su desarrollo futuro permitirá elaborar un modelo teórico que explique mejor los resultados experimentales obtenidos. Asimismo, esta investigación sienta las bases para desarrollar abstracciones en lenguajes de alto nivel que permitan una gestión de la memoria HBM más cómoda y directa que la ofrecida por las soluciones actuales, como el Intel FPGA SDK para OpenCL, facilitando así el aprovechamiento del hardware sin aumentar la carga para el programador.

En conclusión, alcanzar el máximo ancho de banda efectivo requiere un compromiso crítico entre el área disponible y la configuración de los parámetros de la aplicación, lo que exige una estrategia de diseño que priorice la eficiencia en el uso de la memoria interna sobre la replicación masiva de elementos de cómputo.

Agradecimientos

Este trabajo ha sido financiado en parte por el Ministerio de Ciencia e Innovación de España y por el programa del Fondo Europeo de Desarrollo Regional (FEDER) de la Unión Europea, en el marco de la subvención PID2022-142292NB-I00 (Proyecto NA-

TASHA). Este trabajo también ha sido financiado en parte por la subvención TED2021–130367B–I00, financiada por MCIN/AEI/10.13039/501100011033 y por "NextGenerationEU/PRTR de la Unión Europea". Manuel de Castro ha recibido el apoyo del Ministerio de Ciencia, Innovación y Universidades de España, a través de las "Ayudas para la Formación de Profesorado Universitario FPU 2022". Las simulaciones se llevaron a cabo en el superordenador nacional de Luxemburgo, MeluXina. Los autores agradecen sinceramente a los equipos de LuxProvide su apoyo especializado. También queremos expresar nuestro agradecimiento a EuroHPC JU, por apoyar este proyecto en el superordenador MeluXina de LuxProvide a través de una convocatoria de desarrollo de EuroHPC (EHPC-DEV-2025D08-106), y también a el equipo de EPICURE, una iniciativa de EuroHPC JU, por su apoyo. Extendemos nuestro agradecimiento también a Arturo González Escribano por su apoyo en diversas discusiones y necesidades de recursos a lo largo de este trabajo.

Referencias

[1] Manuel de Castro, David L. Vilariño, Yuri Torres, and Diego R. Llanos, "The role of field-programmable gate arrays in the acceleration of modern high-performance computing workloads," *Computer*, vol. 57, no. 7, pp. 66–76, 2024.

[2] Jason Cong, Zhenman Fang, Michael Lo, Hanrui Wang, Jingxian Xu, and Shaochong Zhang, "Understanding performance differences of fpgas and gpus," in *2018 IEEE 26th annual international symposium on field-programmable custom computing machines (FCCM)*. IEEE, 2018, pp. 93–96.

[3] Young-kyu Choi, Yuze Chi, Jie Wang, Licheng Guo, and Jason Cong, "When hls meets fpga hbm: Benchmarking and bandwidth optimization," *arXiv preprint arXiv:2010.06075*, 2020.

[4] Hamid Reza Zohouri, Artur Podobas, and Satoshi Matsuoka, "Combined spatial and temporal blocking for high-performance stencil computation on fpgas using opencl," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, New York, NY, USA, 2018, FPGA '18, p. 153–162, Association for Computing Machinery.

[5] Khronos OpenCL working group, "The OpenCL Specification: Version 1.0," `https://registry.khronos.org/OpenCL/specs/opencl-1.0.pdf`, 2011, (Accedido el 22 de marzo de 2026).

[6] Manuel de Castro, Roberto R. Osorio, Francisco J. Andújar, Rocío Carratalá-Sáez, Yuri Torres, and Diego R. Llanos, "Comparing portability of fpga high-level synthesis frameworks in the context of a highly-parallel application," *IEEE Access*, vol. 13, pp. 50394–50410, 2025.

[7] Riccardo Cattaneo, Giuseppe Natale, Carlo Sicignano, Donatella Sciuto, and Marco Domenico Santambrogio, "On how to accelerate iterative stencil loops: a scalable streaming-based approach," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 12, no. 4, pp. 1–26, 2015.

[8] Yuze Chi, Jason Cong, Peng Wei, and Peipei Zhou, "Soda: Stencil with optimized dataflow architecture," in *Proceedings of the International Conference on Computer-Aided Design*, 2018, pp. 1–8.

[9] Tobias Kenter, Jens Förstner, and Christian Plessl, "Flexible fpga design for fdtd using opencl," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–7.

[10] Enrico Reggiani, Emanuele Del Sozzo, Davide Conficconi, Giuseppe Natale, Carlo Moroni, and Marco D Santambrogio, "Enhancing the scalability of multi-fpga stencil computations via highly optimized hdl components," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 14, no. 3, pp. 1–33, 2021.

[11] JEDEC, "High Bandwidth Memory (HBM) DRAM," `https://www.jedec.org/standards-documents/docs/jesd235a`, 2020, (Accedido el 22 de marzo de 2026).

[12] Altera, "High Bandwidth Memory (HBM2) Interface FPGA IP User Guide," `https://www.intel.com/programmable/technical-pdfs/683189.pdf`, 2024, (Accedido el 22 de marzo de 2026).

[13] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim, "Hbm (high bandwidth memory) dram technology and architecture," in *2017 IEEE International Memory Workshop (IMW)*. IEEE, 2017, pp. 1–4.

[14] Young-kyu Choi, Yuze Chi, Weikang Qiao, Nikola Samardzic, and Jason Cong, "Hbm connect: High-performance hls interconnect for fpga hbm," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 116–126.

[15] Lars Schwenger, Philipp Holzinger, Dietmar Fey, Hector Gerardo Munoz Hernandez, and Marc Reichenbach, "Easyhbm: Simple and fast hbm access for fpgas using high-level-synthesis," in *International Conference on Embedded Computer Systems*. Springer, 2022, pp. 43–57.

[16] Changdao Du and Yoshiki Yamaguchi, "High-level synthesis design for stencil computations on fpga with high bandwidth memory," *Electronics*, vol. 9, no. 8, pp. 1275, 2020.

[17] Altera, *Intel FPGA SDK for OpenCL Pro Edition: Programming Guide*, 22.4 edition, 2022, Section 7.2.2: Partitioning Buffers Across Different Memory Types (Heterogeneous Memory), Disponible en: `https://docs.altera.com/viewer/book-attachment/svo0FFlu4wPxzO91vmrc6A/flw2XqiPomlwTeu~QkEWuw-svo0FFlu4wPxzO91vmrc6A`.

[18] Hasitha Muthumala Waidyasooriya, Yasuhiro Takei, Shunsuke Tatsumi, and Masanori Hariyama, "Opencl-based fpga-platform for stencil computation and its optimization methodology," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 5, pp. 1390–1402, 2016.

[19] Hamid Reza Zohouri, *High performance computing with FPGAs and OpenCL*, Phd thesis, Tokyo Institute of Technology, August 2018.

[20] LuxProvide, "Meluxina user documentation," `https://docs.lxp.lu/`, 2025.

[21] Altera, "Stratix® 10 MX (DRAM System-inPackage) Device Overview," `https://docs.altera.com/viewer/book-attachment/myqdrpKGlZ3OfoWRcb67dw/R840z_WJx7OIIZ5JHpY11Q-myqdrpKGlZ3OfoWRcb67dw`, 2022, (Accedido el 22 de marzo de 2026).

Planificador de Tareas Basado en Green Contexts

Juan José Martín Osuna[1], José Flich Cardo[1] y Carles Hernández Luz[1]

Resumen—**En este artículo se presenta una herramienta orientada a la planificación de tareas de inferencia en GPU, motivada por la creciente necesidad de desarrollar soluciones que permitan explotar las elevadas capacidades de cómputo de las GPU, tanto de propósito general como en entornos de computación en el edge. Asimismo, se persigue ofrecer un alto nivel de garantías en términos temporales y de aislamiento entre procesos.**

Para ello, tras un análisis en el que se selecciona la tecnología Green Contexts de NVIDIA como la alternativa más adecuada para su desarrollo, se propone una herramienta completa que integra diversas funcionalidades y políticas de planificación. Los resultados obtenidos demuestran su utilidad tanto para mejorar el aprovechamiento de las GPU como para ofrecer garantías reales en términos de tiempo de ejecución al ejecutar múltiples procesos en paralelo.

Palabras clave— **Modelos de aprendizaje profundo; Conjuntos de redes neuronales; Aislamiento de procesos; GPU; CUDA; Inferencias; Planificador; Multi-Process Service (MPS); Multi-Instance GPUs (MIG); Green Contexts**

I. Introducción

EN los últimos años, las capacidades de procesamiento de las GPU han experimentado un crecimiento notable, impulsado principalmente por la amplia variedad de cargas de trabajo que pueden soportar y, especialmente, por la rápida expansión de las soluciones basadas en inteligencia artificial (IA). Aunque estas unidades están diseñadas principalmente para el entrenamiento de modelos y simulaciones de gran escala que demandan un alto rendimiento, muchas aplicaciones requieren menos recursos computacionales, lo que deja una parte significativa de la capacidad de la GPU sin aprovechar.

Al mismo tiempo, varias aplicaciones críticas para la seguridad deben operar bajo estrictas limitaciones de hardware, ya sea por espacio, consumo energético o costos. Esto implica que distintas tareas, con niveles de criticidad diferentes, deben ejecutarse simultáneamente en la misma plataforma de cómputo. Un ejemplo claro son los vehículos autónomos, en los que múltiples funcionalidades deben concentrarse en un número reducido de unidades de procesamiento potentes, en lugar de distribuirse entre varios dispositivos independientes [1].

Tradicionalmente, los sistemas críticos de seguridad han logrado un fuerte aislamiento al distribuir las funcionalidades entre distintos componentes de hardware. Sin embargo, a medida que crecen los requisitos computacionales de estos sistemas, los enfoques basados en computación federada [2] se vuelven cada vez más difíciles de escalar, lo que hace que la consolidación gestionada por software en plataformas compartidas sea una solución más práctica [1].

En este artículo se presenta una herramienta funcional de planificación de tareas para GPUs NVIDIA, tanto de propósito general como para el entorno de computación en el edge. Está basada en la tecnología de particionado de recursos de Green Contexts, que ha demostrado su eficiencia al ofrecer particionado de GPUs y garantizar aislamiento espacial y temporal de los procesos.

A continuación, este artículo describe las principales aportaciones:

1. Se analizan las principales tecnologías de particionado, ofreciendo una conclusión sobre cuál es la más adecuada para desarrollar un planificador dinámico y portable entre distintos sistemas.
2. Se describen las principales características del planificador, así como los comandos y la forma de utilizar la herramienta.
3. Se muestran ejemplos de uso real con las distintas configuraciones disponibles, lo que demuestra que se trata de una herramienta de particionado de recursos que, a su vez, ofrece garantías de aislamiento a los procesos ejecutados.

II. Motivación

Las aplicaciones relacionadas con la seguridad deben someterse a un riguroso proceso de verificación y validación para cumplir con los requisitos establecidos por las normas de seguridad funcional, como la IEC 61508 [3] y la ISO 26262 [4]. En el contexto del software, en particular cuando las aplicaciones se implementan en plataformas informáticas complejas, la verificación temporal se convierte en uno de los pasos más desafiantes [5].

La verificación temporal del software [5] requiere derivar límites superiores seguros para el tiempo de ejecución de las tareas. Pero garantizar dichos límites se vuelve cada vez más difícil cuando se integran múltiples funcionalidades dentro del mismo dispositivo informático [6]. Por ello, el enfoque tradicional [2], que se basa en desplegar diferentes funcionalidades en unidades computacionales separadas (por ejemplo, ECUs) para garantizar el aislamiento, no escala con la creciente complejidad de las aplicaciones críticas para la seguridad modernas, como los sistemas de conducción autónoma [1].

Por lo tanto, para abordar la consolidación de múltiples funciones en hardware compartido, se han propuesto varias técnicas de particionado y aislamiento para plataformas de CPU multicore. Estos enfoques suelen basarse en mecanismos de particio-

[1]Depto. de Informática de Sistemas y Computadores, Universitat Politècnica de València. E-mail: juamaros@upvnet.upv.es, jflich@disca.upv.es y carherlu@upv.es

namiento espacial y temporal [7] para controlar la asignación de núcleos de procesamiento, regiones de memoria y recursos compartidos, como las cachés y el ancho de banda de memoria. Al limitar la interferencia entre tareas ejecutadas simultáneamente, estas técnicas buscan proporcionar un comportamiento de ejecución predecible y permitir un análisis temporal confiable en sistemas consolidados [6].

Con esa premisa en mente, en este documento se presentará una herramienta de planificación de tareas de inferencia en GPU basada en Green Contexts (GC), una de las tecnologías de particionado de recursos más recientes en las GPU de NVIDIA.

A. Mecanismos de Compartición de Recursos en las GPUs de NVIDIA

A continuación describiremos los principales mecanismos de compartición de recursos en las GPUs de NVIDIA así como sus principales características y por qué se ha optado por uno de ellos como alternativa para desarrollar el planificador.

A.1 Multi-Process Service (MPS)

El Servicio de Procesos Múltiples (MPS) [8] es una solución de software que permite que múltiples procesos compartan simultáneamente los recursos de la GPU (SMs) en una sola GPU. Actúa como el mecanismo estándar de NVIDIA para habilitar la ejecución concurrente, lo que permite un paralelismo real cuando hay suficientes recursos de GPU disponibles.

En condiciones de recursos limitados, MPS mejora el rendimiento al reducir la sobrecarga del cambio de contexto. Aunque MPS proporciona aislamiento lógico entre procesos, garantizando la separación de los contextos de CUDA, no asegura el aislamiento a nivel de unidad de cómputo, por lo que un proceso que bloquee recursos puede afectar significativamente el rendimiento de los demás procesos.

A.2 Multi-Instance GPUs (MIG)

GPUs Multi-Instancia (MIG) [9] es una tecnología de hardware disponible en algunas GPUs modernas de NVIDIA que permite dividir una sola GPU en varias particiones completamente aisladas, tanto en cálculo como en memoria.

La unidad mínima que se puede asignar a una partición MIG es un GPU Processing Cluster (GPC), que a su vez está formado por varios Texture Processing Clusters (TPCs). Los TPCs son bloques de procesamiento modulares, cada uno con dos SMs, una unidad de texturas y una caché de texturas de nivel 1.

Al crear particiones MIG, hay distintas configuraciones posibles según cuántos GPCs se asignen a cada partición. Estas particiones deben definirse antes de ejecutar los programas, ya que no es posible modificar los recursos sobre la marcha. Si se quiere reasignar recursos, es necesario detener todas las tareas en curso, eliminar las particiones existentes y crear nuevas. Este proceso toma alrededor de un segundo, lo que lo hace poco práctico para tareas en tiempo real que requieran ajustes dinámicos.

Por último, es importante mencionar que MIG no está disponible en dispositivos NVIDIA para edge, por lo que, en esos casos, es necesario recurrir a otras tecnologías si se busca aislar recursos.

A.3 Green Contexts (GC)

Green Contexts (GC) [10], [11] es una solución de particionamiento basada en software introducida en CUDA versión 12.4 [12]. Esta tecnología permite crear contextos CUDA especializados que limitan el número máximo de Streaming Multiprocessors (SMs) accesibles a un proceso que se ejecuta en dicho contexto.

Al igual que con MIG, este enfoque garantiza la ejecución exclusiva de los procesos en los recursos de GPU asignados. Sin embargo, a diferencia de MIG, la creación de particiones con GC es mucho más flexible: el usuario puede generar múltiples particiones que apunten a los mismos SMs, dejando a la lógica de programación la responsabilidad de evitar que los procesos se solapen en los SMs que utilizan. Esta flexibilidad permite una asignación de recursos mucho más ágil sin necesidad de detener procesos en ejecución, a diferencia de MIG. No obstante, GC no proporciona aislamiento a nivel de otros componentes de hardware, como buses de comunicación o subsistemas de memoria, como sí lo hace MIG. Esto puede limitar las capacidades de aislamiento y de rendimiento de este enfoque.

Además, Green Contexts permiten una asignación de recursos significativamente más granular, considerando los SMs individuales como la unidad mínima de particionamiento de cómputo. Según la documentación, existe un número mínimo de SMs por partición, así como un tamaño de paso que permite incrementar el número de SMs asignables. Estas restricciones están determinadas por la arquitectura del chip, y las configuraciones básicas se resumen en la Tabla I.

Tabla I: Asignación mínima de SMs y tamaño de paso para las particiones Green Context según distintas arquitecturas de NVIDIA.

Arquitectura del chip	**SMs mínimos por partición**	**Paso de asignación de SMs**
6.X	2	2
7.X	2	2
8.X	4	2
9.0+	8	8

De manera similar a MIG, tanto el número máximo de instancias como los tamaños de partición dependen fuertemente no solo de la arquitectura, sino también de la cantidad de SM físicos disponibles en cada GPU.

B. Evaluación de Características de los Mecanismos de Compartición de Recursos para su Uso en el Planificador

Una vez presentadas estas tecnologías, se analizarán sus principales características con el fin de determinar cuál constituye la mejor alternativa para utilizar como base en el desarrollo del planificador.

Los aspectos considerados incluyen la compatibilidad con diferentes versiones de GPU frente a la dependencia de modelos específicos (P); la presencia de aislamiento lógico entre los procesos durante su ejecución en paralelo (AL); y el aislamiento de los recursos físicos de la GPU durante la ejecución de los procesos (AF).

Asimismo, se considerará si, además del aislamiento de cómputo, la tecnología proporciona la separación de otros elementos de hardware distintos de las unidades de cálculo, como la memoria global o la caché (AM). También se analizará el impacto de la tecnología en el rendimiento de los procesos (O) y el nivel mínimo de recursos que permite asignar, lo que permite generar particiones de diferentes tamaños (G).

Finalmente, se verificará si las tecnologías permiten la reasignación dinámica de recursos durante la ejecución de los procesos (RD). Los resultados de esta comparativa se presentan en la Tabla II.

Tabla II: Tabla comparativa de las características principales de las tecnologías de compartición de recursos en las GPUs de Nvidia.

	P	AL	AF	AM	O	G	RD
MPS	✓	∼	×	×	✓	×	✓
MIG	×	✓	✓	✓	×	∼	×
GC	✓	✓	✓	×	✓	✓	✓

En cuanto al análisis, queda claro que MPS, pese a ser una tecnología muy portable y a tratarse de una solución de software que permite la reasignación dinámica de recursos, al no ofrecer garantías de aislamiento, no es viable como alternativa para desarrollar un planificador cuyo objetivo central sea ofrecer garantías temporales y de aislamiento en las GPU [13].

Por otro lado, MIG proporciona mayores garantías de aislamiento para los procesos, ya que no solo aísla los recursos de cómputo, sino también la memoria. Sin embargo, presenta varias limitaciones, como son:

- Baja portabilidad: Solo está disponible en algunas GPUs de propósito general (A100, H100, etc.).
- Baja granularidad: La unidad mínima de asignación son los GPCs, lo que puede resultar en una sobreasignación de recursos al usar la GPU para la inferencia de modelos de redes neuronales.
- Ligero overhead: Comparado con otras alternativas, como se ha demostrado en estudios previos [13], [14].
- Rigidez en la asignación de recursos: requiere liberar por completo las tareas en la GPU antes de generar nuevas particiones.

Estas limitaciones hacen que MIG no sea adecuado para nuestra herramienta, que busca ser dinámica y compatible incluso con equipos de edge computing.

Por último, GC se presenta como la mejor alternativa para el planificador que se busca desarrollar, ya que es una solución de software compatible con una amplia variedad de GPUs, tanto de propósito general como de entorno edge. Permite una alta granularidad, incluso la generación de particiones de dos SMs, y facilita la reasignación dinámica de recursos, lo cual resulta de vital importancia en sistemas de tiempo real.

Por el contrario, no ofrece aislamiento de memoria, lo que puede afectar negativamente el aislamiento entre procesos, especialmente según la topología de los modelos que se ejecuten. A pesar de ello, se ha optado por GC como tecnología base para el planificador.

III. Diseño e Implementación del Sistema

Esta primera sección describirá las características del planificador, así como el proceso de uso de la herramienta desde el punto de vista del usuario.

A. Funcionamiento Basado en Green Contexts

Como se ha descrito en la sección anterior, GC es una tecnología que permite la instancia de contextos de CUDA que limitan la ejecución de un proceso a unos SMs concretos de la GPU. Por lo tanto, antes de poder emplear el planificador, es necesario conocer la arquitectura del chip, así como el número máximo de SMs con los que cuenta la GPU que se pretende emplear, ya que estos dos parámetros determinarán el número máximo de particiones que se pueden instanciar.

Por otro lado, en la versión actual de GC, la tecnología permite dividir la GPU en particiones de igual tamaño, asignando de forma estricta un conjunto fijo de SMs físicos a cada partición. Esto puede provocar que, en determinados casos, a pesar de existir SMs disponibles, no puedan ser utilizados, ya que no pertenecen a la partición asignada al proceso.

En la Figura 1 se describen las configuraciones de GC que pueden instanciarse en una Jetson Orin AGX, la plataforma sobre la que se realizarán los experimentos de este artículo, así como los identificadores de los SMs asignados a cada partición. Se trata de una GPU con un máximo de 16 SMs y una arquitectura de tipo 8.x.

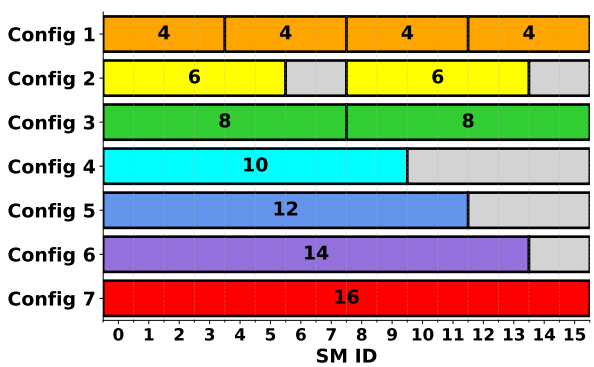


Fig. 1: Posibles configuraciones de instancias de GC para la GPU NVIDIA Jetson Orin AGX. Cada color representa un posible tamaño de partición de GC, basado en el número de SM y en cuáles de ellos se utilizan, dependiendo del tamaño de partición asignado.

B. Descripción de los Modelos de Redes Neuronales

En esta sección se describirán los modelos empleados en los experimentos de este artículo, así como los procesos necesarios para su caracterización previa al despliegue para su uso real. Cabe destacar que este planificador está centrado principalmente en la gestión de procesos de inferencia en la GPU, aunque podría ser compatible con cualquier proceso que se ejecute en este hardware. Dada la motivación del proyecto, se considera esta tarea el objetivo principal del artículo.

B.1 Modelos de redes Neuronales Empleados

Se han utilizado seis arquitecturas diferentes (ConvNeXt-Base, ConvNeXt-Large, ViT_l_32, ViT_b_16, Resnet18 y Mobilenet V2), que abarcan desde redes neuronales convolucionales clásicas y modelos Vision Transformer (ViT) hasta redes neuronales más modernas con topologías cada vez más complejas.

Los pesos, las topologías y los sesgos se obtuvieron de versiones públicas, disponibles en el repositorio Hugging Face [15], todas ellas preentrenadas para la clasificación de imágenes en el conjunto de datos ImageNet-1K [16]. Sin embargo, en este artículo no se considerará ninguna métrica relacionada con la precisión de clasificación de los modelos.

Además, los modelos se convirtieron en un formato binario optimizado, conocido como archivos engine de TensorRT (.engine). Esta optimización, realizada mediante herramientas como trtexec, es una práctica estándar en la computación basada en equipos de edge. El proceso fusiona capas y selecciona los kernels de GPU más eficientes; todas estas optimizaciones están adaptadas al hardware específico [17] [18]. Estos ajustes permiten una optimización general que mejora significativamente la velocidad de inferencia, reduce el consumo de energía y minimiza el uso de memoria, manteniendo una precisión competitiva.

Este enfoque garantiza que los experimentos repliquen escenarios de uso reales en los que los modelos deben operar de manera eficiente en dispositivos con recursos limitados. Pese a ello, el planificador propuesto es completamente compatible con modelos no optimizados, más comúnmente empleados en GPUs de propósito general.

B.2 Caracterización de los Modelos

Al emplear el planificador, es necesario analizar los rendimientos máximos de los modelos en función de los recursos disponibles. Este análisis es de especial relevancia, ya que determina el número mínimo de SMs que el planificador debe asignar a cada proceso para garantizar una frecuencia de inferencia determinada.

Dicho análisis se realiza mediante un procedimiento iterativo en el que cada modelo ejecuta diez mil inferencias para cada tamaño de partición, de forma exclusiva en la GPU. A partir de estos experimentos, se calcula la media de cinco mediciones temporales, con el fin de obtener resultados representativos.

A partir de los tiempos de inferencia en el peor caso, se deriva la frecuencia máxima de inferencia que cada modelo puede alcanzar en función de los recursos asignados. De este modo, al realizar una solicitud al planificador, es posible determinar, en función del modelo y de la frecuencia de inferencia objetivo, los recursos que deben asignarse.

C. Interfaz de Comandos y Funcionalidades

El sistema implementado proporciona una interfaz de control dinámica basada en un hilo de escucha de entrada (input listener) que permite la interacción en tiempo real con el planificador. Esta interfaz es fundamental para la experimentación y el ajuste de parámetros de rendimiento sin necesidad de reiniciar el proceso de ejecución.

La interacción con el sistema se realiza a través de la consola estándar, donde el usuario puede ejecutar comandos para gestionar tareas de inferencia, configurar políticas de garantía y administrar el ciclo de vida del experimento. Los comandos principales se resumen en la Tabla III.

D. Métricas de evaluación

En esta sección analizaremos los parámetros que ofrece el planificador para evaluar su rendimiento:

- **Tabla resumen:** Se trata de una tabla que describe las principales características de los procesos ejecutados al utilizar el planificador. Está compuesta por varias columnas: *Tarea*, que indica el orden de llegada de las peticiones; *Modelo*, en el que se especifica la red CNN ejecutada; y *SM asignados*, que muestra el número de SMs asignados a cada proceso, determinado por el planificador en función del modelo y de la frecuencia de inferencia solicitada.
  Asimismo, se incluyen las inferencias por segundo solicitadas (IPS) por el usuario al lanzar la petición, en la columna de *IPS Solicitado*, junto con los *IPS reales* obtenidos durante la ejecución. También se muestra el porcentaje de garantía (*% Garantía*), que actúa como un margen de seguridad configurable por el usuario. Este parámetro se considera en la asignación de SMs, lo que provoca que el planificador reserve recursos como si la frecuencia solicitada fuera mayor. Esto resulta especialmente relevante en situaciones donde la frecuencia está próxima a un umbral de cambio en el tamaño de partición, permitiendo dotar a los procesos de garantías adicionales.
  Por último, se presenta el número total de inferencias realizadas (*Inferencias realizadas*) por cada petición, calculado a partir de los IPS solicitados y del tiempo de ejecución asignado. Finalmente, se incluye el porcentaje de inferencias exitosas (*% Exitosas*), que representa la proporción de ejecuciones completadas dentro del tiempo establecido, es decir, aquellas que cumplen su deadline, lo que permite analizar las garantías proporcionadas por GC como tecnología

Tabla III: Lista de comandos aceptados por el planificador, junto con los parámetros admitidos y una breve descripción de su funcionamiento.

Comando	Parámetros	Descripción
`task`	`<sms><model_idx> <inferences><freq>`	Lanza una tarea específica definiendo manualmente el número de SMs, el índice del modelo, el número total de inferencias y la frecuencia objetivo (Hz).
`freq`	`<freq><model_idx> <duration>`	Lanza una tarea basada en la frecuencia. El sistema calcula automáticamente los SMs necesarios a partir de una tabla de caracterización previa y del porcentaje de garantía aplicado.
`pct`	`<valor>`	Actualiza dinámicamente el porcentaje de garantía. Este valor actúa como un margen de seguridad sobre los recursos asignados para garantizar el cumplimiento de la frecuencia.
`pol`	`<valor>`	Define la política de planificación mediante un parámetro continuo. Un valor de 0 corresponde a un comportamiento FIFO estricto, mientras que un valor de 1 permite adelantar tareas en la cola, implementando una FIFO con adelantamientos controlados.
`change`	—	Ejecuta un cambio de experimento en caliente. Sincroniza las tareas activas, descarga los motores de TensorRT actuales y los recarga para limpiar el estado de la GPU.
`exit`	—	Finalización controlada. El sistema deja de aceptar nuevas tareas, pero permite que las que están en cola y en ejecución finalicen antes de cerrar.
`stop`	—	Parada de emergencia. Finaliza todos los hilos y libera recursos de forma inmediata, forzando la salida del programa.

de aislamiento.

- **Cronograma:** Esta gráfica muestra la evolución temporal de la ejecución de los distintos procesos en la GPU, así como la ocupación real de los SMs durante su ejecución.
- **Gráfica de utilización:** Esta gráfica muestra el porcentaje de utilización de los SMs a lo largo de una prueba, es decir, la fracción del tiempo en que los recursos están siendo utilizados por los procesos respecto al tiempo total del experimento. Adicionalmente, se incluye el porcentaje de utilización global de la GPU.

IV. Resultados experimentales

Esta sección se centra en mostrar un caso de uso real, desde conocer la plataforma sobre la que se desplegará el planificador hasta generar la caracterización de los modelos y mostrar ejemplos de uso real con el análisis de los resultados obtenidos.

A. *Configuración Experimental*

La plataforma utilizada para la realización de los experimentos es la NVIDIA Jetson AGX Orin Developer Kit, empleando las versiones de software por defecto incluidas en JetPack 6.2 [19].

En el entorno proporcionado por JetPack, las bibliotecas necesarias son mínimas. En particular, se requiere soporte para CUDA, así como el uso de PyTorch como biblioteca principal para la ejecución de inferencias en redes neuronales. CUDA proporciona las instrucciones propias de GC, que permiten generar y asignar las particiones en la GPU, mientras que PyTorch se emplea para la inferencia de los modelos.

B. *Caracterización de los Modelos en Jetson Orin AGX*

Como se ha descrito anteriormente, en esta sección se presentan los resultados de la caracterización de los seis modelos en la Jetson Orin AGX. Para ello, se realizaron pruebas iterativas considerando todos los tamaños de partición disponibles, que coinciden con los descritos en la Figura 1.

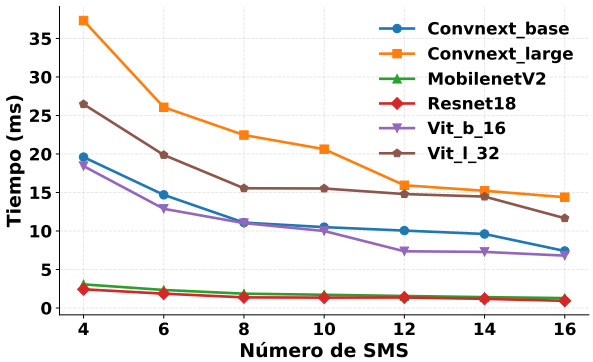


Fig. 2: Tiempo de inferencia en el peor caso para los modelos de red neuronal analizados, en función del número de SMs asignados a cada partición de GC.

Los resultados se muestran en la Figura 2, que ilustra la evolución del tiempo de inferencia de los seis modelos a medida que se asigna el número de SMs a cada partición, y se ejecutan exclusivamente en la GPU.

A la vista de los resultados obtenidos, se observan distintos comportamientos según la topología.

Por un lado, los modelos más pequeños, como la ResNet18 o la MobileNet, sí presentan una mejora muy drástica en el tiempo entre la partición de 4 SMs y la de 8 SMs, con mejoras en el tiempo del 43,19 % y del 39,36 %, respectivamente. Por el contrario, a medida que se siguen aumentando los SMs asignados, esta mejora se reduce drásticamente, ofreciendo un incremento del 32,53 % y del 30,85 % al duplicar los SMs, pasando de 8 a 16. Esto se debe a que estos modelos, al ser tan pequeños, no necesitan tantos recursos de cómputo para ejecutarse, por lo que, al alcanzar ciertos tamaños de partición, pueden llegar a infrautilizar los recursos de cómputo asignados.

Por el contrario, el modelo ConvNeXt-Large, que es el modelo computacionalmente más exigente, sí presenta una mejora en el rendimiento en todos los incrementos de SMs asignados, desde la partición de 4 hasta la de 12 SMs, mostrando una mejora del 57,33 %. Sin embargo, esta mejora se estabiliza li-

geramente en los últimos incrementos de asignación de recursos, limitándose únicamente al 9,72 %, probablemente porque la partición de 12 SMs representa el tamaño mínimo para que el modelo pueda inferir sin estar drásticamente limitado por los recursos de cómputo.

C. Experimentos de planificación

En esta sección, una vez caracterizados los modelos, se presenta un caso de uso real del planificador en la plataforma Jetson Orin AGX. Para ello, se genera un escenario de peticiones aleatorio en el que el planificador gestiona solicitudes de distintos modelos, determinando automáticamente las frecuencias, la duración de los procesos y el orden de llegada de las tareas.

El planificador dispone de dos políticas de gestión de tareas. Por un lado, se encuentra la política FIFO estricta, en la que los procesos se ejecutan en el mismo orden en que llegan al planificador. Esta política resulta adecuada en escenarios donde el cumplimiento del deadline es determinante, aunque puede provocar bloqueos en tareas posteriores por la falta de recursos disponibles para su ejecución.

Por otro lado, se define una política FIFO con adelantamientos. Esta segunda política permite que las tareas que han llegado más tarde se ejecuten antes que otras si hay recursos disponibles. De este modo, se favorece una mayor utilización de la GPU, lo que mejora la eficiencia global del sistema.

Para esta primera prueba, se lanzará una secuencia de peticiones generada aleatoriamente y se mostrarán los resultados obtenidos mediante las gráficas descritas en la Sección III-D. Este experimento se realizará con un porcentaje de garantía del 0 % y aplicando la política FIFO estricta en el planificador. Los resultados se presentan en la Figura 3.

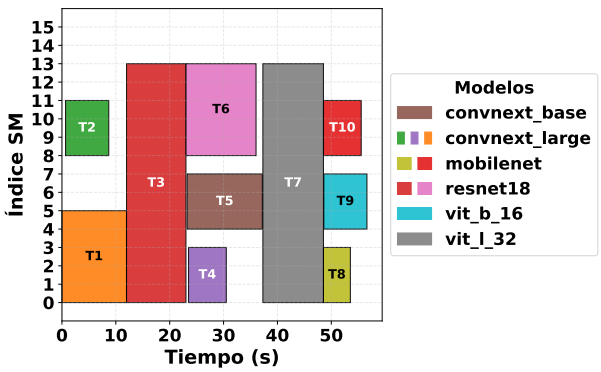
Tras obtener los resultados, se pueden analizar ciertos puntos clave. Por un lado, en la Figura 3.b se puede comprobar cómo el planificador ha respetado la política FIFO estricta. Adicionalmente, se puede apreciar que las tareas 3 y 7 emplean 14 SMs en total, pero, dada la arquitectura de la GPU, estos recursos quedan infrautilizados, ya que no se les puede asignar ningún proceso en paralelo.

Respecto a la utilización general de la GPU, descrita en la Figura 3.c, se observa el comportamiento ya descrito en el cronograma. Debido a que el planificador asigna las particiones empezando por los SMs con índices más bajos, los SMs con índices más altos se ven muy infrautilizados en comparación. Este efecto se debe en parte a la rigidez de GC al generar las particiones, pero no constituye un problema en sí.

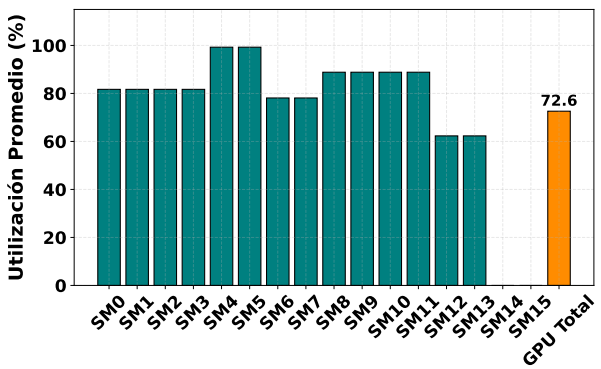
Por último, al analizar la tabla de métricas (Figura 3.a), se aprecia que 9 de los 10 procesos mantienen un porcentaje de inferencias exitosas superior al 93 %, lo cual es un resultado muy positivo, considerando que en algunas ocasiones se ejecutan hasta 3 procesos en paralelo. Por otro lado, el proceso 7 presenta un porcentaje de éxito del 0 %. Pese a ello,

Tarea	Modelo	SM asignados	IPS Solicitado	IPS Real	% Garantía	Inferencias realizadas	% Exitosas
1	conv._large	6	29.4	29.5	0	353	99.2
2	conv._large	4	17.8	17.8	0	143	100.0
3	resnet18	14	828.0	825.7	0	9108	99.8
4	conv._large	4	24.3	24.3	0	170	100.0
5	conv._base	4	12.8	12.9	0	180	100.0
6	resnet18	6	470.4	470.2	0	6115	93.0
7	vit_l_32	14	68.5	67.0	0	753	0.0
8	mobilenet	4	105.4	105.5	0	527	100.0
9	vit_b_16	4	32.5	32.5	0	260	100.0
10	mobilenet	4	127.1	127.2	0	890	100.0

(a) Tabla de métricas.



(b) Cronograma.



(c) Gráfica de utilización.

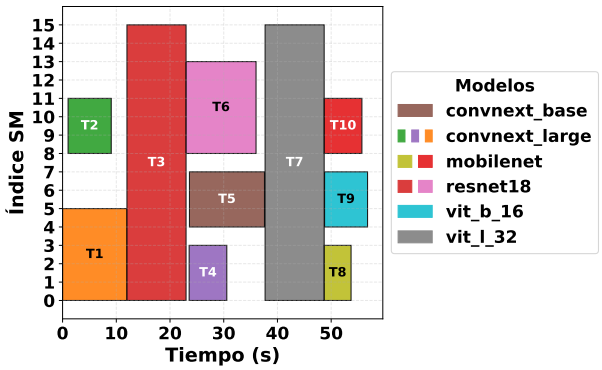
Fig. 3: Tabla de métricas, Cronograma y Gráfica de utilización, resultantes de la utilización del planificador, con política de planificación FIFO estricta y porcentaje de garantía del tiempo de inferencia del 0 %.

la frecuencia de inferencia real obtenida es muy cercana a la solicitada, con unas 67 IPS frente a las 68,5 IPS esperadas. Esto nos lleva a pensar que esta pérdida en los detalles de las inferencias se debe a que la frecuencia de inferencia está muy cerca de la requerida para una asignación de recursos mayor. De ahí surge la idea de repetir el experimento aplicando un porcentaje de garantía del 5 % para comprobar si realmente esto mejora el rendimiento general del sistema.

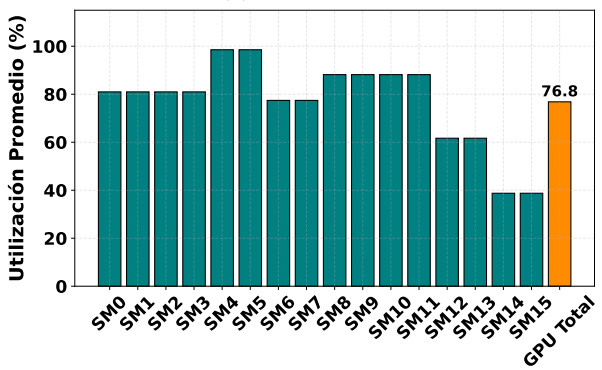
Tras repetir el experimento aplicando el factor de garantía del 5 %, se pueden analizar los efectos que esto ha tenido. Por lo que respecta al cronograma (Figura 4.b), no se aprecian muchos cambios, salvo que las tareas 3 y 7 han pasado de tener 14 a 16 SMs asignados. Cabe destacar que esto se debe a la naturaleza de este experimento, ya que en otros casos el cambio en el factor de garantía puede generar efectos mucho mayores en el cronograma.

Tarea	Modelo	SM asignados	IPS Solicitado	IPS Real	% Garantía	Inferencias realizadas	% Exitosas
1	conv._large	6	29.4	29.5	5	353	99.4
2	conv._large	4	17.8	17.8	5	143	100.0
3	resnet18	16	828.0	827.9	5	9108	99.9
4	conv._large	4	24.3	24.3	5	170	99.4
5	conv._base	4	12.8	12.9	5	180	100.0
6	resnet18	6	470.4	469.8	5	6115	92.6
7	vit_l_32	16	68.5	68.5	5	753	99.5
8	mobilenet	4	105.4	105.5	5	527	100.0
9	vit_b_16	4	32.5	32.5	5	260	100.0
10	mobilenet	4	127.1	126.9	5	890	100.0

(a) Tabla de métricas.



(b) Cronograma.

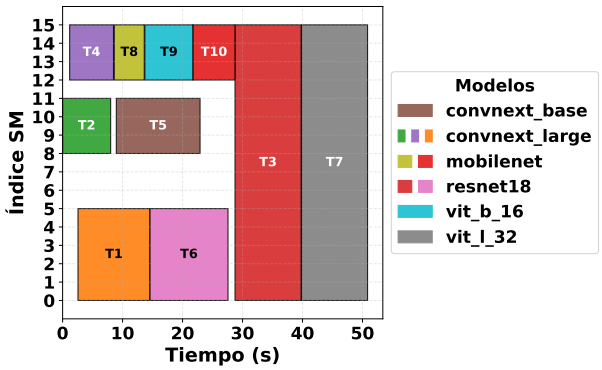


(c) Gráfica de utilización.

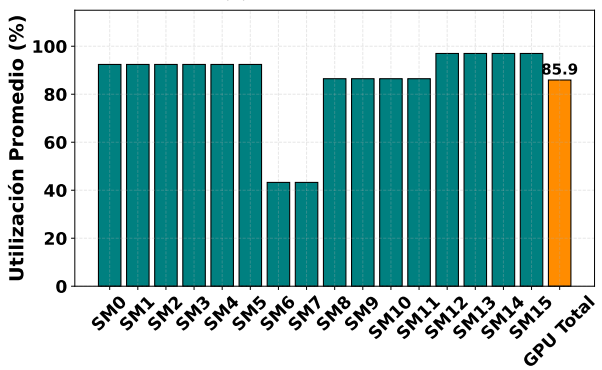
Fig. 4: Tabla de métricas, Cronograma y Gráfica de utilización, resultantes de la utilización del planificador, con política de planificación FIFO estricta y porcentaje de garantía del tiempo de inferencia del 5 %.

Tarea	Modelo	SM asignados	IPS Solicitado	IPS Real	% Garantía	Inferencias realizadas	% Exitosas
1	conv._large	6	29.4	29.4	5	353	97.5
2	conv._large	4	17.8	17.9	5	143	98.6
3	resnet18	16	828.0	827.8	5	9108	99.5
4	conv._large	4	24.3	24.3	5	170	99.4
5	conv._base	4	12.8	12.9	5	180	100.0
6	resnet18	6	470.4	470.2	5	6115	96.3
7	vit_l_32	16	68.5	68.5	5	753	99.9
8	mobilenet	4	105.4	105.5	5	527	99.4
9	vit_b_16	4	32.5	32.4	5	260	100.0
10	mobilenet	4	127.1	127.1	5	890	100.0

(a) Tabla de métricas.



(b) Cronograma.



(c) Gráfica de utilización.

Fig. 5: Tabla de métricas, Cronograma y Gráfica de utilización, resultantes de la utilización del planificador, con política de planificación FIFO con adelantamientos y porcentaje de garantía del tiempo de inferencia del 5 %.

En cuanto a la gráfica de utilización (Figura 4.c), se observa un aumento en la utilización de los SMs 14 y 15 y, por consiguiente, un incremento en la utilización total de la GPU.

Por último, el factor más importante respecto al experimento se puede apreciar en el aumento del porcentaje de garantía, que ha logrado solventar el problema relativo al bajo porcentaje de éxito de la tarea 7, alcanzando un porcentaje de inferencias completadas exitosamente del 99,5 %, sin afectar de manera significativa a ningún otro proceso.

Con los resultados obtenidos, se puede apreciar en la Figura 5.b cómo el cambio de política ha modificado drásticamente la planificación de las tareas, provocando que todas las tareas que requieren particiones más pequeñas se adelanten a las tareas 3 y 7, que requieren la GPU de manera exclusiva. Pese a ello, sí se aprecia una mejora significativa en el tiempo total de ejecución del experimento, propia de que, mediante esta política, no se está produciendo bloqueo entre las tareas, pasando de un tiempo de ejecución total de 56,6 segundos a 50,8 segundos.

Respecto a la utilización, como se muestra en la Figura 5.c, como consecuencia del cambio en el orden de las tareas, también ha habido un cambio en la utilización de los SMs. En general, la utilización ha subido, alcanzando un 85 %, algo esperable ya que la carga de trabajo se ha mantenido constante, pero el tiempo total de ejecución ha disminuido.

Por último, la tabla de métricas (Figura 5.a) se mantiene constante respecto al experimento anterior, algo esperable dado que los procesos ejecutados no han cambiado. Pese a ello, parece haber ligeros cambios en los porcentajes de inferencias exitosas, ya que, como se ha descrito anteriormente, existe un cambio drástico en las tareas que se ejecutan simultáneamen-

te con el cambio de política de asignación, pasando, por ejemplo, en la tarea 6 de un 92,6 % de inferencias exitosas a un 96,3 %, probablemente debido a la naturaleza de las redes que se han ejecutado en paralelo, pudiendo haber pasado de ser tareas que ejercen mayor conflicto en memoria a ser modelos menos exigentes en ese aspecto.

V. Trabajos Relacionados

La planificación de tareas de inferencia dentro de las GPU es un campo ampliamente explorado. Son numerosos los proyectos que, empleando diversas alternativas de particionado, han presentado distintas soluciones.

Algunas de las más relevantes son MIGperf [14] y Paris & Elsa [20], que han explorado el uso de algoritmos de orquestación basados en tecnologías de particionado, como Multi-Instance GPU (MIG). Estas investigaciones introducen un concepto clave: la capacidad de adaptar dinámicamente el tamaño de las particiones en función de la demanda de la carga de trabajo, lo cual resulta fundamental para optimizar tanto el rendimiento como la eficiencia energética del sistema. Sin embargo, estas tecnologías están centradas en sistemas con múltiples GPU, principalmente en entornos de servidores de alto rendimiento.

Más cercanos a lo presentado en este artículo, existen enfoques orientados a la paralelización de procesos dentro de una única GPU, como MIGraptor [21], que propone un sistema de ejecución de GPU en entornos cloud. Este sistema permite que múltiples usuarios ejecuten tareas simultáneamente en la misma GPU sin interferir entre sí, aprovechando las capacidades de aislamiento de MIG.

No obstante, todas estas alternativas no superan las limitaciones inherentes a la tecnología de base. En particular, todas las soluciones desarrolladas con MIG no pueden aplicarse a sistemas de computación en edge, lo que limita su uso en estos entornos.

VI. Conclusiones

Como se ha demostrado a lo largo de esta publicación, actualmente se dispone de una herramienta completamente funcional y versátil para la planificación de tareas de inferencia en GPU, que ofrece garantías relevantes en términos de aislamiento y tiempos de ejecución.

Por tanto, se puede concluir que, a pesar de tratarse de una herramienta en desarrollo y en constante evolución, y de depender en cierta medida de las mejoras proporcionadas por NVIDIA en GC, el planificador presenta un elevado potencial entre las soluciones de particionado de GPUs, tanto en entornos de propósito general como en escenarios de edge computing.

VII. Trabajos Futuros

Aunque a lo largo del artículo se ha demostrado que ha sido posible desarrollar una herramienta plenamente operativa, aún existen ciertos aspectos y líneas de desarrollo que permitirían mejorar la solución presentada.

El primer punto consiste en dotar al planificador de un mecanismo de particionado de memoria. Como se ha expuesto anteriormente, GC, a diferencia de otras tecnologías como MIG, no cuenta con aislamiento de memoria entre procesos, lo cual puede afectar negativamente a las garantías y a la robustez del sistema. Por tanto, añadir un mecanismo de particionado de memoria mejoraría significativamente las garantías de tiempo de ejecución que se pueden ofrecer a los procesos.

Por otro lado, otra línea de trabajo consiste en incorporar nuevas políticas de planificación. Por ejemplo, políticas que tengan en cuenta la naturaleza de los modelos y permitan crear agrupaciones más eficientes en términos de necesidades de cómputo y de memoria, mejorando así no solo la utilización global de los recursos, sino también el aislamiento entre procesos.

Por último, se plantea la posibilidad de incorporar un mecanismo de expulsión de tareas basado en prioridades. En sistemas de tiempo real, la expulsión de tareas en ejecución es un mecanismo de gran relevancia y utilidad demostrada [22]. Por ello, se propone el desarrollo de un sistema mediante el cual una petición de mayor prioridad pueda expulsar a otra de su partición de GC para ejecutarse. Si bien muchas GPUs de NVIDIA no permiten la expulsión de tareas en ejecución, sí se contempla la posibilidad de aplicar este mecanismo entre las inferencias de una misma petición.

Agradecimientos

From the Chips Joint Undertaking project EdgeAITrust no. 101139892 and the Spanish Agencia Estatal de Investigación PCI2024-155056-2 grant. Funding for open access charge: CRUE-Universitat Politècnica de València

El trabajo de Juan José Martín Osuna fue apoyado por parte de la “GVA” vía “Subvenciones para la contratación de personal investigador predoctoral” en la Beca CIACIF/2023/260.

Referencias

[1] Chaitanya Shinde et al., *The Autonomous. (2023). Safe Automated Driving: Requirements and Architectures*, 03 2024.

[2] Mihir Mody et al., “Understanding vehicle E/E architecture topologies for automated driving: System partitioning and tradeoff parameters,” in *Autonomous Vehicles and Machines 2018, Burlingame, CA, USA, January 28 - February 1, 2018*, Patrick Denny et al., Eds. 2018, pp. 1–5, Society for Imaging Science and Technology.

[3] International Electrotechnical Commission, Geneva, Switzerland, *IEC 61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems*, 2010, Parts 1–7.

[4] ISO, “Road vehicles — functional safety,” Tech. Rep. ISO 26262-1:2018, International Organization for Standardization, 2018, Edition 2, Replaces ISO 26262-1:2011.

[5] Jaume Abella et al., “Wcet analysis methods: Pitfalls and challenges on their trustworthiness,” in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2015, pp. 1–10.

[6] Sebastian Altmeyer et al., “WCET and Mixed-Criticality: What does Confidence in WCET Estima-

tions Depend Upon?," in *15th International Workshop on Worst-Case Execution Time Analysis (WCET 2015)*, Francisco J. Cazorla, Ed., Dagstuhl, Germany, 2015, vol. 47 of *Open Access Series in Informatics (OASIcs)*, pp. 65–74, Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[7] Sparsh Mittal, "A survey of techniques for cache partitioning in multicore processors," *ACM Comput. Surv.*, vol. 50, no. 2, may 2017.

[8] NVIDIA Corporation, "Multi-process service (mps) overview," NVIDIA Documentation, n.d., Accessed Mar. 24, 2025.

[9] NVIDIA Corporation, "Multi-instance gpu (mig) technology," NVIDIA Documentation, n.d., Accessed Mar. 24, 2025.

[10] NVIDIA Corporation, "Cuda driver api documentation, section 6.35: Green contexts," NVIDIA Documentation, 2025, Version 12.9.1, Last updated June 4, 2025. Accessed Jul. 29, 2025.

[11] NVIDIA Corporation, "Cuda programming guide, section 4.6: Green contexts," NVIDIA Documentation, 2025, Version 13.1, Last updated December 1, 2025. Accessed Dec. 4, 2025.

[12] NVIDIA Corporation, "Cuda toolkit documentation," `https://docs.nvidia.com/cuda/archive/12.4.0/`, 2023, Version 12.4.

[13] Juan José Martín et al., "Peformance isolation for inference processes in edge gpu systems," 2026.

[14] Huaizheng Zhang et al., "Migperf: A comprehensive benchmark for deep learning training and inference workloads on multi-instance gpus," *arXiv preprint arXiv:2301.00407*, 2023.

[15] Hugging Face, "Hugging face platform," Online platform, 2025, Retrieved from https://huggingface.co. Accessed Mar. 14, 2025.

[16] Jia Deng et al., "Imagenet: A large-scale hierarchical image database," in *CVPR 2009*. IEEE, 2009, pp. 248–255.

[17] Jacob Sander et al., "On accelerating edge ai: Optimizing resource-constrained environments," *arXiv preprint arXiv:2501.15014*, 2025.

[18] Dominique Heller et al., "Marine objects detection using deep learning on embedded edge devices," in *RSP 2022*. IEEE, 2022, pp. 1–7.

[19] NVIDIA Corporation, "Nvidia jetpack sdk 6.2," `https://docs.nvidia.com/jetson/archives/jetpack-archived/jetpack-62/release-notes/index.html`, 2025.

[20] Younghoon Kim et al., "Paris and elsa: An elastic scheduling algorithm for reconfigurable multi-gpu inference servers," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*. July 2022, pp. 607–612, ACM.

[21] Tian Wang et al., "Improving gpu multi-tenancy through dynamic multi-instance gpu reconfiguration," *arXiv preprint arXiv:2407.13126*, 2024.

[22] Nicolo Capodieci et al., "Deadline-based scheduling for gpu with preemption support," in *Proceedings of the 2018 IEEE Real-Time Systems Symposium (RTSS)*. December 2018, pp. 119–130, IEEE.

Toolchain Automatizado para la Integración de Kernels CUTLASS en Motores TensorRT

Alejandro Ripoll Acosta, José Flich Cardo, Carles Hernández Luz[1]

Resumen— **Este trabajo presenta el diseño, implementación y evaluación de un *toolchain* automatizado para la integración de kernels CUTLASS personalizados en motores de inferencia NVIDIA TensorRT. A diferencia de las implementaciones nativas —comúnmente denominadas "caja negra"—, el sistema propuesto otorga control granular sobre el código ejecutado en la GPU, sustituyendo las convoluciones estándar por kernels optimizados generados con CUTLASS. El núcleo del sistema automatiza un *pipeline* de cuatro etapas: perfilado de kernels convolucionales por capa, modificación topológica del grafo ONNX, generación de código C++/CUDA especializado, e integración dinámica como plugin `IPluginV2Ext` en TensorRT.**

Los resultados experimentales han sido obtenidos sobre un dispositivo NVIDIA Jetson AGX Orin con el modelo YOLOv8n a 256×256 px en FP16, mostrando una latencia de 0,760 ms frente a los 0,711 ms del motor nativo (∼7 % de penalización). Sin embargo, el tiempo neto de cómputo convolucional es incluso inferior en el motor CUTLASS, validando la calidad de los kernels seleccionados. La penalización global se atribuye al *overhead* de las operaciones de transposición de layout (NHWC ↔ NCHW). Estos resultados demuestran que el enfoque propuesto constituye una alternativa viable y competitiva al compilador propietario de NVIDIA. Al aportar trazabilidad absoluta y control granular bajo un paradigma de caja blanca, este sistema sienta las bases necesarias para futuras optimizaciones y para la inyección de técnicas de tolerancia a fallos a nivel de kernel.

Palabras clave— **TensorRT, CUTLASS, kernel optimization, neural network inference, edge computing, Jetson Orin, YOLOv8, ONNX, GEMM, operator fusion, fault tolerance, white-box inference**

I. Introduccion

La inferencia de redes neuronales convolucionales (CNN) en dispositivos de computación embebida de alto rendimiento, como las plataformas NVIDIA Jetson [1], constituye uno de los retos centrales de los sistemas de visión artificial en tiempo real. El despliegue de modelos SSD (Single Shot MultiBox Detector) de detección de objetos como los de la arquitectura YOLO [2] (You Only Look Once) sobre hardware con recursos limitados exige maximizar la eficiencia de cada operación, siendo la convolución el núcleo computacional dominante en la mayoría de estas arquitecturas.

NVIDIA TensorRT [3] es el compilador y motor de inferencia de referencia en el ecosistema CUDA. A partir de un grafo ONNX [4] (Open Neural Network Exchange), TensorRT aplica optimizaciones como fusión de operaciones, cuantización de precisión (FP32, FP16, INT8) y selección automática de kernels mediante su propio algoritmo de autoajuste. Sin embargo, este proceso es relativamente opaco, ofreciendo escaso control al desarrollador sobre que implementación concreta se ejecuta para cada capa convolucional (caja negra).

Por otro lado, CUTLASS [5] (CUDA Templates for Linear Algebra Subroutines and Solvers) es la biblioteca de referencia de NVIDIA para implementar operaciones matriciales y convoluciones de alta eficiencia directamente sobre el hardware tensorial (Tensor Cores) [6] de las GPUs. CUTLASS expone una jerarquía de plantillas en C++ que permite instanciar kernels parametrizados, así como el soporte integral para precisión mixta, la fusión de operaciones (como sesgos y activaciones) para reducir accesos a memoria, y el manejo eficiente de tensores dispersos y operaciones por lotes.

Este trabajo propone un sistema integral que une ambos enfoques mediante un *pipeline* automatizado estructurado en cuatro fases principales: (1) perfila y selecciona los kernels de convolución de mayor rendimiento para un hardware específico utilizando el evaluador de CUTLASS; (2) modifica topológicamente el grafo ONNX original para inyectar nodos personalizados, optimizar el *layout* de memoria y fusionar operadores mediante la librería *onnx-graphsurgeon* [7]; (3) genera automáticamente el código C++/CUDA altamente especializado para cada capa; y (4) compila e integra esta lógica en TensorRT a través de un *plugin* dinámico para la construcción del motor de inferencia.

El objetivo principal es evaluar en qué medida la inyección de kernels CUTLASS, ajustados específicamente para cada capa, puede igualar la velocidad de inferencia del mecanismo de autoajuste nativo de TensorRT, así como identificar y comprender las fuentes de sobrecarga (*overhead*) que puedan limitar dicho rendimiento. Esta aproximación aporta el valor añadido de un control absoluto sobre el flujo de ejecución (paradigma de caja blanca). Dicha trazabilidad es un requisito indispensable en entornos críticos para poder, en última instancia, implementar e inyectar técnicas de tolerancia a fallos directamente a nivel de kernel, algo inviable sobre el motor compilado propietario.

El resto del artículo se organiza de la siguiente manera: la Sección II revisa el trabajo relacionado en el campo; la Sección III describe en detalle el diseño e implementación del sistema; la Sección IV expone y analiza los resultados experimentales obtenidos al validar la metodología con un modelo YOLOv8n [8] sobre hardware NVIDIA Jetson Orin AGX [1]; y, finalmente, la Sección V recoge las conclusiones, las principales aportaciones y las posibles líneas de trabajo futuro.

[1]Contacto: aripaco@etsinf.upv.es, jflich@disca.upv.es, carherlu@upv.es.

II. Trabajo Relacionado

La optimización de la inferencia de redes neuronales convolucionales en hardware embebido ha sido abordada desde distintas perspectivas. Los compiladores de ML de propósito general como Apache TVM [9] y OpenAI Triton [10] automatizan la búsqueda de kernels eficientes mediante autotuning, pero su proceso de compilación resulta igualmente opaco y su integración nativa con el ecosistema TensorRT no es directa. NVIDIA mantiene un repositorio oficial de plugins de referencia para TensorRT [11] que cubre operadores ausentes del grafo estándar mediante la interfaz `IPluginV2Ext`; sin embargo, estos plugins no sustituyen convoluciones ya soportadas por kernels especializados por capa. La biblioteca CUTLASS [5] ha sido adoptada como backend en trabajos sobre aceleración de transformers [12], donde la fusión de operaciones en el epílogo del kernel reduce drásticamente los accesos a memoria global, pero su uso sistemático como sustituto capa a capa dentro de un motor TensorRT para inferencia convolucional en edge no ha sido documentado en la literatura pública. El presente trabajo cubre ese hueco: proporciona un pipeline automatizado que selecciona, instancia e integra kernels CUTLASS para cada capa convolucional del modelo dentro de TensorRT, con plena trazabilidad del código ejecutado.

III. Diseño del Sistema

El sistema propuesto sigue el pipeline de cuatro etapas ilustrado en la Fig. 1. A continuación se describe cada etapa con detalle técnico.

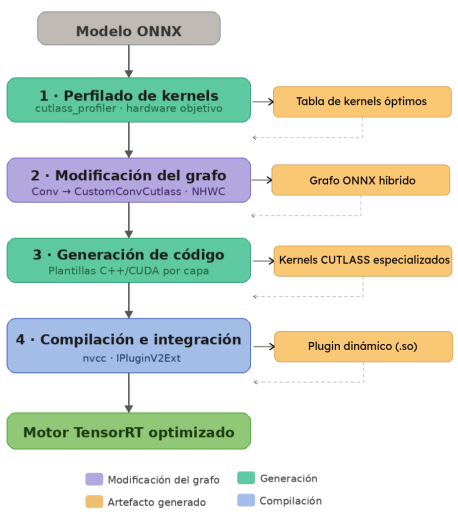


Fig. 1: Pipeline de construcción del motor TensorRT a partir del modelo ONNX original con kernels CUTLASS personalizados.

A. Etapa 1: Exploración y perfilado de kernels CUTLASS

El objetivo de esta etapa es determinar empíricamente cuál es la configuración de kernel CUTLASS de mayor rendimiento para cada capa convolucional del modelo. Para ello, el sistema extrae y analiza la geometría y los atributos básicos de cada nodo `Conv` del grafo (dimensiones de los tensores de entrada y salida, tamaño del filtro, *padding*, *stride*, etc.).

Con estos datos, se define un espacio de búsqueda basado en los principales parámetros de optimización de la GPU: el tamaño del bloque computacional (*tile*), el número de etapas (*stages*), el nivel de particionado paralelo (Split-K) y el alineamiento de memoria.

Dado que evaluar empíricamente todo el abanico de combinaciones posibles resultaría ineficiente, el perfilador aplica una heurística adaptativa para podar el espacio de búsqueda en función de la forma y el tamaño de la capa:

- **Capas de baja carga (*tiny layers*):** Correspondientes a mapas de características muy reducidos (ej. 8×8). En estas capas se activa un modo especial que restringe el uso de *tiles* grandes y fuerza la exploración de técnicas como Split-K para compensar la falta de paralelismo espacial.
- **Capas medias y masivas:** La exploración se filtra según el volumen total de operaciones (MACs) de la capa. En las capas más pesadas, se descartan de antemano las configuraciones de bloques pequeños, ya que la GPU no lograría amortizar el *overhead* de lanzarlos, restringiendo la búsqueda únicamente a las plantillas de alta eficiencia.

Finalmente, las combinaciones candidatas que superan este filtrado se ejecutan y miden iterativamente en el hardware destino mediante el evaluador (*profiler*) nativo de CUTLASS. La configuración que registra el menor tiempo de ejecución (*Runtime*) se selecciona como ganadora. Sus parámetros óptimos se serializan en un archivo CSV, el cual servirá como plano de construcción para guiar la modificación del grafo y la posterior generación de código.

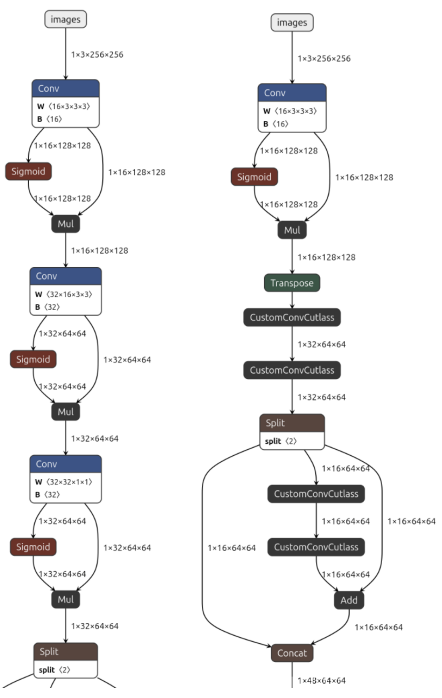
B. Etapa 2: Modificación del grafo ONNX

Una vez encontrados los kernels óptimos, se procede a modificar la estructura del grafo ONNX [4] original. El proceso consiste en reemplazar los nodos estándar de convolución por nodos personalizados, identificados como `CustomConvCutlass`. Esta modificación se realiza con la librería `onnx-graphsurgeon` [7], que permite operar a nivel de nodo, facilitando la creación o eliminación de estos y la redirección del flujo del grafo, entre otras operaciones.

1. **Sustitución de nodos Conv:** Cada nodo `Conv` del grafo (identificado por su índice de capa según el mapeo del archivo CSV) se reemplaza por un nodo `CustomConvCutlass` con los atributos necesarios: nombre del kernel seleccionado, pesos, sesgos (*bias*) y todos los parámetros geométricos propios de la operación de convolución (canales de entrada y salida, tamaño del kernel, *padding*, *stride*, etc.). Además, los pesos originales en formato NCHW se transponen a NHWC y se aplanan en un vector contiguo en FP32 antes de ser inyectados como atributo del nodo *plugin*. Cuando el kernel seleccionado ope-

ra en otra precisión, los pesos se mantienen en FP32 para que TensorRT realice la conversión correspondiente internamente.

2. **Gestión del *layout*:** Para garantizar la alineación en memoria de los tensores —requisito indispensable para maximizar el ancho de banda y mantener la coherencia a la hora de integrar los kernels CUTLASS en el grafo—, se inserta una transpuesta NCHW→NHWC antes del primer nodo *plugin* de cada secuencia continua, y una transpuesta NHWC→NCHW tras el último. Las operaciones elementales (`Add`, `Mul`, `SiLU`, `Concat`, `Split`) se clasifican como compatibles con NHWC, por lo que no requieren transpuestas intermedias. Las operaciones incompatibles (`MaxPool`, `Resize`, `Reshape`, `Flatten`) sí las requieren, lo que provoca que el sistema inyecte automáticamente nuevos nodos de transposición para mantener la coherencia. Además, se realiza un ajuste de ejes: los nodos `Concat` y `Split` con eje 1 (canal en NCHW) se reasignan al eje 3 (canal en NHWC). Por último, para la restauración de las salidas del modelo: tras el proceso de propagación, cualquier tensor de salida del grafo que permanezca en NHWC recibe una transpuesta final NHWC→NCHW, garantizando que la interfaz exterior del modelo respete el formato NCHW que esperan las etapas de postprocesamiento (*downstream*).
3. **Fusión de operadores:** Asimismo, se han aplicado patrones de fusión de operadores (*Conv* → *Sigmoid* → *Mul*, equivalente a la activación SiLU [13]: $x \cdot \sigma(x)$), reduciendo drásticamente el número de accesos a memoria global y la sobrecarga asociada al lanzamiento de múltiples kernels de forma individual.



(a) Grafo Nativo (b) Grafo con Plugin

Fig. 2: Modificación topológica de la red neuronal.

Este re-diseño optimiza integralmente el modelo: sustituye las convoluciones nativas por los kernels alta eficiencia, fusiona operadores para reducir drásticamente los accesos a memoria y gestiona el *layout* para minimizar las transposiciones. Una vez completada esta transformación, el grafo se limpia eliminando los tensores huérfanos y se reordena topológicamente antes de la serialización. El resultado final es un grafo ONNX híbrido completamente equivalente al original, donde las operaciones fusionadas y las convoluciones se ejecutan mayoritariamente en formato NHWC de forma continua y eficiente, minimizando las operaciones que no corresponden a cómputo útil.

C. Etapa 3: Generación automática de código

Una vez generado el grafo modificado, TensorRT no puede interpretar nativamente los nodos `CustomConvCutlass`: el ONNX define *qué* hacer, pero no *cómo* ejecutarlo. El sistema de exportación lee el CSV de perfilado y genera automáticamente el código C++/CUDA necesario. Cada kernel queda especializado en tiempo de compilación a partir de cuatro grupos de parámetros:

- ***Tile* y *stages*:** el *tile* (p. ej., `GemmShape<64,64,64>`) fija el bloque de cómputo procesado por cada grupo de hilos, controlando el compromiso entre reutilización de datos en SMEM y ocupación de la GPU. El número de `stages` solapa la carga de *tiles* con el cómputo en los Tensor Cores; este *trade-off* se evalúa empíricamente durante el perfilado.
- **Split-K y reducción paralela:** en capas profundas con mapas de características pequeños (8×8), el paralelismo espacial es insuficiente para saturar la GPU. El parámetro `split_k_slices` divide la dimensión de acumulación K en fragmentos procesados en paralelo. El modo **Serial** acumula resultados parciales mediante operaciones atómicas; el modo **Parallel** usa un *buffer* temporal para evitar contención. Dado que SiLU [13] no es distributiva, la activación se aplica únicamente sobre la partición final (`k_partition_ == k_partition_count_ - 1`).
- **Swizzle:** reorganiza el orden de lanzamiento de los bloques en el *grid* para mejorar la localidad en caché L2. Su valor óptimo se determina automáticamente en función del tamaño del *grid* de cada capa.
- **Alineamiento:** determina cuántos elementos se cargan por transacción de memoria (8, 4, 2 ó 1). Se selecciona automáticamente en función de la divisibilidad de C y K por el ancho del vector de memoria, que depende de la precisión manejada: a mayor tamaño por elemento, menor es el alineamiento óptimo alcanzable. Valores no alineados fuerzan plantillas de menor eficiencia, con impacto especialmente notable en capas con dimensiones de canal reducidas o no potencia de dos.

D. Etapa 4: Integración del plugin y construcción del motor

La librería encapsula el código generado mediante la interfaz `IPluginV2Ext` [11] y cubre tres fases del ciclo de vida de TensorRT:

- **Registro (*Plugin Creator*):** una clase `PluginCreator` se registra automáticamente al cargarse en memoria, notificando al *parser* que la librería provee la implementación de `CustomConvCutlass` y que TensorRT no debe optimizar dichos nodos.
- **Configuración y Deserialización:** durante el *build*, el *plugin* recibe los datos del ONNX (pesos, *strides*, *padding*), valida los formatos de entrada/salida y transforma los pesos al *layout* NHWC requerido por CUTLASS, dejándolos listos en VRAM.
- **Ejecución (*Enqueue*):** la función `enqueue` despacha los kernels CUTLASS al *stream* de CUDA construyendo previamente el objeto `Conv2dProblemSize` con los parámetros exactos de la capa. El despacho se realiza a través de un *wrapper* de plantillas C++ generado automáticamente, que instancia de forma quirúrgica `GemmShape`, `stages`, `alignment` y *swizzle* por capa, haciendo el sistema escalable a otras arquitecturas sin intervención manual.

Finalmente, la librería (`.so`) se compila con `nvcc` y `g++` enlazando contra `nvinfer`, `cudart` y `cublas`. Un script de construcción la carga dinámicamente vía `ctypes`, parsea el grafo ONNX híbrido y serializa el motor de inferencia en el archivo `.engine` definitivo, listo para su despliegue.

IV. Resultados Experimentales

A. Entorno de evaluación

Las pruebas se han ejecutado sobre un dispositivo NVIDIA Jetson AGX Orin [1] con las siguientes especificaciones: GPU Ampere [6] con 1792 núcleos CUDA y 56 Tensor Cores (arquitectura `sm_80`), 32 GB LPDDR5 de memoria unificada a 204,8 GB/s y entorno JetPack 5.x (CUDA 11.4, TensorRT 8.5, cuDNN 8.6 [14]). El modelo evaluado es YOLOv8n [8] con una entrada de 256×256 píxeles en modo inferencia ($batch = 1$, precisión FP16 [15]). La precisión de acumulación en todos los kernels CUTLASS [5] es FP32, al igual que en el motor nativo de TensorRT [3]. Todos los experimentos se ejecutaron en modo MAXN (`nvpmodel`) con relojes fijados al máximo mediante `jetson_clocks`, eliminando cualquier *throttling* dinámico que pudiera enmascarar diferencias arquitectónicas entre motores.

La herramienta de medición empleada es `trtexec` de NVIDIA. A continuación, se evalúan dos escenarios para la comparativa: el motor nativo (generado por el *autotuner* de TensorRT) y el motor exportado con el *toolchain* desarrollado.

B. Análisis de rendimiento global

Como se observa en la comparativa de métricas (Figura 3) y en la distribución de latencias por capa (Figura 4), ambas implementaciones ofrecen un rendimiento sobresaliente, adecuado para aplicaciones de visión en tiempo real.

En lo que respecta al tiempo de cómputo puro, el motor nativo de TensorRT logra una media de 0,711 ms, mientras que el motor con el *plugin* CUTLASS registra 0,760 ms. Esta diferencia de apenas 49 μs (un ∼7 % de incremento) demuestra que el código C++/CUDA generado (el plugin CUTLASS) es capaz de competir con el compilador propietario de NVIDIA. El gráfico de violín (Figura 3, derecha) confirma la estabilidad de ambas soluciones ya que la varianza es mínima en los dos motores.

El histograma de latencia por capa (Figura 4) revela además una distribución bimodal en ambas implementaciones, con un grupo de capas de ejecución rápida (concentradas en torno a 0,006–0,008 ms) y un segundo grupo de capas más costosas (alrededor de 0,013–0,016 ms). En ambos casos la cola derecha es larga pero poco poblada, lo que indica que las capas atípicas no llegan a dominar el tiempo total de inferencia. No obstante, destaca que el motor CUTLASS concentra una mayor proporción de capas en los valores bajos de latencia, resultando en una distribución más leptocúrtica en ese rango, lo que sugiere una mayor concentración en el número de capas más rápidas.

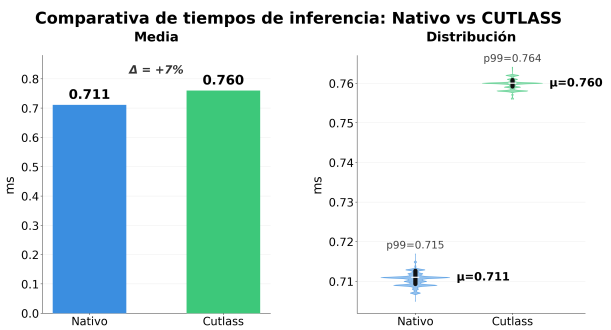


Fig. 3: Comparativa de tiempos de inferencia (media y distribución)

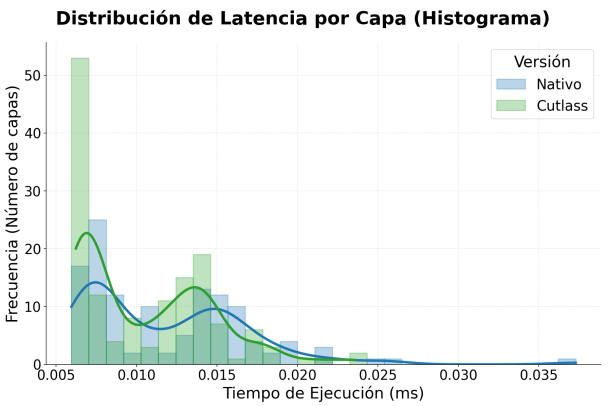


Fig. 4: Distribución de latencia por capa (histograma con estimación de densidad): Nativo vs. CUTLASS.

C. Desglose por categorías y convoluciones

Para comprender el origen de la ligera diferencia en la latencia global, es necesario analizar el tiempo de ejecución desglosado por tipo de operación (Figura 5 y 6).

El análisis revela que el tiempo neto invertido en operaciones de convolución es comparable en ambos motores: la suma de las convoluciones *fused* (Conv+SiLU [13]) y no-*fused* asciende a 0,742 + 0,212 = 0,954 ms en el motor nativo, frente a 0,902 + 0,017 = 0,919 ms en el motor con el *plugin* CUTLASS, lo que valida la calidad del explorador y de las plantillas seleccionadas por el perfilador de CUTLASS.

En cuanto a la composición porcentual, las convoluciones dominan en ambos casos, representando el 64,8 % del tiempo total en el motor nativo frente al 60,0 % en el motor CUTLASS. La penalización del motor híbrido radica, por tanto, en la sobrecarga (*overhead*) de las operaciones auxiliares (*Reformat* y *Transpose*), que suponen un 26,3 % del tiempo frente al 21,0 % del motor nativo. Como se anticipó en la metodología, la inserción de nodos `CustomConvCutlass` añade un ligero aumento en el número de nodos de transposición originalmente no previstos en el grafo, con el objetivo de mantener la coherencia para compatibilizar los *layouts* de memoria entre las capas nativas y los *plugins* NHWC. Si bien el análisis de la Figura 7 revela que el tiempo dedicado a las operaciones de *Reformat* es menor en el modelo híbrido, los nodos *Transpose* inyectados en ambos sentidos introducen cierta latencia. Esto se traduce en penalización de 0,1 ms en comparación con el motor nativo de TensorRT.

Un hallazgo analítico relevante radica en las estrategias de fusión de operadores. Mientras que el compilador de TensorRT opta heurísticamente por no fusionar ciertas convoluciones, la estrategia implementada en nuestro exportador fuerza la fusión siempre que se detecta el patrón topológico compatible. Este comportamiento explica la drástica reducción temporal de las convoluciones no fusionadas en el escenario CUTLASS (apenas 0,017 ms frente a los 0,212 ms del nativo). Como consecuencia directa, se produce una divergencia en el peso de las operaciones *Elementwise* (operaciones aritméticas elementales independientes, como sumas o multiplicaciones): al no aplicar una fusión tan exhaustiva, el motor nativo arrastra una mayor cantidad de estos nodos aislados.

En cuanto al resto de operaciones de la red (*MaxPool*, *Resize*, *Split* y DFL), representan una fracción minoritaria del tiempo de inferencia. Su impacto temporal y sus proporciones se mantienen prácticamente idénticos en ambos escenarios, debido a que son una parte más estructural del grafo, habiendo sido menos manipulados en ambas técnicas de exportación.

Entrando en mayor detalle dentro de las convoluciones del modelo, los kernels de CUTLASS seleccionados logran igualar o superar al motor nativo en

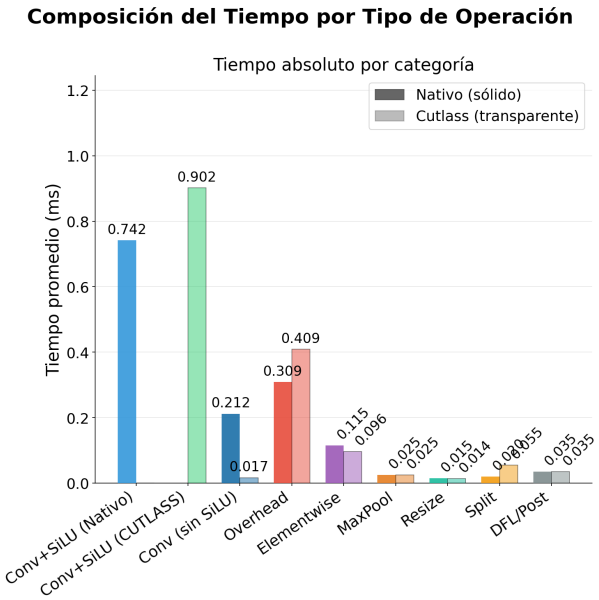


Fig. 5: Desglose del tiempo de ejecución total acumulado por categoría de operación.

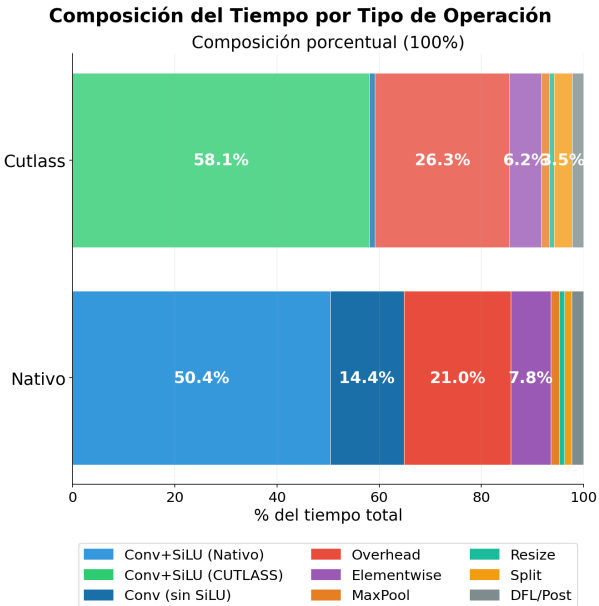


Fig. 6: Desglose relativo (%) del tiempo de ejecución acumulado por categoría de operación.

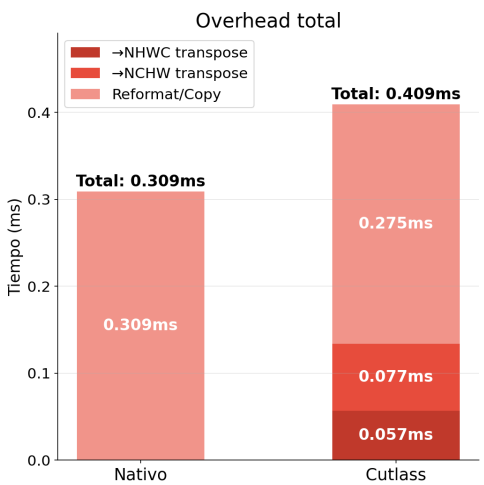


Fig. 7: Desglose del tiempo de Overhead.

la mayoría de capas del *backbone*, donde los tensores son suficientemente grandes para amortizar el coste de lanzamiento del *plugin*. Sin embargo, se observa una desaceleración clara en las últimas convoluciones del modelo, especialmente en las correspondientes al *head* o cabeza del modelo, por ello se identifican tres categorías de capas donde nuestro explorador no es competitivo:

1. Las capas de detección final con $k = 14$ canales de salida: al no ser k múltiplo de 16 —el tamaño del *tile* de tensor core—, el explorador de kernels rebaja automáticamente el alineamiento y selecciona una plantilla de menor eficiencia (`align2` en lugar de `align8`), con un coste por GFLOP hasta 30 veces superior al de una capa normal del *backbone*.
2. Las convoluciones 3×3 con resolución espacial 8×8 y canales asimétricos ($c = 256$, $k = 64$): con tan solo 64 posiciones espaciales, los CTAs de CUTLASS quedan infrautilizados y los *warps* permanecen ociosos, resultando en una intensidad aritmética de apenas 31 FLOP/byte.
3. Las convoluciones 1×1 de baja intensidad aritmética en resoluciones 8×8 y 16×16 presentan un ratio de entre 21 y 28 FLOP/byte, situándolas en régimen *memory-bound*, donde CUTLASS —diseñado para operaciones *compute-bound*— no puede competir con los kernels escalares que TensorRT selecciona de cuDNN [14] para estas formas.

D. Utilización de GPU

La utilización de la GPU se ha medido mediante dos métodos complementarios. `tegrastats` [16] reporta `GR3D_FREQ`, que marca la GPU como activa en cuanto existe cualquier kernel en vuelo: el motor nativo alcanza el 95,9 % y el CUTLASS el 94,8 %, manteniéndose ambos en régimen de ocupación prácticamente continua. Los registros CUPTI de Nsight Systems [17], que miden la densidad real de cómputo en ventanas de 5 ms, arrojan un 43,0 % para el nativo y un 40,8 % para CUTLASS; la diferencia refleja los breves intervalos adicionales introducidos por las transposiciones NHWC↔NCHW del motor híbrido. Ambos valores confirman que el *pipeline* propuesto satura los recursos del dispositivo de forma comparable al compilador propietario.

V. Conclusiones

Este trabajo ha presentado un sistema completo para la integración de kernels CUTLASS [5] personalizados en motores TensorRT [3] para inferencia, evaluado sobre YOLOv8n [8] en Jetson Orin [1]. Las principales conclusiones son:

1. Viabilidad técnica demostrada. El pipeline propuesto (perfilado → selección → generación de código → plugin) es completamente automatizado y produce un motor totalmente funcional, compatible con CUDA Graph y serializable. El diseño modular facilita su aplicación a cualquier CNN.

2. Aportaciones del enfoque. Si bien el motor nativo conserva una ligera ventaja absoluta de rendimiento gracias a su profunda integración y a la ausencia de *overhead* de interoperabilidad, el sistema basado en CUTLASS se consolida como una alternativa viable y altamente competitiva. La leve penalización en la latencia global es el costo asumible por adoptar un diseño deliberadamente más genérico y transparente. Por otro lado, una optimización demasiado rígida o específica podría mermar la capacidad de adaptación de la solución frente a otras arquitecturas de redes neuronales. Por tanto, su principal valor —a diferencia del compilador propietario de NVIDIA— radica en la evolución del paradigma de inferencia hacia un entorno de caja blanca, asegurando así la predictibilidad y una trazabilidad absoluta del código.

A. Trabajo futuro

El *toolchain* desarrollado abre una línea de trabajo especialmente relevante para entornos críticos: la integración de técnicas de tolerancia a fallos directamente en el código de inferencia. Esta posibilidad es exclusiva del enfoque de caja blanca adoptado en este trabajo —un motor TensorRT nativo no expone los puntos de intervención necesarios— y constituye una de las aportaciones más valiosas del *toolchain* desarrollado.

Entre las técnicas a implementar destacan la protección por redundancia a nivel de bit en pesos (*FP/VP Bit-Level Redundancy*) [18], el recorte dinámico de activaciones dentro del epílogo (*NaN/Inf Trapping*) y verificación algebraica mediante ABFT [19]. La meta es combinar estas aproximaciones bajo un modelo de *Selective Layer Hardening*, permitiendo una aplicación estratégica de las protecciones en las capas computacionalmente más críticas y vulnerables.

Agradecimientos

Este trabajo ha sido realizado en el marco del proyecto Chips Joint Undertaking EdgeAITrust no. 101139892, con financiación del programa Horizonte Europa de la Unión Europea, y de la Agencia Estatal de Investigación, proyecto PCI2024-155056-2. La financiación para el acceso abierto ha sido proporcionada por la CRUE-Universitat Politècnica de València.

Referencias

[1] NVIDIA Corporation, "NVIDIA Jetson AGX Orin Technical Reference Manual," 2022.
[2] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.
[3] NVIDIA Corporation, "TensorRT Developer Guide," 2024.
[4] ONNX Community, "ONNX: Open Neural Network Exchange," 2019.
[5] NVIDIA Corporation, "CUTLASS: CUDA Templates for Linear Algebra Subroutines and Solvers," 2023.
[6] NVIDIA Corporation, "NVIDIA Ampere GA102 GPU Architecture Whitepaper," 2021.

[7] NVIDIA Corporation, "onnx-graphsurgeon: A Library for Modifying ONNX Graphs," 2023.
[8] Glenn Jocher, Ayush Chaurasia, and Jing Qiu, "Ultralytics YOLOv8," 2023.
[9] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy, "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018, pp. 578–594.
[10] Philippe Tillet, H. T. Kung, and David Cox, "Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations," in *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)*, 2019, pp. 10–19.
[11] NVIDIA Corporation, "TensorRT Open Source Software – Sample Plugins Repository," 2023.
[12] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré, "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022, vol. 35.
[13] Prajit Ramachandran, Barret Zoph, and Quoc V. Le, "Searching for Activation Functions," *arXiv preprint arXiv:1710.05941*, 2017.
[14] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer, "cuDNN: Efficient Primitives for Deep Learning," 2014.
[15] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu, "Mixed Precision Training," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.
[16] NVIDIA Corporation, "Tegrastats utility," `https://docs.nvidia.com/jetson/archives/r35.1/DeveloperGuide/text/AT/JetsonLinuxToolchain/Tegrastats.html`, 2023.
[17] NVIDIA Corporation, "Nsight systems user guide," `https://docs.nvidia.com/nsight-systems/`, 2024.
[18] Izan Catalán, José Flich, and Carles Hernández, "Exploiting neural networks bit-level redundancy to mitigate the impact of faults at inference," *The Journal of Supercomputing*, vol. 81, pp. 183, 2025.
[19] Kuang-Hua Huang and Jacob A. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations," *IEEE Transactions on Computers*, vol. C-33, no. 6, pp. 518–528, 1984.

Optimización de SpMM iterativo en RISC-V mediante ELLPack y reordenamiento de datos

Andrea Herrerías, Luis F. Romero, Gerardo Bandera y Óscar Plata [1]

Resumen—**La multiplicación dispersa matriz–matriz densa de bloque pequeño (SpMM) es un núcleo típicamente limitado por memoria debido a sus accesos indirectos e irregulares. En este trabajo analizamos un kernel iterativo SpMM sobre una plataforma RISC-V con soporte RVV y jerarquía de memoria reducida, utilizando matrices derivadas de búsquedas kNN acotadas sobre nubes de puntos. Mostramos que, en este contexto, la regularidad estructural del formato disperso tiene un impacto más importante sobre el rendimiento total que la complejidad de la vectorización manual sobre CSR. En particular, una representación ELL de ancho fijo supera, de forma consistente, a CSR, mientras que el reordenamiento Morton es la única estrategia evaluada que amortiza claramente su coste fijo en tiempo total. Además, el análisis microarquitectónico confirma que la versión RVV con intrínsecos reduce el número de instrucciones ejecutadas, pero sigue limitada por la latencia de los accesos indirectos y de las reducciones vectoriales. Estos resultados indican que, en RISC-V con memoria limitada, el formato de almacenamiento y la localidad de memoria pueden dominar sobre la vectorización explícita.**

Palabras clave—**RISC-V, RVV, SpMM, matrices dispersas, ELLPACK, CSR, Morton, RCM.**

I. Introducción

La multiplicación dispersa matriz–matriz densa de bloque pequeño (*Sparse Matrix–Dense Matrix Multiplication*, SpMM) es un núcleo fundamental en métodos iterativos de simulación, procesado geométrico y computación científica. En este tipo de aplicaciones, el rendimiento suele estar dominado por el acceso irregular a memoria más que por el coste aritmético del kernel, por lo que la localidad de datos y la estructura del formato disperso resultan determinantes, especialmente en arquitecturas con jerarquías de memoria reducidas. [1], [2]

Este problema es particularmente relevante en plataformas RISC-V orientadas a eficiencia energética. Aunque la extensión vectorial RVV permite explotar paralelismo de datos, su efectividad en kernels dispersos sigue limitada por la irregularidad estructural del formato y por el coste de los accesos indirectos. En consecuencia, la mera vectorización del código no siempre produce mejoras proporcionales en tiempo de ejecución. [3], [4]

En este trabajo se parte de una observación central: en un SpMM iterativo sobre RISC-V, modificar la representación de la matriz dispersa puede ser más beneficioso que insistir únicamente en la vectorización manual de un formato irregular. En lugar de centrarse solo en CSR, se estudia una representación más regular basada en ELLPACK de ancho fijo. La hipótesis es que, cuando el número de elementos no nulos por fila está acotado, la regularidad del formato reduce la variabilidad del flujo de control y expone al hardware un patrón de ejecución más uniforme.

Las matrices evaluadas proceden de una nube de puntos sobre la que se realiza una búsqueda local de vecinos. A partir de esa vecindad se construye una matriz de pesos, y dado que la búsqueda está limitada a un máximo de 32 vecinos por punto, las matrices resultantes contienen como máximo 32 elementos no nulos por fila. Esta propiedad hace especialmente natural el uso de una representación ELLPACK de ancho fijo 32.

Además del formato de almacenamiento, se analizan dos estrategias de reordenamiento con objetivos distintos: Morton, orientado a mejorar la localidad espacial, y Reverse Cuthill–McKee (RCM), orientado a reducir el ancho de banda de la matriz. En ambos casos, el interés no reside solo en mejorar el tiempo por iteración, sino en determinar si el coste fijo del reordenamiento se amortiza en el tiempo total de ejecución. [5], [6]

El estudio se realiza sobre una plataforma RISC-V con soporte RVV, utilizando un kernel iterativo SpMM sobre tres conjuntos de datos de 18 670, 24 209 y 33 853 puntos. Se evalúa conjuntamente el impacto del formato, del reordenamiento y de una implementación manual con intrínsecos RVV. Los resultados muestran que el cambio de formato hacia ELLPACK mejora de forma consistente frente a CSR, que Morton proporciona el mejor compromiso entre coste inicial y mejora acumulada, y que la versión con intrínsecos RVV es competitiva cuando la estructura de datos ha sido previamente regularizada.

Las contribuciones principales de este trabajo son: (i) una evaluación experimental del impacto de un formato disperso de ancho fijo sobre un kernel iterativo SpMM en RISC-V; (ii) una comparación entre reordenamiento espacial y topológico, considerando explícitamente el tiempo total de ejecución; y (iii) un análisis del comportamiento de una implementación manual con intrínsecos RVV frente a versiones escalar y autovectorizada.

II. Fundamentos y trabajo relacionado

A. SpMM disperso como kernel limitado por memoria

La multiplicación dispersa matriz–matriz densa de bloque pequeño presenta un comportamiento característico de los kernels limitados por memoria. Aunque el número de operaciones en coma flotante por elemento no nulo es reducido, el patrón de acceso a

[1]Dpto. de Arquitectura de Computadores, Universidad de Málaga, e-mail: andreaherrerias@uma.es

los datos depende de la estructura dispersa de la matriz y, por tanto, introduce indirecciones que reducen la localidad espacial y temporal. En consecuencia, el rendimiento suele estar más condicionado por el subsistema de memoria que por la capacidad aritmética pico del procesador. [1], [2]

En un escenario iterativo, esta limitación se vuelve todavía más importante. Cuando la salida de una iteración se reutiliza como entrada de la siguiente, la presión sobre la caché de datos aumenta y cualquier mejora en la regularidad del acceso o en la organización de los datos puede tener un efecto acumulativo significativo sobre el tiempo total de ejecución.

B. CSR frente a ELL en procesadores vectoriales

CSR (*Compressed Sparse Row*) es uno de los formatos dispersos más utilizados por su compacidad y eficiencia en almacenamiento. Sin embargo, en arquitecturas vectoriales presenta una limitación importante: la longitud de las filas es variable. Esta variabilidad introduce irregularidad en el flujo de control, dificulta la construcción de bucles uniformes y reduce la efectividad de la autovectorización y del acceso predecible a memoria. [7], [8]

ELL adopta una estrategia distinta: cada fila reserva una ancho fijo de almacenamiento. En el caso considerado en este trabajo, la estructura del problema permite utilizar una ancho fijo de 32 elementos por fila, ya que las matrices proceden de una búsqueda de vecinos sobre una nube de puntos con vecindad acotada. De este modo, una estructura dispersa irregular pasa a procesarse mediante un recorrido por fila completamente regular, usando relleno cuando la ocupación real es menor. Esta regularidad simplifica el control y resulta especialmente favorable para arquitecturas vectoriales.

C. Reordenamiento y localidad: Morton frente a RCM

Además del formato de almacenamiento, otra estrategia habitual para mejorar el comportamiento de kernels dispersos consiste en reordenar la matriz y las estructuras asociadas con el fin de mejorar la localidad de memoria o la organización topológica del problema.

Morton, también conocido como orden Z, es una técnica de reordenamiento espacial que preserva la proximidad geométrica en la linealización de índices. En matrices derivadas de vecindades sobre nubes de puntos, este tipo de ordenación puede reducir la distancia efectiva entre accesos indirectos consecutivos y mejorar la localidad espacial.

Por su parte, Reverse Cuthill–McKee (RCM) es una técnica clásica orientada a reducir el ancho de banda de la matriz acercando los elementos no nulos a la diagonal. A diferencia de Morton, RCM actúa sobre la estructura topológica del grafo asociado a la matriz dispersa. [5], [6]

Ambas técnicas persiguen mejorar el comportamiento del kernel desde perspectivas distintas: Morton mejora la localidad espacial, mientras que RCM reduce la distancia topológica en la matriz. En un SpMM iterativo, su utilidad debe evaluarse no solo por el tiempo por iteración, sino también por su capacidad para amortizar el coste fijo de preprocesado en el tiempo total de ejecución.

D. RVV y regularidad del formato

La extensión vectorial RVV de RISC-V proporciona un modelo flexible para expresar paralelismo de datos, pero su efectividad en kernels dispersos sigue dependiendo de la regularidad del formato de almacenamiento y del patrón de acceso a memoria. Cuando el kernel conserva una estructura irregular, la reducción del número de instrucciones escalares no implica necesariamente una reducción equivalente del número de ciclos, especialmente si los accesos indirectos siguen dominando el tiempo de ejecución. [3], [4]

Por ello, en este tipo de arquitecturas, la vectorización debe entenderse como una optimización dependiente del formato. Un formato más regular no solo facilita el uso explícito de intrínsecos vectoriales, sino que reduce la sobrecarga de control y expone al hardware un camino de ejecución más uniforme. Esta observación motiva el estudio conjunto de ELLPACK, RVV y el reordenamiento realizado en este trabajo.

III. Metodología e implementación

A. Kernel iterativo SpMM

El kernel estudiado corresponde a una multiplicación iterativa de una matriz dispersa por una matriz densa de bloque pequeño. En cada iteración se calcula

$$\mathbf{Y} = \mathbf{W}\mathbf{X}, \tag{1}$$

donde $\mathbf{W}$ es una matriz dispersa de tamaño $N \times N$ y $\mathbf{X}$ es una matriz densa de tamaño $N \times 3$ que contiene las coordenadas espaciales (x, y, z) asociadas a cada punto de la nube. Tras cada iteración, la salida pasa a utilizarse como entrada de la siguiente, de modo que el estudio se realiza en un entorno de 700 iteraciones con dependencia temporal entre pasos sucesivos. Esta configuración incrementa la presión sobre la caché de datos y hace de la localidad de memoria un factor central del rendimiento.

Las matrices se construyen a partir de una búsqueda local de vecinos sobre la nube de puntos. Como esta búsqueda está limitada a un máximo de 32 vecinos por punto, la matriz dispersa resultante contiene como máximo 32 elementos no nulos por fila, lo que justifica el uso de una representación ELL de ancho fijo 32.

La Figura 1 muestra el patrón de no nulos de una de las matrices del estudio tras aplicar el reordenamiento Morton. Puede observarse una estructura dispersa con una banda relativamente compacta en torno a la diagonal, coherente con el origen geométrico del problema y con la limitación del número de vecinos por fila.

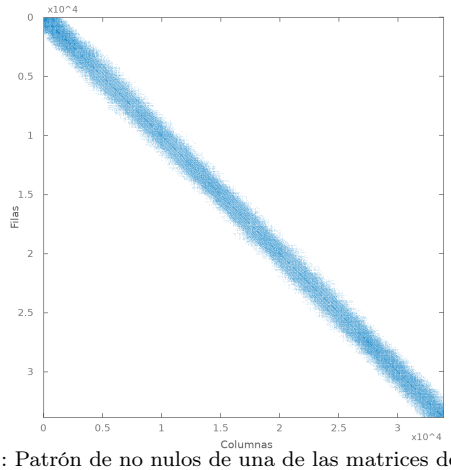


Fig. 1: Patrón de no nulos de una de las matrices del estudio.

B. Formatos evaluados

Se comparan dos formatos de almacenamiento disperso: CSR y ELLPACK.

En CSR, la matriz se representa mediante los vectores de valores, índices de columna y punteros de fila. Esta organización es compacta, pero mantiene una longitud de fila variable, lo que introduce una sobrecarga de control y un patrón de ejecución no uniforme.

En ELLPACK, cada fila reserva un ancho fijo de 32 posiciones. Cuando una fila tiene menos de 32 elementos no nulos, las posiciones restantes se completan mediante relleno con ceros. Esta decisión incrementa el almacenamiento respecto a CSR, pero garantiza un flujo de ejecución constante de 32 elementos por fila, completamente alineado con la estructura del problema estudiado.

C. Implementaciones comparadas

El análisis experimental se centra en tres implementaciones:

- **CSR autovectorizado por compilador**, utilizado como línea base.
- **ELLPACK escalar optimizado**, que explota la regularidad del formato, manteniendo una implementación escalar especializada.
- **ELLPACK con intrínsecos RVV**, implementado manualmente para aprovechar la estructura fija por fila.

Esta selección permite estudiar por separado el impacto del formato, el beneficio de la regularidad estructural y el potencial real de una vectorización explícita sobre un hardware RISC-V con soporte RVV.

D. Implementación ELLPACK con intrínsecos RVV

La implementación manual con intrínsecos RVV se construye a partir del ancho fijo de 32 elementos por fila. Esta decisión permite eliminar el *overhead* asociado a la reconfiguración dinámica del vector dentro de cada fila. En particular, al fijar el ancho a 32, desaparece la necesidad de gestionar de forma repetida la lógica de `vsetvl` (configuración de los vectores) en el bucle interno, lo que reduce la sobrecarga de control respecto a una implementación genérica sobre filas de longitud variable.

Para cada fila, el kernel carga los valores asociados a sus 32 posiciones y procesa en paralelo las tres componentes densas de entrada. Dependiendo de la estructura concreta de los datos, el acceso puede expresarse mediante cargas vectoriales regulares sobre estructuras contiguas en memoria o, cuando intervienen índices de columna, mediante gathers vectoriales sobre la entrada densa. En la implementación evaluada, los accesos indirectos se realizan mediante instrucciones del tipo `vluxei32.v`, apoyadas en una estructura de desplazamientos precalculados para reducir el trabajo de direccionamiento dentro del camino caliente. Los valores de la fila se cargan mediante instrucciones vectoriales contiguas del tipo `vle32.v`.

La acumulación se realiza en registros vectoriales independientes para cada una de las tres componentes del bloque denso. Una vez finalizado el producto correspondiente a la fila, el resultado parcial de cada acumulador se consolida mediante una reducción vectorial `vfredusum.vs`, obteniendo finalmente tres escalares de salida, uno por componente. Gracias a esta organización, la implementación manual evita parte de la lógica de control inherente a CSR y explota de forma directa la regularidad introducida por ELLPACK.

E. Reordenamiento

Además del formato de almacenamiento, el estudio considera tres configuraciones de datos:

- **Sin reordenamiento**.
- **Morton**, para mejorar la localidad espacial.
- **Reverse Cuthill–McKee (RCM)**, para reducir el ancho de banda de la matriz.

El reordenamiento no forma parte del núcleo iterativo del cálculo, sino que se aplica una única vez como fase de preprocesado. No obstante, modifica la organización de la matriz dispersa y de las estructuras densas asociadas, de modo que su efecto se propaga a todas las iteraciones posteriores. Por ello, su evaluación debe considerar no solo la mejora en tiempo por iteración, sino también su capacidad para amortizar el coste fijo inicial dentro del tiempo total de ejecución.

F. Criterio de evaluación

La metodología experimental utiliza dos métricas principales. La primera es el tiempo por iteración, que mide el comportamiento del kernel una vez fijada la representación de los datos. La segunda es el tiempo total de ejecución, que incorpora tanto la fase iterativa como el coste fijo del reordenamiento. Esta distinción es esencial para comparar, de forma justa, técnicas que mejoran la localidad o la regularidad del cómputo a cambio de un coste previo de preparación.

Además, el estudio se complementa con métricas microarquitectónicas orientadas a analizar la evolución del número de instrucciones ejecutadas, la efi-

ciencia de ejecución y el comportamiento de la memoria caché, con el fin de interpretar con mayor precisión las diferencias observadas entre variantes.

IV. Evaluación experimental

A. Plataforma experimental

Los experimentos se realizaron sobre una placa Banana Pi BPI-F3 basada en el procesador Spacemit X60, con ocho núcleos `riscv64` y soporte para la extensión vectorial RVV. La arquitectura dispone de una longitud vectorial de 256 bits y presenta una jerarquía de memoria reducida, sin caché de último nivel. El sistema cuenta con 256 KiB de caché L1 de datos agregada (8 instancias), 256 KiB de caché L1 de instrucciones agregada y 1 MiB de caché L2 agregada (2 instancias). La frecuencia máxima observada es de 1.6 GHz. En todos los experimentos se utilizaron 8 hilos mediante OpenMP.

El código se compiló con GNU `g++` 14.2.0, estándar C++17 y configuración `Release`. Las opciones comunes de compilación incluyeron `-O3`, `-fomit-frame-pointer`, `-funroll-loops`, `-ffast-math`, `-fno-trapping-math`, `-fno-math-errno`, junto con `-fopenmp`. En la plataforma RISC-V se forzó además `-march=rv64gcv` y `-mabi=lp64d`. La variante CSR autovectorizada empleó adicionalmente `-ftree-vectorize` y `-fopenmp-simd`, mientras que en las implementaciones con intrínsecos se desactivó la autovectorización automática del compilador mediante `-fno-tree-vectorize` y `-fno-tree-slp-vectorize`.

La carga evaluada corresponde a un kernel iterativo SpMM con actualización de la entrada tras cada iteración. Cada configuración se ejecutó durante 700 iteraciones y se repitió 5 veces, reportando el valor medio obtenido. El análisis se realiza sobre tres conjuntos de datos reales de 18 670, 24 209 y 33 853 puntos.

B. Coste de almacenamiento de los formatos

Aunque la evaluación del kernel se centra en CSR y ELL, se incluye también el *footprint* de otros formatos representativos. La Tabla I resume el coste relativo de almacenamiento, normalizado respecto a COO, para los tres tamaños de matriz analizados.

Los resultados indican que CSR, SELL-C-σ y ELL forman el grupo más compacto para este tipo de matrices, mientras que formatos como DIA-CSR resultan claramente menos favorables en memoria. En particular, ELL mantiene un coste de almacenamiento cercano a CSR en los tres datasets analizados. Esto refuerza la idea de que la mejora de rendimiento observada posteriormente no depende de un sobrecoste de memoria desproporcionado, sino de la regularidad estructural del formato.

Esta comparación es importante porque permite interpretar mejor los resultados de ejecución. En estas matrices, la ventaja de ELL no se obtiene a costa de una penalización extrema de memoria, sino en un rango de almacenamiento todavía cercano al de CSR y SELL-C-σ. Esto resulta coherente con la baja variabilidad del número de no nulos por fila y con el origen geométrico del problema, donde una representación de ancho fijo resulta natural.

Tabla I: Coste de almacenamiento relativo de formatos dispersos representativos, normalizado respecto a COO.

Formato	18 670	24 209	33 853
CSR	67.7 %	67.8 %	67.6 %
SELL-C-σ	71.3 %	70.2 %	69.0 %
ELL	72.7 %	70.1 %	68.9 %
DIA-CSR	124.7 %	129.9 %	83.8 %

C. Impacto del formato y del reordenamiento en tiempo total

La primera conclusión relevante del estudio es que el cambio de formato de almacenamiento tiene un impacto más importante que la mera complejidad de la vectorización sobre CSR. La Figura 2 resume el efecto conjunto del formato de almacenamiento y del reordenamiento Morton sobre el tiempo total de ejecución. En los tres tamaños de matriz considerados, el uso de Morton reduce de forma consistente el tiempo total respecto a la configuración sin reordenamiento, independientemente del kernel evaluado. Además, las dos variantes basadas en ELL obtienen sistemáticamente mejores resultados que CSR autovectorizado, lo que refuerza la idea central de este trabajo: la regularidad estructural del formato tiene un impacto más decisivo en el rendimiento global que la mera complejidad de la vectorización sobre un formato irregular.

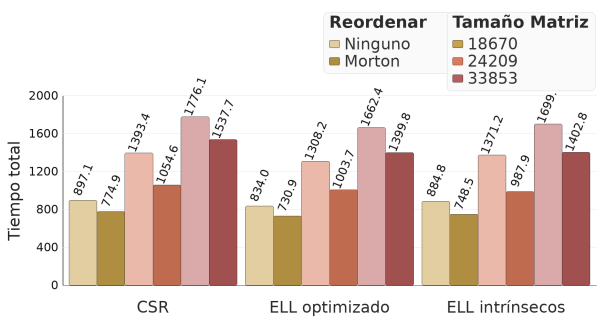


Fig. 2: Tiempo total medio de ejecución en ms para CSR autovectorizado, ELL optimizado y ELL con intrínsecos RVV en tres tamaños de matriz, comparando las configuraciones sin reordenamiento y con reordenamiento Morton.

La mejora introducida por Morton es visible en las tres implementaciones, pero resulta especialmente relevante cuando se combina con ELL, ya que el formato regular y el reordenamiento actúan sobre aspectos complementarios del problema. Mientras ELL reduce la irregularidad del flujo de ejecución al imponer un ancho fijo por fila, Morton mejora la localidad espacial de los accesos indirectos sobre la entrada densa. Como consecuencia, la mejor configuración global pertenece en todos los casos a la familia ELL con reordenamiento Morton, con ELL optimizado como mejor opción en dos tamaños de matriz y ELL con intrínsecos como mejor resultado en el tamaño intermedio.

Este comportamiento confirma también una observación importante de la bateria de pruebas base sin reordenamiento: incluso en ausencia de transformaciones adicionales, la versión ELL optimizada ya supera de forma consistente a CSR autovectorizado en los conjuntos de datos evaluados. Por tanto, la ventaja de ELL no depende exclusivamente del reordenamiento, sino que aparece ya desde el propio cambio de representación dispersa.

D. Morton frente a RCM

La segunda comparación analiza el efecto del reordenamiento sobre las implementaciones finalistas. Los resultados muestran que Morton ofrece de forma consistente el mejor compromiso entre el coste inicial y la mejora acumulada. En la bateria de pruebas de reordenamiento sobre los kernels finalistas, Morton es el ganador en tiempo total en todos los conjuntos de datos evaluados, mientras que RCM queda por detrás debido a su mayor coste fijo de preprocesado.

Los resultados agregados muestran que la ventaja de Morton no se explica únicamente por una reducción del tiempo iterativo, sino por su mejor equilibrio entre coste inicial y beneficio acumulado. La Figura 3 descompone el tiempo total medio en dos componentes: el coste iterativo del kernel y el *overhead* asociado al reordenamiento. Esta descomposición permite observar con claridad por qué Morton resulta la opción más favorable cuando se considera el tiempo total de ejecución.

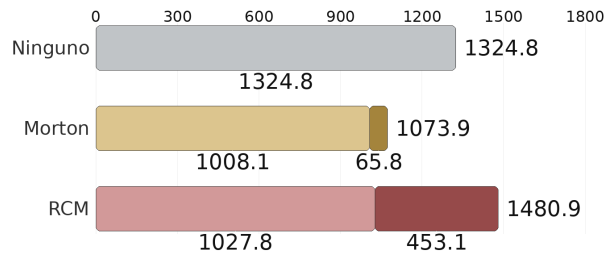


Fig. 3: Descomposición del tiempo total medio en ms en coste iterativo y *overhead* de reordenamiento.

Como puede observarse en la Figura 3, la configuración sin reordenamiento evita cualquier coste inicial, pero mantiene el mayor tiempo iterativo. Morton reduce de forma clara ese coste iterativo e introduce un *overhead* muy pequeño, del orden de 65.8 ms. En cambio, RCM logra tiempos iterativos similares a Morton, pero a costa de un coste fijo de reordenamiento muy superior, en torno a 453 ms. Como consecuencia, esta estrategia empeora el tiempo total frente a Morton e incluso puede quedar por encima de configuraciones más simples.

Este resultado es especialmente relevante en un escenario iterativo. El reordenamiento no forma parte del núcleo de cálculo, sino que se aplica una sola vez como fase de preprocesado. Por ello, su utilidad práctica depende de que la mejora acumulada a lo largo de las iteraciones compense el coste fijo inicial. En las condiciones evaluadas, Morton es la única estrategia que satisface claramente este criterio.

E. Mejor configuración global

Al combinar formato y reordenamiento, las mejores configuraciones globales corresponden a ELL junto con Morton. En la bateria de pruebas final afinada, tanto la versión ELL optimizada como la variante con intrínsecos RVV superan de forma consistente a la línea base CSR autovectorizada. Además, la diferencia agregada entre ELL optimizado y ELL con intrínsecos resulta muy reducida: en el agregado de los tres conjuntos de datos, la versión con intrínsecos queda solo un 0.19 % por detrás del mejor resultado global, lo que constituye un empate técnico desde el punto de vista práctico.

Este comportamiento refuerza la tesis central del artículo. Una vez que la estructura del problema se regulariza mediante ELL y se mejora la localidad mediante Morton, la vectorización explícita con RVV puede acercarse mucho al mejor resultado obtenido. Sin embargo, el factor más determinante sigue siendo el cambio de formato, ya que incluso la versión escalar de ELL supera a CSR autovectorizado.

F. Análisis microarquitectónico

El análisis con contadores `perf` permite interpretar con mayor precisión la diferencia entre las implementaciones. Los datos muestran que la versión ELL con intrínsecos reduce de forma drástica el número de instrucciones ejecutadas respecto a CSR autovectorizado y ELL optimizado. Sin embargo, esta reducción no se traduce en una caída proporcional del número de ciclos, ya que el IPC de la ruta con intrínsecos es notablemente menor.

Para profundizar en estas diferencias, la Figura 4 compara dos métricas microarquitectónicas representativas: el IPC y la tasa de fallos en la caché L1 de datos. Ambas métricas se muestran para las tres implementaciones finalistas, comparando las configuraciones sin reordenamiento y con reordenamiento Morton. Esta vista permite separar con mayor claridad el efecto de la localidad de memoria del efecto puramente asociado al número de instrucciones ejecutadas.

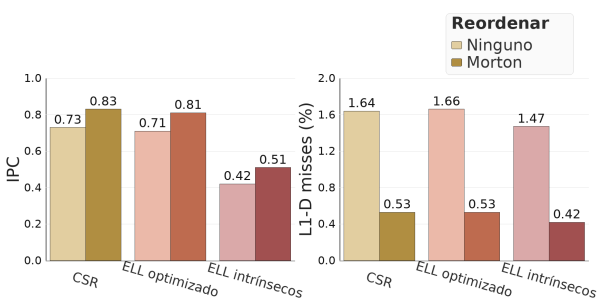


Fig. 4: Comparación microarquitectónica de las implementaciones en términos de IPC y tasa de fallos en L1-D, para las configuraciones sin reordenamiento y con reordenamiento Morton.

Como se observa en la Figura 4, el efecto más claro de Morton es la fuerte reducción de los fallos en L1-D. En las tres implementaciones, la tasa de fallos pasa de valores situados aproximadamente entre el 1.47 % y el 1.66 % a valores entre el 0.42 % y el 0.53 %, lo

que evidencia una mejora sustancial de la localidad efectiva de los accesos indirectos. Esta reducción de fallos se traduce, además, en un aumento sistemático del IPC, que pasa de 0.73 a 0.83 en CSR, de 0.71 a 0.81 en ELL optimizado y de 0.42 a 0.51 en ELL con intrínsecos.

La figura también permite interpretar mejor el comportamiento de la implementación con intrínsecos. Aunque esta variante reduce el número de instrucciones ejecutadas, su IPC sigue siendo el menor de las tres, incluso tras aplicar Morton. Esto indica que la ruta vectorial continúa limitada por la latencia de los *gathers* y de las reducciones, que el hardware evaluado no logra solapar completamente. Sin embargo, la mejora simultánea en IPC y en fallos de caché confirma que la combinación de ELL y Morton empuja el kernel en la dirección correcta: la ruta con intrínsecos no está limitada por un exceso de trabajo dinámico, sino por la latencia de operaciones vectoriales de memoria no contigua.

Aun así, el resultado es muy relevante: aunque la ruta manual con intrínsecos esté limitada por la latencia de memoria no contigua y por el coste de las reducciones, su tiempo por iteración queda prácticamente al nivel del mejor kernel escalar optimizado. Esto valida la dirección de la optimización y confirma que, una vez regularizado el formato de almacenamiento, la vectorización manual sobre RVV es una estrategia plenamente competitiva.

V. Conclusiones y trabajo futuro

Este trabajo ha estudiado el comportamiento de un kernel iterativo SpMM sobre una plataforma RISC-V con soporte RVV y jerarquía de memoria limitada, evaluando de forma conjunta el impacto del formato de almacenamiento, del reordenamiento de datos y de la vectorización explícita. Los resultados muestran que, en esta clase de cargas, la regularidad estructural de la matriz dispersa resulta tan importante como la propia estrategia de vectorización.

La principal conclusión del estudio es que la combinación de ELLPACK de ancho fijo 32 y reordenamiento Morton constituye la configuración más recomendable para este tipo de SpMM iterativo sobre RISC-V. ELLPACK regulariza el flujo de ejecución y reduce la sobrecarga de control asociada a filas de longitud variable, mientras que Morton mejora la localidad espacial de los accesos indirectos y ofrece el mejor compromiso entre coste inicial y beneficio acumulado en tiempo total.

Asimismo, la implementación manual con intrínsecos RVV demuestra ser una alternativa competitiva. No obstante, su rendimiento sigue estando limitado por la latencia de las instrucciones de acceso indirecto y por el coste de las reducciones vectoriales sobre el hardware evaluado. En consecuencia, la vectorización manual no debe entenderse como un sustituto del cambio de formato, sino como una optimización que alcanza su máximo potencial una vez que la estructura de datos ha sido previamente regularizada.

En conjunto, los resultados confirman que, en una arquitectura RISC-V con recursos de memoria modestos, el cambio desde CSR hacia un formato ELLPACK ajustado a la estructura real del problema proporciona un beneficio más determinante que la complejidad adicional de optimizar agresivamente un kernel irregular. La elección del formato y del reordenamiento adecuado define, en gran medida, el techo de rendimiento alcanzable.

Como trabajo futuro, resulta interesante extender este análisis a matrices de mayores dimensiones, donde el impacto del reordenamiento y de la jerarquía de memoria podría tener más relevancia. También sería interesante evaluar nuevas implementaciones RVV capaces de reducir la penalización asociada a *gathers* y reducciones en arquitecturas RISC-V de próxima generación. Finalmente, un estudio sobre diferentes distribuciones de vecindad permitiría delimitar con mayor precisión en qué condiciones la regularidad del formato domina sobre la complejidad de la vectorización explícita.

Agradecimientos

El presente trabajo ha sido financiado mediante el proyecto TSI-069100-2023-0013

Referencias

[1] Samuel Williams, Andrew Waterman, and David Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[2] Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, and James Demmel, "Optimization of sparse matrix-vector multiplication on emerging multicore platforms," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2007.

[3] Krste Asanović and David A. Patterson, "Instruction sets should be free: The case for risc-v," Tech. Rep. UCB/EECS-2014-146, EECS Department, University of California, Berkeley, 2014.

[4] RISC-V International, "The risc-v instruction set manual, volume i: Unprivileged isa, chapter 31: V standard extension for vector operations, version 1.0," 2024, Official RISC-V documentation.

[5] Elizabeth Cuthill and James McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 24th National Conference of the ACM*, 1969, pp. 157–172.

[6] Wai-Hung Liu and Andrew H. Sherman, "Comparative analysis of the cuthill–mckee and the reverse cuthill–mckee ordering algorithms for sparse matrices," *SIAM Journal on Numerical Analysis*, vol. 13, no. 2, pp. 198–213, 1976.

[7] Yousef Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, 2 edition, 2003.

[8] Nathan Bell and Michael Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *Proceedings of the ACM/IEEE Conference on High Performance Computing (SC)*, 2009.

Tecnologías clúster, plataformas distribuidas, BigData y Deep Learning (4/4)

Diseño, despliegue y gestión de un sistema de alto rendimiento para computación científica

Cosmin Petre[1], Javier García-Blas[1], David Expósito-Singh[1] y Jesús Carretero[1]

Resumen— **El avance de la ciencia moderna y la ingeniería está intrínsecamente ligado a la capacidad de procesamiento de datos y la simulación numérica a gran escala en supercomputadores. Sin embargo, la transición de entornos de experimentación a sistemas HPC en producción conlleva una complejidad técnica y logística que a menudo se subestima. Desplegar un sistema de cómputo de altas prestaciones para uso productivo no consiste únicamente en la agregación de *hardware* de vanguardia, implica la creación de un ecosistema resiliente capaz de ofrecer disponibilidad, escalabilidad y eficiencia de manera sostenida. En este artículo se describe el diseño, despliegue y gestión del sistema de alto rendimiento del Centro de Computación Científica (C3) de la Universidad Carlos III de Madrid. El C3 se presenta como un caso de estudio de una infraestructura Tier-2 orientada a la producción científica real, diseñada para ofrecer un ecosistema resiliente, escalable y eficiente.**

Palabras clave— **Arquitectura de Computadores, Supercomputación, Computación de Altas Prestaciones.**

I. Introducción

HOY en día el progreso de la ciencia y la ingeniería depende en gran medida de la capacidad de procesar grandes cantidades de datos y ejecutar simulaciones numéricas a gran escala. La Computación de Alto Rendimiento (HPC, por sus siglas en inglés) ha dejado de ser una herramienta de nicho para convertirse en el pilar fundamental de disciplinas que van desde la astrofísica y la genómica hasta la inteligencia artificial y la dinámica de fluidos. No obstante, en el proceso de convertir un sistema de altas prestaciones experimental en una infraestructura HPC en producción surgen desafíos técnicos y logísticos que se suelen subestimar [1].

El despliegue de una infraestructura HPC no se limita a ofrecer *hardware* avanzado, sino que requiere mantener un ecosistema robusto, escalable y eficiente para garantizar la disponibilidad continua del servicio [2]. Los desafíos principales incluyen:

- Heterogeneidad del hardware: la integración de arquitecturas diversas (CPUs multicanal, aceleradores GPU, FPGAs) exige pilas de software (*software stacks*) extremadamente complejas para garantizar la interoperabilidad y el rendimiento óptimo de las aplicaciones.
- Gestión de cargas de trabajo y orquestación: en producción, el sistema debe gestionar de forma equitativa y eficiente las peticiones de cientos de usuarios mediante gestores de colas (como Slurm), minimizando los tiempos de espera y maximizando la utilización de los recursos.
- Eficiencia energética y refrigeración: la densidad de cómputo actual genera demandas térmicas y eléctricas críticas que requieren infraestructuras de centros de datos de última generación para evitar el estrangulamiento térmico (*thermal throttling*) y garantizar la sostenibilidad económica.
- Sistemas de almacenamiento: los sistemas de almacenamiento actuales deben de ser capaces de hacer frente a gran variedad de cargas de trabajo. A día de hoy, la capa de almacenamiento tiene que dar soporte a gran variedad de aplicaciones. Estas aplicaciones presentan un comportamiento muy dispar en términos de acceso, desde aplicaciones del campo de la inteligencia artificial con accesos aleatorios y de pequeño tamaño, a aplicaciones del campo de la computación científica o analítica de datos con accesos grandes y poco frecuentes.

En este trabajo se discute el diseño, despliegue y gestión del sistema de altas prestaciones del Centro de Computación Científica (C3) de la Universidad Carlos III de Madrid, una infraestructura Tier-2 orientada a la producción científica real que ofrece un entorno escalable, eficiente y resiliente.

En este contexto de alta exigencia, C3 se erige como un caso de estudio relevante en el panorama nacional español [3]. El sistema C3 ha sido diseñado no solo para ofrecer una potencia de cálculo bruta superior a la de los sistemas estándar, sino para funcionar como un centro de servicios de apoyo a la investigación (CAI) que da soporte a grupos de I+D+i, organismos públicos y empresas. El despliegue del C3 destaca por su enfoque en la computación híbrida, combinando nodos de cómputo de alta densidad con capacidades avanzadas de almacenamiento confiable. Su arquitectura permite abordar retos de biocomputación, simulaciones físicas complejas y entrenamiento de modelos de IA, integrando herramientas modernas de contenedores, como Apptainer [4] y gestión de software mediante Spack para resolver los problemas de reproducibilidad y gestión de dependencias en entornos compartidos.

Este trabajo analiza los procesos de configuración, optimización y los cuellos de botella enfrentados durante la puesta en marcha del sistema C3, proporcionando una hoja de ruta para instituciones que buscan transformar sus capacidades de investigación a través de infraestructuras HPC robustas y orientadas a la producción científica real.

El resto del trabajo se estructura de la siguiente manera. La sección II introduce los conceptos necesa-

[1]Dpto. de Informática, Universidad Carlos III de Madrid, e-mail: {cpetre,fjblas,dexposit,jcarrete}@inf.uc3m.es

rios para identificar y clasificar supercomputadores. La sección III detalla el diseño del sistema C3. En la sección IV se describen las distintas capacidades de gestión de datos del sistema. El software de gestión de la plataforma es presentando en la sección V. El trabajo también presenta alguno de los casos de uso más destacables del sistema en la sección VI. Finalmente, la sección VII concluye el trabajo y presenta las siguientes líneas de acción.

II. Clasificación de los supercomputadores

Los supercomputadores suelen clasificarse por su potencia de procesamiento (velocidad), medida en FLOPS (operaciones de coma flotante por segundo), y por su enfoque arquitectónico para resolver problemas complejos. Aunque la lista TOP500 clasifica anualmente las máquinas más rápidas, un sistema de clasificación más funcional, especialmente en contextos europeos de HPC, clasifica los supercomputadores según su accesibilidad y alcance.

A. Sistema de clasificación funcional

La organización de los recursos de cálculo de alto rendimiento se articula mediante una estructura jerárquica que pondera tanto la accesibilidad geopolítica como la capacidad de cómputo bruto [5]. En la cúspide de esta pirámide se sitúan los sistemas de Nivel 0 (Tier-0), representados por centros de envergadura continental o internacional. Estas instalaciones albergan máquinas en la escala de petaflops y exaflops, diseñadas para liderar proyectos científicos y de ingeniería de vanguardia a nivel global. Exponentes actuales de este nivel incluyen sistemas como JUPYTER en Alemania, Leonardo en Italia, MareNostrum 5 en España y El Capitan en Estados Unidos.

En un estrato intermedio, el Nivel 1 (Tier-1) comprende los centros nacionales de supercomputación. Estas arquitecturas están dimensionadas para satisfacer las demandas específicas de la comunidad científica y el sector industrial de un país concreto, actuando como el núcleo de la soberanía tecnológica nacional. Casos representativos son el CESGA en España o el sistema VEGA en Eslovenia. Finalmente, el Nivel 2 (Tier-2) abarca la infraestructura de ámbito regional o universitario. Estos sistemas están orientados a proporcionar soporte computacional localizado para grupos de investigación académicos y proyectos de menor escala, destacando en el contexto español instalaciones como COMPUTAEX o el supercomputador Picasso.

B. Clasificación basada en el rendimiento

Los supercomputadores también se agrupan según su capacidad para manejar tareas modernas intensivas en datos:

- Computadores exaescala: Máquinas capaces de realizar más de 1 exaflop (10^{18}) de cálculos por segundo. Ejemplos incluyen Frontier (EE. UU.), Aurora (EE. UU.) y Eagle (Microsoft).
- Computadores petaescala: Máquinas que operan en el rango de mil billones (10^{15}) de FLOPS. Fugaku de Japón fue un líder destacado en este nivel antes del auge de la computación exaescala.

La lista TOP500 [6] se actualiza cada 6 meses desde hace años para mostrar la lista de los 500 supercomputadores más potentes del mundo. En Noviembre 2025, El Capitan (EE. UU.) está clasificado como el supercomputador más potente, seguido de Frontier, Aurora y Jupyter.

C. Clasificación arquitectónica

La taxonomía de los sistemas de alto rendimiento se define primordialmente por su versatilidad funcional y la especialización de su hardware. En primera instancia, los sistemas de propósito general se caracterizan por su capacidad para ejecutar una amplia gama de aplicaciones científicas complejas, tales como el modelado molecular o la predicción meteorológica. Estas arquitecturas suelen fundamentarse en procesadores vectoriales o clústeres masivos, siendo el Marenostrum 5 un exponente de esta categoría [7].

Paralelamente, la emergencia de la Inteligencia Artificial (IA) ha consolidado un segmento crítico de supercomputadores especializados en IA. Desde 2019, este sector presenta un crecimiento exponencial donde el rendimiento computacional se duplica cada nueve meses, a pesar del incremento sostenido en los costes de adquisición y la demanda energética [8]. Un hito en esta evolución es el sistema Colossus de xAI, que integra 200.000 chips de procesamiento masivo. No obstante, existe una convergencia tecnológica en sistemas híbridos como Leonardo BOOSTER, donde servidores dedicados de GPU permiten que arquitecturas de propósito general alcancen una alta eficiencia en tareas de aprendizaje profundo.

Finalmente, los sistemas de propósito específico representan el grado máximo de optimización para tareas singulares, como la dinámica molecular o el cálculo cuántico. A diferencia de las plataformas convencionales, estos equipos emplean circuitos integrados de aplicación específica (ASIC) o arquitecturas basadas en cúbits, permitiendo resolver problemas computacionalmente intratables para arquitecturas clásicas [9].

D. Características clave de los niveles

La operatividad de los sistemas de alto rendimiento se articula en torno a la distinción crítica entre capacidad computacional (*capability*) y capacidad de flujo de trabajo (*capacity*). Mientras que los sistemas de nivel superior se diseñan específicamente para la resolución de un único problema de escala masiva y alta complejidad técnica (*capability computing*), las arquitecturas de niveles inferiores optimizan su rendimiento mediante la ejecución concurrente y eficiente de múltiples tareas de menor envergadura (*capacity computing*). Esta diferenciación determina no solo el diseño del hardware, sino también las políticas de planificación de trabajos en el sistema.

En cuanto a la gestión de datos, el panorama tecnológico de 2024–2025 revela una estandarización casi absoluta hacia sistemas operativos basados en Linux y arquitecturas de almacenamiento masivamente paralelas. La eficiencia de estos ecosistemas de datos se evalúa mediante el ranking IO500 [10], donde actualmente predominan soluciones de alto rendimiento como DAOS, Lustre y WekaIO. Estos sistemas permiten mitigar el cuello de botella que representa la entrada/salida (I/O) en procesos de computación intensiva, garantizando la escalabilidad necesaria para las cargas de trabajo de última generación.

III. Diseño del sistema

Desde el punto de vista de clasificación anterior, C3 es un sistema TIER 2 de propósito general, aunque con una granja de GPUs, que presta servicio a nivel regional en Madrid. En conjunto tiene una capacidad de alrededor de 1,9 PFlops.

A continuación se muestran las principales características del C3

- 11.840 cores de CPU que ofrecen una potencia total de casi 300 TFlops.
- 42 GPUs Nvidia A40 con 48 GB de VRAM cada una, con un total de 451.584 CUDA cores, que proporcionan una potencia total aproximada de 1,6 PFlops.
- 90 nodos de cómputo, cada uno equipado con 2 procesadores AMD con 64 cores físicos de CPU, 1 TB de memoria RAM y 3,5 TB de almacenamiento NVMe local.
- 5 nodos de GPU con 8 o 9 GPUs cada uno. Cada servidor dispone de 64 cores de CPU, 512 GB de memoria RAM y 7 TB de almacenamiento NVMe.
- Red de alta velocidad Liqid para interconexión de nodos servidores de GPU.
- Conexión InfiniBand a 100 Gbps en configuración *Fat Tree*, además de doble conexión Ethernet a 10 Gbps.
- 1 PByte de almacenamiento con servidores Lustre y NFS.

La disposición física de los nodos del sistema se muestra en la Fig. 1.

A. Topología de red

La topología de red del sistema C3, que se muestra en la Fig. 2, se compone de tres redes diferentes: una red InfiniBand de alto rendimiento, una red Ethernet de servicio y una red de gestión. La red InfiniBand sigue una disposición *Fat Tree* y proporciona conexiones de 100 Gbps y baja latencia (RDMA) a todos los nodos del supercomputador, siendo la red principal del clúster. Por otro lado la red de servicio, también en configuración *Fat Tree*, ofrece redundantes conexiones Ethernet a 10 Gbps a todos los nodos y sirve como red secundaria. Finalmente, la red de gestión se emplea para acceder a las interfaces de gestión fuera de banda de las máquinas, principalmente por IPMI, permitiendo la configuración remota de las mismas así como su apagado y encendido.

B. Composición de GPUs

En los sistemas con servidores de GPU tradicionales, habitualmente la asignación de GPUs debe hacerse en base al número de GPUs de los servidores. El problema es que las aplicaciones actuales superan normalmente las capacidades de los servidores tradicionales, que normalmente están limitados a 4–8 GPU.

El sistema C3 incluye un sistema arquitectónico novedoso (de la empresa Liqid Inc [11]) que permite escalar vertical y horizontalmente los servidores de GPU mediante un sistema de agregación de GPUs que permite asignar cualquier número de GPUS a una carga de trabajo para lograr un 100 % de utilización de las mismas. Esta infraestructura componible permite al C3 lograr un mayor rendimiento al proporcionar una forma flexible, escalable y eficiente de gestionar y asignar recursos informáticos de manera dinámica.

Como se muestra en la Fig. 3, este sistema incluye un software de composición de sistemas y una red propia para la gestión de las GPUs. Con este sistema, los servidores de GPUs van conectados a switches PCIe específicos que permiten gestionar todo el tráfico de datos internos entre GPUs, sin afectar a la red de altas prestaciones de las CPUs. Además, el software de control permite ofrecer a SLURM una visión unificada de las GPUs, que se pueden asignar desbordando las fronteras de los servidores. Así, es posible asignar a un trabajo GPUs de 2 servidores, o asignar a un trabajo todas las GPUs del sistema, sin que el usuario sea consciente de la configuración *hardware* de los servidores de GPUs o la distribución de las mismas.

Este sistema permite obtener al menos el doble de tokens por vatio y evita la dependencia del vendedor del clúster, dado que integra cualquier servidor de PCIe de GPUs. Además, en lugar de sobreprovisionar o infrautilizar el hardware, los recursos se optimizan entre múltiples proyectos, reduciendo las pérdidas de tiempo y maximizando el retorno de la inversión, además de simplificar mucho la gestión de trabajos con GPUs en el sistema.

C. Eficiencia energética y refrigeración

La arquitectura térmica del sistema se fundamenta en un esquema de refrigeración *in-row* integrado directamente en la infraestructura de servidores. Como se ilustra en la Fig. 4, el diseño utiliza una disposición intercalada donde los racks de cómputo, encargados del procesamiento de datos, alternan con unidades de enfriamiento especializadas o racks *in-row*. Este modelo operativo optimiza la gestión del flujo de aire mediante la segregación de pasillos fríos y calientes, permitiendo que las unidades de refrigeración capturen el calor residual inmediatamente después de ser expulsado por los nodos de cómputo. El proceso de transferencia térmica se visualiza a través del gradiente de las flechas, que representan la extracción

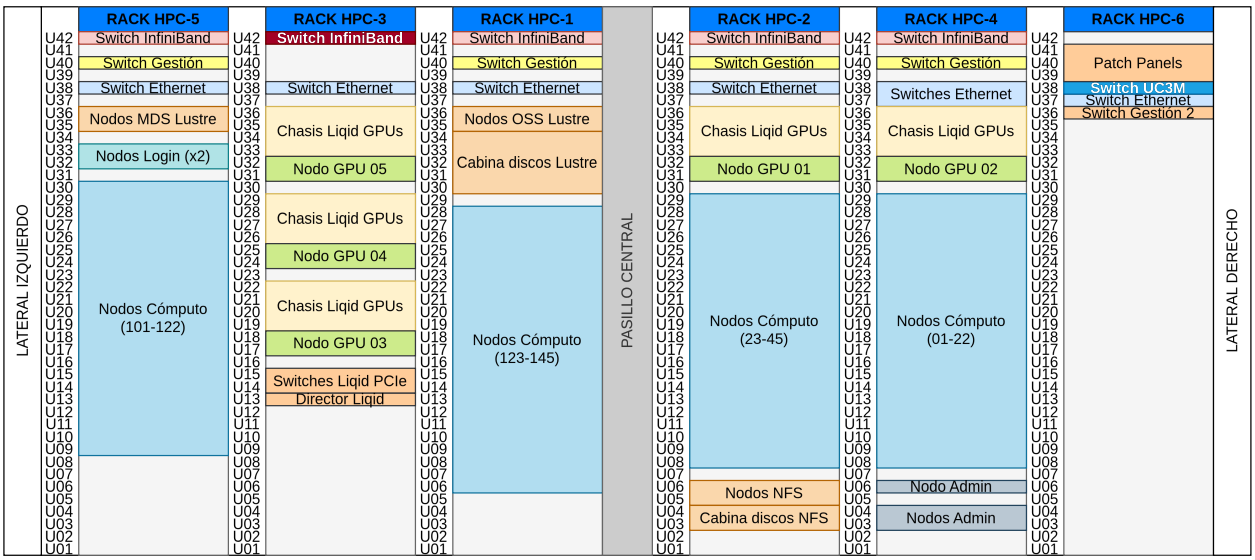


Fig. 1: Disposición física de los elementos del sistema propuesto.

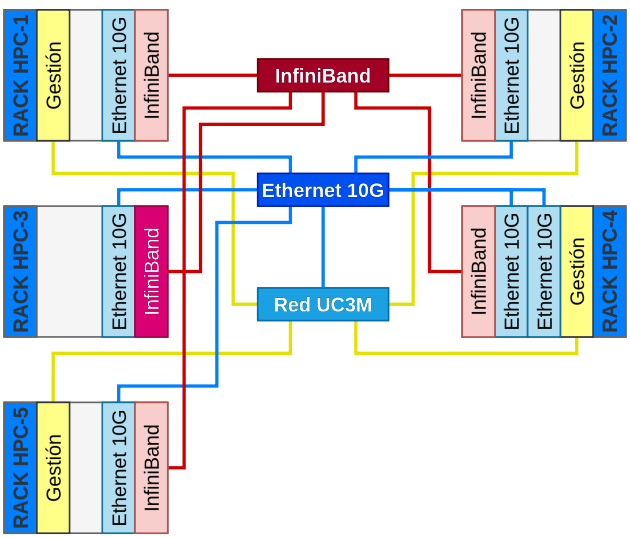


Fig. 2: Arquitectura de la Red del C3.

del aire caliente del pasillo posterior y su posterior reintroducción como aire refrigerado hacia el pasillo frontal. Al situar los intercambiadores de calor en la propia fila, el sistema reduce significativamente las pérdidas por mezcla de aire y la potencia necesaria para la ventilación, garantizando una disipación eficiente y escalable para entornos de alta densidad de potencia.

Por otro lado, para mantener una alta eficiencia energética en el supercomputador se ha implementado la siguiente estrategia de ahorro de energía: cuando el clúster está en modo *standby* (sin trabajos activos) su consumo energético total es aproximadamente 13,6 kW, ya que 86 de los 90 nodos de cómputo de CPU permanecen apagados. En cambio, si todos los nodos ociosos estuvieran encendidos el consumo eléctrico sería de 26,5 kW, ya que los nodos de CPU consumen unos 150 W cada uno como mínimo. De esta manera obtenemos una reducción en el consumo energético en modo *standby* del 49 %.

D. Monitorización del sistema físico

La monitorización del CPD (Centro de Procesamiento de Datos) y del supercomputador C3 es crítica para asegurar su correcto funcionamiento 24*365 y poder responder a las incidencias con la celeridad necesaria para minimizar tiempos de parada y mantenimiento. Habitualmente se usan sensores IoT y software especializado, que permiten detectar fallos, prevenir incidencias y automatizar respuestas, asegurando la continuidad del negocio y optimizando el rendimiento energético.

Los aspectos clave en la monitorización de un CPD son:

- Infraestructura Física (entorno): control de temperatura y humedad (sondas de calor), detección de inundaciones, calidad del aire y gestión de flujos de aire para prevenir sobrecalentamiento.
- Gestión de Energía: monitorización ininterrumpida de sistemas de alimentación ininterrumpida (SAI/UPS), grupos electrógenos, niveles de combustible y consumo eléctrico.
- Seguridad Física: control de accesos a las salas técnicas y vigilancia por vídeo IP para evitar intrusiones.
- Infraestructura Lógica y Red: monitorización del estado de servidores, rendimiento de equipos, gestión inteligente del cableado estructurado y conectividad de red.
- Alertas Automatizadas: sistema de alertas y alarmas inmediatas para actuar proactivamente ante cualquier desviación de los parámetros preestablecidos, reduciendo tiempos de inactividad.

La monitorización del C3 utiliza la plataforma Data Center Expert como solución DCIM (Data Center Infrastructure Management) [12] para integrar todas estas variables en un único panel de control, facilitando la gestión proactiva (ver Fig. 5). Esta herramienta permite automatizar algunas alarmas definiendo

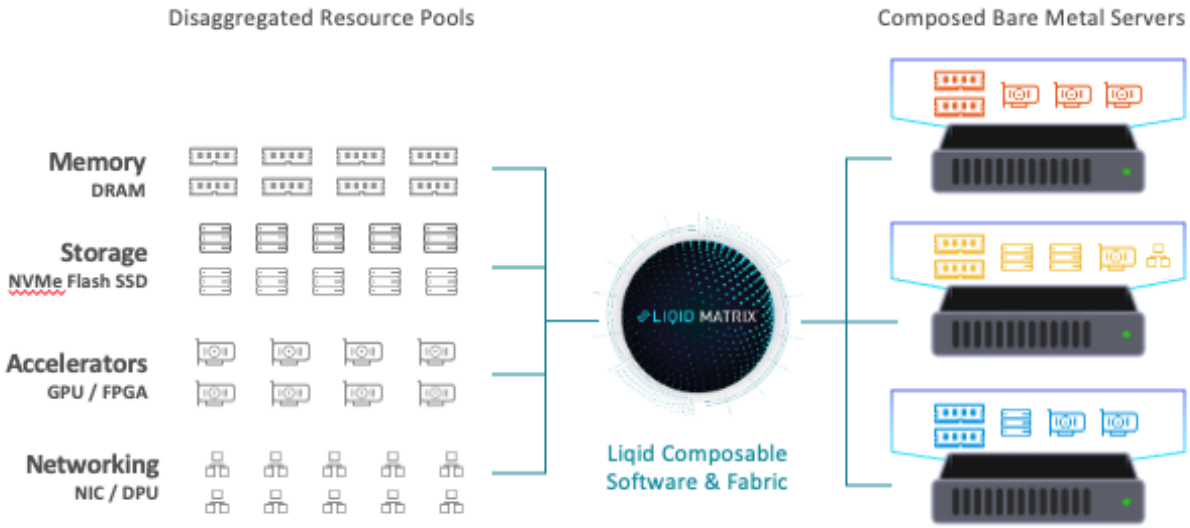


Fig. 3: Sistema de configuración dinámica de GPUs.

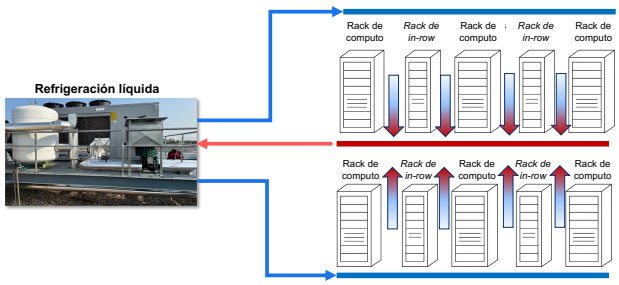


Fig. 4: Sistema de refrigeración *in-row* basado en pasillo caliente/frio.

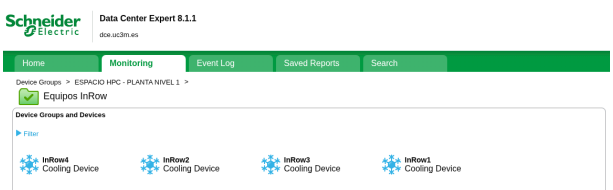


Fig. 5: DataCenter Expert.

umbrales mínimos y máximos para fijar las posibles alertas, que se envían por correo. Además, se ha incrementado la redundancia de sensores IoT de forma externa por los miembros del grupo ARCOS incluyendo sondas de temperatura de sala y pasillo caliente, en la sala SAI y sistemas de visión artificial y arranque remoto del aire acondicionado de respaldo en caso de que fuera necesario.

La Fig. 6 muestra la evolución de temperatura de la sala (azul) y zonas calientes (rojo), la temperatura de consigna (25 grados centígrados) así como el umbral de alarma (35 grados centígrados). Como se puede apreciar, el sistema de refrigeración mantiene la temperatura estable aunque la carga de trabajo varíe. Las oscilaciones del día 1-10-25 se deben a una prueba de mantenimiento forzado de los sistemas de refrigeración. Todos los datos se almacenan en un sistema Prometheus y, en caso de oscilaciones, se envían por correo electrónico a los administradores. En caso de que la temperatura de la sala sobrepase el umbral de alarma, se han generado *scripts* de apagado total controlado de los sistemas (ver algoritmo 1).

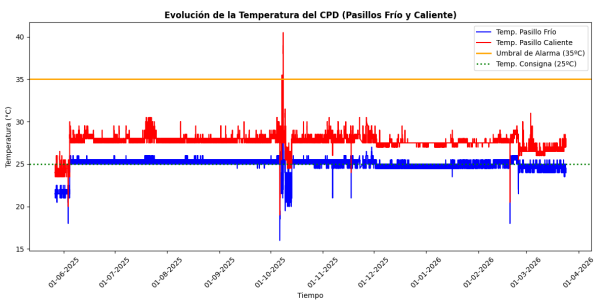


Fig. 6: Evolución de la temperatura del CPD en un año.

Algoritmo 1 Algoritmo de parada total en caso de emergencia.

Require: Temperatura del CPD excede el umbral de alarma
Ensure: Ejecución del algoritmo de apagado total de emergencia
1: **Definir** *Accion* ← Suspend (-s)
2: **Configurar** Entorno y *PATH* local
3: **Almacenar** Registro de alarma en fichero de registros
— Apagado por IPMI —
4: **Filtrar** *hostlist* ← nodos de cómputo, de GPU y de login
5: **for cada** *host* **en** *hostlist* **do**
6: **Ejecutar** `power_ipmi` *Accion host*
7: **end for**

IV. Almacenamiento de datos

El sistema de almacenamiento del C3 se divide en tres niveles: Lustre, NFS y *Scratch*. Su arquitectura se representa esquemáticamente en la Fig. 7. A continuación se detalla cada uno de sus niveles:

- **NFS [13]**: Sistema de ficheros de red tradicional con conexión InfiniBand para acceso *RDMA* de baja latencia y 48 TB de capacidad. Se emplea principalmente como el */home* común de todos los nodos del C3. Está compuesto por dos nodos de almacenamiento configurados en alta disponibilidad, ambos doblemente conectados por *Fibre Channel* a una pequeña cabina de almacenamiento equipada con 8 discos duros de 8 TB cada uno.
- **Lustre [14]**: Sistema de ficheros paralelo de alto rendimiento y gran capacidad (546 TB). Se emplea principalmente para cómputo y para almacenar grandes cantidades de datos. Está disponible tanto desde los nodos de cómputo y GPU como los de login, y se puede acceder tanto por InfiniBand 100 Gbps con *RDMA* como por Ethernet 10 Gbps. Dispone de una configuración de alta disponibilidad, con sendos nodos de metadatos redundantes y dos nodos OSS redundantes que sirven 16 OSTs, divididos entre ambos para balanceo de carga. También cabe mencionar que ambos nodos OSS están doblemente conectados por *Fibre Channel* (*multipath*) a una cabina de almacenamiento equipada con 42 discos duros de 20 TB cada uno.
- ***Scratch***: Cada nodo de cómputo dispone de 3,5 TB de almacenamiento NVMe local (7 TB en los nodos de GPU). Para acceder el usuario debe lanzar primero un trabajo de Slurm, y tendrá acceso temporal al almacenamiento local de los nodos de cómputo que se le hayan asignado. Una vez concluido su trabajo de Slurm todo el contenido almacenado se borrará automáticamente.

V. Software de gestión

A continuación se presentan los principales sistemas de gestión del supercomputador, entre ellos el sistema de colas Slurm y los gestores de paquetes de software.

A. Gestor de trabajos

El supercomputador C3 cuenta con el sistema de colas Slurm para la gestión de recursos y trabajos. Dentro de su configuración cabe destacar la gestión de recursos heterogéneos, permitiendo que el sistema no solo asigne núcleos de CPU, sino que también contabilice el uso de aceleradores *hardware* (GPUs) a nivel de trabajo, facilitando el control completo sobre el consumo de recursos del clúster.

Otro pilar fundamental de esta configuración es su compromiso con la eficiencia energética y sostenibilidad, implementando un sistema dinámico de *Power Saving*. Mediante el uso de los *scripts nodesuspend* (algoritmo 2) y *noderesume* (algoritmo 3), el clúster es capaz de apagar automáticamente los nodos que permanecen inactivos durante más de 30 minutos, optimizando el gasto eléctrico sin sacrificar la disponibilidad para trabajos críticos, ya que los nodos de GPU y algunos nodos de cómputo siempre permanecen encendidos.

Algoritmo 2 Algoritmo de parada de nodos en C3 (nodesuspend).

Require: Lista de nodos sin trabajos activos proporcionada por Slurm (*hostlist*)
Ensure: Ejecución del comando de apagado según el tipo de hardware.

Definir *Accion* ← Suspend (-s)
Configurar Entorno y *PATH* local
Obtener Tabla de Nodos y características mediante `sinfo`
Almacenar resultados en archivo temporal $TMPFILE$
— Procesamiento por IPMI —
if Existe nodo con IPMI `power_ipmi` **then**
    Filtrar *Lista* ← nodos con `power_ipmi`
    Ejecutar `power_ipmi` *Accion Lista*
end if
Eliminar $TMPFILE$

Algoritmo 3 Algoritmo de arranque de nodos en C3 (noderesume).

Require: Lista de nodos proporcionada por Slurm (*hostlist*)
Ensure: Ejecución del comando de encendido según el tipo de hardware.

Definir *Accion* ← Resume (-r)
Configurar Entorno y *PATH* local
if *hostlist* está vacío **then**
    Finalizar ejecución
end if
Obtener Tabla de Nodos y características mediante `sinfo`
Almacenar resultados en archivo temporal $TMPFILE$
— Procesamiento por IPMI —
if Existe nodo con IPMI `power_ipmi` **then**
    Filtrar *Lista* ← nodos con `power_ipmi`
    Ejecutar `power_ipmi` *Accion Lista*
end if
Eliminar $TMPFILE$

La seguridad y el aislamiento de procesos se garantizan mediante la integración de *Cgroups*, lo que asegura que ningún trabajo exceda los límites de memoria o CPU asignados, manteniendo la estabilidad global del sistema.

Los recursos computacionales del C3 están repartidos en diferentes particiones o colas de Slurm, como se puede observar en la tabla I. Existen tres colas generales que dan acceso a los nodos de cómputo, y tres particiones especiales que permiten el uso de los nodos de GPU, cada una con diferentes tiempos máximos, siendo necesario que cada trabajo escoja

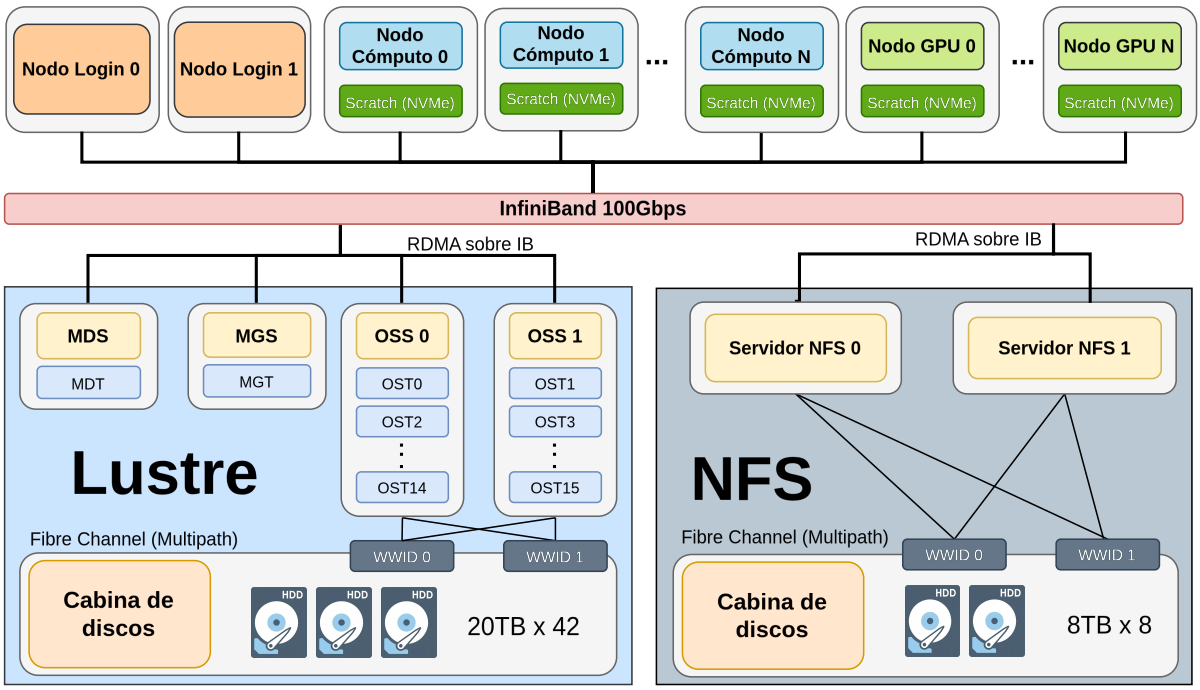


Fig. 7: Arquitectura del sistema de almacenamiento en C3.

explícitamente la cola que más se ajuste a sus necesidades según la duración del mismo. Cabe mencionar que la asignación de nodos tiene en cuenta la topología de red, ya que por defecto Slurm intentará asignar nodos de un mismo *rack* o *racks* contiguos para garantizar una menor latencia en las comunicaciones entre nodos.

B. Contabilidad

Slurm también actúa como la herramienta de contabilidad del supercomputador, guardando registros detallados de todos los trabajos ejecutados en el clúster. Dispone de herramientas como *sreport* que permiten consultar las horas de cómputo de CPU y GPU que un usuario ha consumido durante un periodo de tiempo dado (ver ejemplo de uso en la Fig. 8).

```
# sreport cluster AccountUtilizationByUser -t hour -T cpu,gres/gpu format=cluster,account%30,lo
gin%15,tresname,used start=2026-01-01T00:00:00 end=2026-01-31T23:59:59 Account=cuenta_de_slurm
--------------------------------------------------------------------------------
Cluster/Account/User Utilization 2026-01-01T00:00:00 - 2026-01-31T23:59:59 (2678400 secs)
Usage reported in TRES Hours
--------------------------------------------------------------------------------
  Cluster                        Account          Login      TRES Name     Used
--------- ------------------------------ -------------- -------------- --------
  uc3mhpc                cuenta_de_slurm                            cpu     1703
  uc3mhpc                cuenta_de_slurm                        gres/gpu        7
  uc3mhpc                cuenta_de_slurm       usuario1            cpu        4
  uc3mhpc                cuenta_de_slurm       usuario1        gres/gpu        7
  uc3mhpc                cuenta_de_slurm       usuario2            cpu     1699
  uc3mhpc                cuenta_de_slurm       usuario2        gres/gpu        0
```

Fig. 8: Ejemplo de informe mensual con *sreport*.

C. Gestión de paquetes de supercomputación

Para gestionar las diferentes librerías y paquetes de *software* que se ofrecen en el C3 se emplean dos herramientas: Lmod [15] y Spack [16].

Lmod es un sistema de módulos basado en Lua que permite modificar dinámicamente el entorno del usuario mediante ficheros de módulos. Es una herramienta ampliamente utilizada en muchos centros de supercomputación por todo el mundo. El usuario puede cargar y descargar diferentes módulos de *software* mediante comandos de Lmod (*module load* y *module unload*), modificando la variable de entorno *PATH* de manera transparente. Los módulos en sí son ficheros que proporcionan variables de entorno, las cuales especifican los directorios donde están instalados los diferentes paquetes de software, permitiendo así el fácil acceso a los mismos.

Por otro lado, la herramienta Spack no solo permite cargar y descargar paquetes de *software* ya instalados al entorno del usuario, sino también instalar nuevo software de su amplio catálogo. Spack facilita la gestión del *software* del supercomputador, ya que permite compilar cómodamente los paquetes de software aplicando optimizaciones específicas para el *hardware* del clúster y facilita la gestión de dependencias. Además, si los usuarios del supercomputador necesitan paquetes de *software* adicionales que no se encuentren disponibles en el catálogo de *software* del C3, Spack les permite instalar dicho *software* a nivel de usuario fácilmente, sin tener que compilarlo a mano.

VI. Casos de uso

A continuación se describe una muestra de las aplicaciones que se están ejecutando actualmente en el supercomputador C3, incluyendo varias disciplinas para mostrar su versatilidad para ejecutar cargas de HPC de propósito general y cargas de IA.

EpiGraph [17] es un simulador paralelo basado en agentes que realiza simulaciones estocásticas realistas de la propagación del virus COVID-19 en amplias extensiones geográficas. EpiGraph fue diseñado originalmente en el grupo de investigación de Arquitectura de Computadores de la UC3M y es una aplicación masivamente paralela que integra datos geográficos, climáticos, sociales, colectivos profesionales, patro-

Tabla I: Configuración de particiones.

Partición	Nodos	Default	Límite	Descripción / Notas
`cpu`	srv[01-45, 101-145]	Sí	2 horas	Cola estándar de corta duración.
`batch`	srv[01-45, 101-145]	No	2 días	Trabajos de producción estándar.
`large`	srv[01-45, 101-145]	No	14 días	Ejecuciones de larga duración.
`gpu`	srvgpu[01-05]	No	2 horas	Acceso rápido a nodos NVIDIA A40.
`gpu-batch`	srvgpu[01-05]	No	2 días	Producción en GPU.
`gpu-large`	srvgpu[01-05]	No	14 días	Cálculo intensivo en GPU.

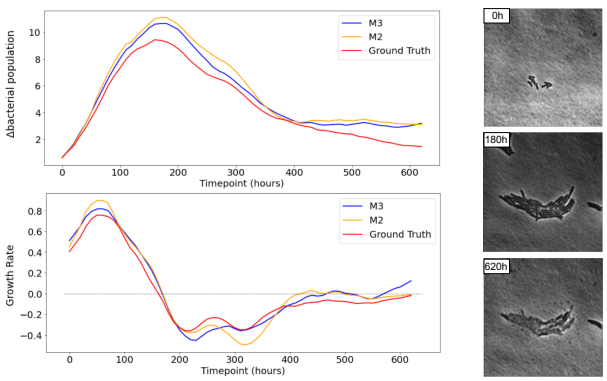


Fig. 9: Predicción de curva de crecimiento bacteriano para una secuencia de video de imágenes de microscopía.

nes de contacto, restricciones de movimiento, patrones de vacunación, etc. de las 642 ciudades europeas más grandes y una población total de 189.433.972 personas. Las pruebas realizadas en los últimos años se han entregado en diversos informes al Ministerio de Sanidad y al European Center for Disease Control y se han usado como entrada en la toma de decisiones sobre COVID-19.

El proyecto **PREFI-CM** [18] está desarrollando una simulación de la cascada de coagulación de la sangre con un enfoque multifidelidad que aprovecha la lenta difusión de las especies químicas para desacoplar la simulación de su transporte por flujo y su reacción química. El costo computacional de modelar la cascada de coagulación en sangre en movimiento es muy alto, pero este enfoque logra una reducción significativa en los requisitos computacionales manteniendo la precisión de nuestras simulaciones.

En el proyecto **ERA4TB** se han desarrollado técnicas de IA para predecir la curva de crecimiento de bacterias en placas *in vitro* con técnicas de fotografía "time lapse" usando aprendizaje profundo [19]. La Fig. 9 muestra un ejemplo de la salida de la predicción para un video completo de microscopía. El volumen de datos a analizar es considerable y requiere un largo tiempo de aprendizaje de días en un clúster de CPUs y se ha reducido a unos 20 minutos usando GPUs del C3.

El proyecto **HEEP**, Propulsión Electromagnética por Efecto Hall, se centra en la investigación y desarrollo en el campo de la propulsión eléctrica de plasma, específicamente en propulsores de efecto Hall (Hall Thrusters), que son un tipo de motor de iones utilizado para la propulsión espacial [20].

VII. Conclusiones

El diseño y despliegue del sistema de alto rendimiento del Centro de Computación Científica (C3) de la Universidad Carlos III de Madrid demuestra que la creación de una infraestructura de producción científica va más allá de la mera adquisición de hardware de vanguardia, a través de este caso de estudio de una infraestructura Tier-2.

Asimismo, la implementación de un ecosistema de monitorización integral, que combina herramientas de DCIM con sensores IoT y protocolos de seguridad automatizados, ha probado ser fundamental para garantizar la resiliencia del sistema. Esta infraestructura no solo protege la integridad física del equipamiento ante contingencias térmicas o eléctricas, sino que asegura una alta disponibilidad de los servicios. La capacidad de respuesta del centro se traduce en una mejora directa de la productividad científica, permitiendo a los grupos de investigación centrarse en la resolución de problemas complejos sin las limitaciones técnicas de entornos de computación menos optimizados.

Agradecimientos

Este trabajo ha sido parcialmente financiado por la Agencia Española de investigación a través del proyecto "New scalable I/O techniques for hybrid HPC and data-intensive workloads (SCIOT)" con referencia PID2022-138050NB-I00. Adicionalmente, este trabajo ha recibido apoyo del supercomputador de la Universidad Carlos III de Madrid del proyecto "Centro Para El Análisis Y Modelado De Sistemas Complejos En Ingeniería Y Biomedicina" con referencia EQC2021-007184-P, financiado por el Programa Estatal para Impulsar la Investigación Científico-Técnica de la Agencia Estatal de Investigación (AEI).

Referencias

[1] Ewa Deelman, Jack Dongarra, Bruce Hendrickson, Amanda Randles, Daniel Reed, Edward Seidel, and Katherine Yelick, "High-performance computing at a crossroads," *Science*, vol. 387, no. 6736, pp. 829–831, 2025.

[2] Robert McLay, Karl W Schulz, William L Barth, and Tommy Minyard, "Best practices for the deployment and management of production HPC clusters," in *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–11. IEEE, November 2011.

[3] Cosmin Petre, David Exposito-Singh, and Jesus Carretero, "Centro de Computación Científica (C3)," https://www.uc3m.es/cai/C3/, 2025, [Online; accessed 15-March-2026].
[4] Rajesh Pasupuleti, Ravi Vadapalli, Christopher Mader, and Victor J Milenkovic, "Apptainer-Based Containers for Legacy and Platform Dependent Machine Learning Applications on HPC Systems," in *2024 IEEE International Conference on Big Data (BigData)*. IEEE, 2024, pp. 6083–6091.
[5] D Bonacorsi, T Barrass, S Metson, J Andreeva, W Jank, N Sinanis, N Colino, P Garcia-Abia, JM Hernandez, FJ Rodriguez-Calonge, et al., "ROLE OF TIER-0, TIER-1 AND TIER-2 REGIONAL CENTRES IN CMS DC04," *presneted at CHEP04, Interlaken*, 2004.
[6] "TOP500: The list," https://top500.org/lists/top500/2025/11/, [Online; accessed 15-March-2026].
[7] Fabio Banchelli, Marta Garcia-Gasulla, Filippo Mantovani, Joan Vinyals, Josep Pocurull, David Vicente, Beatriz Eguzkitza, Flavio CC Galeazzo, Mario C Acosta, and Sergi Girona, "Introducing MareNostrum5: A European pre-exascale energy-efficient system designed to serve a broad spectrum of scientific workloads," *Future Generation Computer Systems*, p. 108125, 2025.
[8] Konstantin F Pilz, James Sanders, Robi Rahman, and Lennart Heim, "Trends in AI supercomputers," *arXiv preprint arXiv:2504.16026*, 2025.
[9] Muhammad AbuGhanem, "IBM quantum computers: evolution, performance, and future directions," *The Journal of Supercomputing*, vol. 81, no. 5, pp. 687, 2025.
[10] "IO500 list," https://io500.org/, [Online; accessed 15-March-2026].
[11] Liqid Inc., "Solution for composable AI infrastructure," https://www.liqid.com/, [Online; accessed 24-March-2026].
[12] Moises Levy and Jason O Hallstrom, "A new approach to data center infrastructure monitoring and management (DCIMM)," in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2017, pp. 1–6.
[13] T. Haynes and Editors Noveck, D., "Network file system (NFS) version 4 protocol," Tech. Rep., Internet Engineering Task Force, March 2015.
[14] Philip Schwan et al., "Lustre: Building a file system for 1000-node clusters," in *Proceedings of the 2003 Linux symposium*. Ottawa, Ontario, Canada, 2003, vol. 2003, pp. 380–386.
[15] Markus Geimer, Kenneth Hoste, and Robert McLay, "Modern scientific software management using easybuild and lmod," in *2014 First International Workshop on HPC User Support Tools*. IEEE, 2014, pp. 41–51.
[16] Todd Gamblin, Matthew LeGendre, Michael R Collette, Gregory L Lee, Adam Moody, Bronis R De Supinski, and Scott Futral, "The Spack package manager: bringing order to HPC software chaos," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*, 2015, pp. 1–12.
[17] Miguel Guzmán-Merino, Christian Durán, Maria-Cristina Marinescu, Concepción Delgado-Sanz, Diana Gomez-Barroso, Jesus Carretero, and David E Singh, "Data management in EpiGraph COVID-19 epidemic simulator," in *European Conference on Parallel Processing*. Springer, 2021, pp. 267–278.
[18] Manuel Guerrero-Hurtado, Manuel Garcia-Villalba, Alejandro Gonzalo, Pablo Martinez-Legazpi, Andrew M Kahn, Elliot McVeigh, Javier Bermejo, Juan C Del Alamo, and Oscar Flores, "Efficient multi-fidelity computation of blood coagulation under flow," *PLOS Computational Biology*, vol. 19, no. 10, pp. e1011583, 2023.
[19] Lara Visuña, Javier Garcia-Blas, Santiago Ferrer-Bazaga, Patricio Lopez-Exposito, and Jesus Carretero, "Automated monitoring of Mycobacterium population growth in time-lapse microscopy with Deep Learning pseudo-supervised algorithm," *Engineering Applications of Artificial Intelligence*, vol. 162, pp. 112503, 2025.
[20] Davide Poli and Eduardo Ahedo, "Towards Full-fluid Modelling of the Axial-azimuthal Hall Thruster Discharge," in *Proceedings of The 39th International Electric Propulsion Conference*. Electric Rocket Propulsion Society, 2025.
[21] William E. Allcock, Benjamin S. Allen, and et al., "Aurora: Architecting Argonne's First Exascale Supercomputer for Accelerated Scientific Discovery," 2025.
[22] Schneider Electric, "EcoStruxure IT Data Center Expert," https://www.se.com/es/es/product-range/61851-ecostruxure-it-data-center-expert/#overview, [Online; accessed 24-March-2026].

Evaluación de Mecanismos Semiasíncronos Deterministas y Adaptativos en Aprendizaje Federado

Víctor Hidalgo-Izquierdo[1], Carmen Carrión[2] y Blanca Caminero[2]

Resumen— **El aprendizaje automático ha evolucionado significativamente en los últimos años, expandiendo su aplicación a entornos de Internet de las Cosas (IoT), donde estas técnicas poseen un gran potencial en sectores como la mejora de diagnósticos médicos, el reconocimiento facial o la clasificación de imágenes. En este contexto, el Aprendizaje Federado Semiasíncrono (SAFL) surge como una extensión del Aprendizaje Federado (FL), diseñada para entornos con clientes heterogéneos y recursos limitados, lo que permite que los nodos participen en el entrenamiento sin necesidad de sincronizarse completamente en cada ronda, lo que mejora la eficiencia y la flexibilidad del sistema. Sin embargo, esta semiasincronía introduce nuevos desafíos, como la variabilidad en la contribución de los clientes y la inestabilidad en la convergencia, que requieren estrategias específicas de gestión y mitigación.**

Este trabajo evalúa diversas estrategias semiasíncronas, contrastando enfoques deterministas, basados en un grado de semiasincronía (M) estático, frente a una propuesta dinámica que se autoajusta según la disponibilidad y el comportamiento de los clientes. La contribución principal consiste en analizar la viabilidad de una estrategia basada en una heurística de ajuste dinámico, que permite al sistema adaptarse a un entorno heterogéneo de recursos limitados. Este estudio preliminar sienta las bases para futuras estrategias federadas adaptativas que incluyan posibles mecanismos de mitigación de la obsolescencia y otras ineficiencias derivadas de la semiasincronía, orientadas a infraestructuras distribuidas de bajo coste y escenarios IoT.

Palabras clave— **Aprendizaje federado, computación en la niebla, internet de las cosas, aprendizaje automático, semiasincronía.**

I. Introducción

En los últimos años, el crecimiento de los entornos IoT ha incrementado significativamente la heterogeneidad de los dispositivos participantes en sistemas distribuidos de aprendizaje automático, así como la cantidad de datos generados [1]. En este contexto, el paradigma de FL [2] permite entrenar modelos de forma distribuida, preservando la privacidad de los datos al mantenerlos en los dispositivos *edge* y al compartir únicamente las actualizaciones del modelo.

Sin embargo, la implementación tradicional de FL, basada en una sincronización estricta, resulta ineficiente en escenarios altamente heterogéneos, donde las diferencias en capacidad computacional y conectividad entre clientes generan el conocido efecto *straggler* [3]. Este fenómeno consiste en el retraso provocado por los dispositivos de menor capacidad, que actúan como un cuello de botella al obligar al resto a esperar a que se complete cada ronda de entrenamiento. Como consecuencia, además del incremento del tiempo total de entrenamiento, se produce una infrautilización de los recursos de los clientes más rápidos, que permanecen inactivos durante los periodos de espera. Para mitigar este problema, SAFL [4] engloba diferentes enfoques basados en la relajación de la sincronización estricta, entre los cuales se encuentran mecanismos de sincronización parcial controlados por el grado de semiasincronía M, que determina el número de clientes necesarios para realizar la agregación del modelo global [5].

La elección del parámetro M es determinante para el rendimiento del sistema. No obstante, los enfoques deterministas basados en valores fijos de M presentan limitaciones claras: requieren una exploración exhaustiva de configuraciones para adaptarse a distintos entornos y, además, resultan poco flexibles ante cambios en las capacidades computacionales y/o comunicativas de los dispositivos.

En este contexto, surge la necesidad de estrategias dinámicas que permitan ajustar automáticamente y de manera eficiente el grado de semiasincronía. No obstante, dichas estrategias deben diseñarse respetando la filosofía de FL, evitando que el servidor dependa de información sensible de los clientes. En particular, se pretende limitar el uso de métricas internas o de características privadas (p.ej., características de los datos locales o propiedades específicas del dispositivo).

El objetivo de este trabajo es analizar la viabilidad de estrategias de selección del grado de semiasincronía en SAFL, comparando un enfoque determinista con otro dinámico basado en heurísticas. En particular, los objetivos específicos son los siguientes:

- Analizar el impacto de la selección del grado de semiasincronía M sobre la eficiencia computacional del sistema.
- Evaluar el efecto de las estrategias deterministas y dinámicas sobre la convergencia del modelo global.
- Explorar la viabilidad de enfoques heurísticos que permitan ajustar M sin abandonar la filosofía agnóstica de FL.

El resto del trabajo se organiza de la siguiente manera. En la Sección II se presentan brevemente algunos trabajos relacionados. A continuación, en la Sec-

[1]Instituto de Investigación en Informática de Albacete, Universidad de Castilla-La Mancha, e-mail: victor.hidalgo@uclm.es.

[2]Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha, e-mail: {carmen.carrion, mariablanca.caminero}@uclm.es.

ción III, se describe el sistema diseñado e implementado. La Sección IV detalla el entorno experimental, incluyendo el modelo y los datos utilizados, y presenta los resultados obtenidos. Finalmente, la Sección V presenta las conclusiones del trabajo y plantea posibles líneas futuras de investigación.

II. Estado del Arte

En los últimos años, se ha observado un notable incremento en el número de trabajos centrados en SAFL, lo que refleja el creciente interés por abordar los desafíos derivados de la heterogeneidad en entornos distribuidos. Este auge responde a la necesidad de mejorar la eficiencia de FL en escenarios en los que los clientes presentan capacidades computacionales y condiciones de red dispares.

En términos generales, SAFL relaja la sincronización estricta propia de FL mediante mecanismos de agregación parcial, lo que permite al servidor actualizar el modelo sin esperar a todos los clientes. La Figura 1 ilustra una posible implementación en la que un servidor central recibe actualizaciones de los clientes de forma asíncrona y las almacena en un búfer de agregación. Cuando el número de modelos recibidos alcanza un umbral M, estos se utilizan para actualizar el modelo global, lo que reduce los tiempos de espera y mitiga el efecto *straggler* de los nodos más lentos.

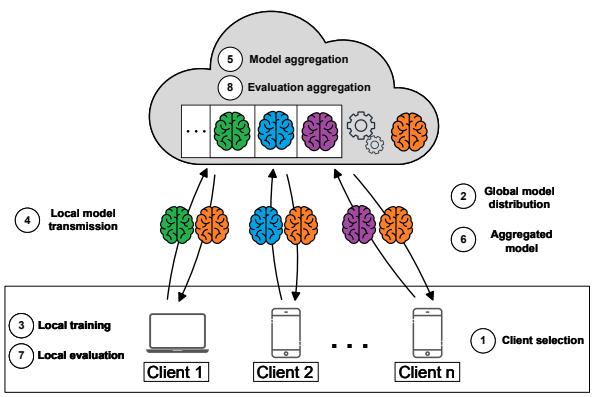


Fig. 1: Workflow of SAFL for IoT-edge environments.

A continuación, se describe el proceso general del sistema. Para formalizar el modelo, se define la siguiente notación: sea $\mathcal{N}$ el conjunto de clientes, donde $n \in \mathcal{N}$ denota un cliente, k el índice de ronda, w^k el modelo global en la ronda k, D_n el conjunto de datos locales del cliente n, y $D = \sum_{n\in\mathcal{N}} D_n$ el conjunto total de datos agregados del entorno federado. El sistema está coordinado por un servidor central, encargado de orquestar el proceso de entrenamiento, que incluye la selección de clientes, la agregación de actualizaciones y el mantenimiento del modelo global.

1. **Selección de clientes:** El servidor selecciona un subconjunto de clientes $T^k \subseteq \mathcal{N}$ para entrenamiento (y, opcionalmente, $E^k \subseteq \mathcal{N}$ para evaluación) en la ronda k, utilizando distintas políticas según la estrategia SAFL.
2. **Distribución del modelo global:** El modelo global actual w^k del servidor se envía a los clientes seleccionados.
3. **Entrenamiento local:** Cada cliente $n \in T^k$ entrena localmente su copia del modelo con sus datos privados D_n, obteniendo su modelo local actualizado w_n^{k+1}.
4. **Envío de modelos locales:** Los clientes envían al servidor los pesos actualizados de sus modelos, w_n^{k+1}.
5. **Agregación federada:** Las actualizaciones se almacenan en un búfer S. Cuando se alcanzan M contribuciones, el servidor agrega los modelos disponibles. No existe una única forma de realizar esta agregación. Un enfoque común consiste en aplicar Federated Averaging (FedAvg) sobre los modelos recibidos en cada ronda:

$$w^{k+1} \leftarrow \sum_{n\in S^k} \frac{|D_n|}{|D|} w_n^{k+1}, \tag{1}$$

donde S^k representa el subconjunto de clientes cuyas actualizaciones han sido consideradas para la agregación de la ronda k.
Sin embargo, esta agregación es inestable en entornos semiasíncronos. Para mitigar este efecto, suelen emplearse formulaciones más robustas que incorporan explícitamente el modelo previo w^k. Este enfoque permite que, en agregaciones con pocos clientes, el modelo global no se desestabilice:

$$w^{k+1} = \left(1 - \sum_{n\in S^k} \frac{|D_n|}{|D|}\right) w^k + \sum_{n\in S^k} \frac{|D_n|}{|D|} w_n^k, \tag{2}$$

donde el sumatorio recorre los clientes que participan en la ronda k, asignando a cada uno un peso proporcional al tamaño de sus datos respecto al total considerado en la agregación. De este modo, la contribución de cada cliente queda modulada por su volumen de datos, mientras que el término complementario $1 - \sum_{n\in S^k} \frac{|D_n|}{|D|}$ preserva la influencia del modelo global previo. Esto actúa como un mecanismo estabilizador que atenúa los cambios bruscos en los parámetros del modelo cuando el número de clientes activos es bajo. Nótese que en el caso de agregación síncrona, donde se dispone de la totalidad de los datos en cada ronda ($\sum_{n\in S^k} |D_n| = |D|$), el término asociado al modelo global previo desaparece, recuperándose la formulación equivalente a la Ecuación 1.
6. **Distribución y evaluación del modelo:** El modelo agregado se distribuye a los clientes para su evaluación (o se evalúa en el servidor), y las métricas obtenidas se agregan para monitorizar el rendimiento del sistema.

SAFL puede entenderse como un punto intermedio entre los enfoques síncrono y asíncrono. En particular, cuando se introduce el grado de semiasincronía

M, este paradigma puede interpretarse como un caso general en el que $M = 1$ corresponde a un esquema completamente asíncrono, mientras que $M = |\mathcal{N}|$ equivale a un enfoque totalmente síncrono. Valores intermedios de M permiten explorar distintos compromisos entre la eficiencia y la estabilidad del entrenamiento.

El grado de semiasincronía ha sido abordado en la literatura desde diversas perspectivas, especialmente en el contexto de los sistemas *resource-aware*. En estos enfoques, SAFL se modela como un sistema en el que el servidor determina un parámetro M que representa el número de clientes cuyas actualizaciones se agregan de forma síncrona en cada ronda. Mientras tanto, los clientes más lentos, limitados por recursos computacionales o de red, continúan procesando sus datos de forma asíncrona hasta la siguiente agregación. Ajustando el valor de M en función de las capacidades de los clientes, estos métodos buscan equilibrar la eficiencia del entrenamiento con la heterogeneidad del sistema, aumentando así la robustez frente al efecto *straggler*. En general, estos enfoques realizan la agregación una vez recibidas las actualizaciones de los primeros M clientes, donde este valor se determina mediante técnicas óptimas o heurísticas.

Entre los trabajos más relevantes, FedSA [3] propone un algoritmo eficiente para determinar el grado de sincronía óptimo M en función de la heterogeneidad del entorno y de la distribución de los datos, utilizando métricas como los tiempos de entrenamiento y los retardos de comunicación estimados. Por su parte, ASAFL [6] formula un problema de optimización que permite equilibrar la latencia y la precisión mediante la asignación de recursos y el cálculo óptimo de M en cada ronda. En una línea similar, FedSAP [5] ajusta dinámicamente el valor de M en función de los requisitos de eficiencia y rendimiento de la tarea, mientras que ASFL [7] introduce una estrategia en dos fases que combina la selección de clientes y el ajuste del grado de semiasincronía, basada en las latencias y apoyada en técnicas de aprendizaje por refuerzo profundo. Finalmente, SASAFL [8] adapta dinámicamente el grado de sincronización en función de la evolución de la función de pérdida global y de la presencia de clientes obsoletos (*stale*), lo que permite mitigar su impacto mediante mecanismos de agregación selectiva.

En este contexto, el uso de estrategias heurísticas, como la propuesta en este trabajo, se presenta como una alternativa viable que permite adaptarse de forma razonable a las condiciones del entorno manteniendo un bajo coste computacional y respetando la filosofía agnóstica de FL.

III. Estrategia dinámica semiasíncrona

En entornos IoT heterogéneos, la eficiencia de SAFL depende críticamente del grado de semiasincronía M, que determina cuántos clientes deben actualizar el modelo global antes de realizar la agregación. La elección de M influye directamente en el tiempo de entrenamiento y en la calidad del modelo, y su valor óptimo puede variar dinámicamente según la capacidad y la disponibilidad de los clientes. En este contexto, se introduce *FedMDyn*, un primer enfoque de estrategia dinámica para la selección de M, que estima adaptativamente este parámetro en cada ronda de entrenamiento, denotado como M_{Dyn}, buscando un equilibrio entre eficiencia y utilidad sin requerir información detallada sobre los clientes.

SAFL introduce un parámetro de control de sincronización M, que determina cuántas actualizaciones de clientes se requieren antes de realizar la agregación global. Este parámetro define un espectro continuo que va del entrenamiento totalmente asíncrono al totalmente síncrono, con múltiples configuraciones intermedias que equilibran rapidez y frescura de los datos, tal y como se ha descrito en la Sección II.

Sin embargo, la selección de un valor óptimo de M es compleja, ya que depende de factores dinámicos como la heterogeneidad de los clientes, la carga del sistema y la distribución de los datos, que no son directamente observables ni estáticos. Una exploración exhaustiva de este parámetro mediante múltiples ejecuciones del sistema resulta, en general, computacionalmente ineficiente.

Para abordar esta limitación, FedMDyn propone un enfoque basado en modelado estadístico y simulación del proceso semiasíncrono, que permite ajustar M_{Dyn} de forma adaptativa a partir de información agregada del comportamiento de los clientes, sin requerir conocimiento detallado del entorno.

A. Estrategia de agregación

La estrategia dinámica semiasíncrona FedMDyn se introduce como una capa de abstracción intermedia entre el servidor semiasíncrono y el algoritmo de agregación. En particular, FedMDyn no modifica la lógica interna del método de agregación, sino que adapta dinámicamente el grado de semiasincronía recalculando heurísticamente el valor de M en cada ronda de entrenamiento. Este enfoque permite ajustar el comportamiento del sistema en función del estado actual de los clientes y del progreso de la ejecución, lo que aporta mayor flexibilidad en entornos heterogéneos.

Antes de detallar el funcionamiento de FedMDyn, es importante describir el mecanismo de penalización sensible a la obsolescencia *(staleness-aware)* incorporado en el agregador semiasíncrono subyacente. Este mecanismo no es específico de FedMDyn, sino que forma parte del esquema de agregación considerado.

Para cada cliente n que envía su modelo local, se define un factor de penalización según la Ecuación 3:

$$\mu_n = e^{-\beta\left(\tau_n + \ln\left(1+\frac{b_n}{\gamma}\right)\right)}, \tag{3}$$

donde $\mu_n \in (0, 1]$ mide la confianza en su actualización: valores cercanos a 1 indican contribuciones recientes y consistentes, mientras que valores pequeños reflejan mayor obsolescencia o sesgo. En particular, el *staleness* $\tau_n = k - k_n$ mide el número de rondas transcurridas desde la última participación del cliente n, donde k denota la ronda global actual y

k_n la última ronda en la que dicho cliente contribuyó. Asimismo, b_n cuantifica la desviación acumulada del cliente respecto al resto del sistema y se define como:

$$b_n = \sum_{m \in \mathcal{N}} \text{máx}(0, k_n - k_m) \tag{4}$$

donde k_m denota la última ronda en la que el cliente m ha participado. En este contexto, b_n cuantifica la desviación acumulada del cliente respecto al resto del sistema en términos de rondas, sumando el desfase temporal de todos los clientes que se encuentran por detrás de n. El parámetro γ regula el impacto de esta desviación, mientras que $\beta \in (0, 1]$ controla la intensidad global de la penalización.

La combinación de τ_n y el término logarítmico permite integrar tanto la frescura de las actualizaciones como la heterogeneidad del cliente, penalizando suavemente pero de forma creciente a aquellos con mayor obsolescencia o mayor desviación. Además, la función exponencial garantiza que μ_n permanezca acotada en $(0, 1]$.

A partir de este factor, la contribución efectiva de cada cliente en la agregación viene dada por $\mu_n \frac{|D_n|}{|D|}$, de modo que $\sum_{n \in S^k} \mu_n \frac{|D_n|}{|D|}$ representa el peso total de las actualizaciones locales consideradas en cada iteración.

Bajo este esquema, un agregador semiasíncrono basado en FedAvg puede expresarse como:

$$w^{k+1} = \left(1 - \sum_{n \in S^k} \mu_n \frac{|D_n|}{|D|}\right) w^k + \sum_{n \in S^k} \mu_n \frac{|D_n|}{|D|} w_n^k, \tag{5}$$

donde el factor μ_n modula la contribución de cada cliente en función de su obsolescencia y desviación, manteniendo el equilibrio de pesos en la agregación. Este esquema es independiente de FedMDyn y puede integrarse en cualquier estrategia semiasíncrona.

Sobre esta base, FedMDyn actúa exclusivamente sobre la dinámica de participación de los clientes mediante la adaptación del parámetro M_{Dyn}, sin alterar el mecanismo de agregación descrito.

B. Grado de semiasincronía dinámico, M_{Dyn}

El enfoque se basa en una extensión semiasíncrona de FL, en la que el servidor no espera a que todos los clientes finalicen su entrenamiento, sino que define un valor objetivo M y realiza la agregación de parámetros una vez que recibe M actualizaciones. Esta base semiasíncrona, introducida en [9] como una modificación del framework síncrono Flower [10], permite la recepción y la agregación parcial de actualizaciones sin necesidad de una sincronización global estricta. Sobre este esquema, se implementa una versión semiasíncrona de FedAvg como estrategia determinista que, dado un valor fijo de M, controla el número de contribuciones necesarias para cada agregación parcial dentro del marco SAFL.

Sobre esta base, la estrategia FedMDyn introduce un mecanismo dinámico que permite ajustar el grado de semiasincronía del sistema antes de cada ronda de entrenamiento, específicamente durante la fase de configuración. Para ello, FedMDyn se apoya exclusivamente en información observable de la ejecución, como la velocidad de los clientes y el último estado conocido de su entrenamiento, manteniendo así un enfoque agnóstico respecto de sus características internas.

En este contexto, los principales parámetros de entrada considerados son: (i) la ronda global actual t (*server_round*), que permite calcular la obsolescencia (*staleness*); (ii) el registro de actualizaciones previas (*prev_updates*); (iii) el parámetro β, que controla la penalización por obsolescencia; y (iv) el parámetro γ, que regula la penalización asociada al sesgo.

Inicialmente, FedMDyn introduce una ronda de exploración ($k = 0$) en la que el entrenamiento se realiza de forma síncrona. Esta fase permite recopilar información sobre los tiempos de entrenamiento de los clientes. En rondas posteriores, el sistema utiliza y actualiza dicha información para estimar dinámicamente el grado de semiasincronía.

B.1 Estimación del comportamiento de clientes

Se modela la velocidad de entrenamiento de cada cliente mediante un doble suavizado exponencial [11], que permite capturar tanto el nivel como la tendencia a partir de observaciones históricas. El servidor mantiene un estado por cliente, que se actualiza únicamente cuando este participa en el entrenamiento. En la ronda inicial ($k = 0$), dicho estado se inicializa para todos los clientes, lo que requiere una ejecución síncrona previa para obtener una primera observación de cada cliente.

Sea $y_{n,k}$ el logaritmo del tiempo de entrenamiento reportado por el cliente n en la ronda $k-1$, observado en la ronda k. Para modelar su evolución temporal, se emplea el método de suavizado exponencial de Holt, que descompone la serie en dos componentes: el *nivel*, que representa el valor suavizado del proceso en la ronda k, y la *tendencia*, que captura la dirección y la magnitud del cambio entre rondas consecutivas.

El modelo de Holt se define mediante las Ecuaciones 6 y 7, donde $L_{n,k}$ denota el nivel y $T_{n,k}$ la tendencia:

$$L_{n,k} = \alpha\, y_{n,k} + (1 - \alpha)(L_{n,k-1} + T_{n,k-1}) \tag{6}$$

$$T_{n,k} = \delta(L_{n,k} - L_{n,k-1}) + (1 - \delta) T_{n,k-1} \tag{7}$$

Los parámetros del modelo se derivan a partir de un único grado de suavizado base s, de modo que $\alpha = \frac{2}{s+1}$ y $\delta = \frac{\alpha}{3}$. En este contexto, α controla el suavizado del nivel y δ el de la tendencia. Este diseño permite a FedMDyn capturar tanto el valor representativo del tiempo de entrenamiento como su evolución temporal, lo cual resulta clave para anticipar cambios en la disponibilidad de los clientes y ajustar proactivamente el grado de semiasincronía M_{Dyn}.

La predicción a una ronda vista del log-tiempo de entrenamiento se obtiene como:

$$\hat{y}_{n,k} = L_{n,k-1} + T_{n,k-1} \tag{8}$$

En este contexto, el error de predicción en la ronda k se define como la diferencia entre la observación real y la predicción realizada en la ronda anterior:

$$e_{n,k} = y_{n,k} - \hat{y}_{n,k} \tag{9}$$

A partir de este error, se estima la incertidumbre del modelo mediante una media móvil exponencial del error cuadrático:

$$\sigma^2_{n,k} = \lambda e^2_{n,k} + (1-\lambda)\sigma^2_{n,k-1} \tag{10}$$

donde $\lambda = \frac{\alpha}{2}$ controla la velocidad de adaptación de la estimación de la varianza.

Este esquema induce una distribución lognormal de los tiempos de entrenamiento en el dominio original, parametrizada por la media predictiva $\hat{y}_{n,k}$ y la varianza asociada $\sigma^2_{n,k}$. Esta caracterización probabilística permite a FedMDyn modelar el comportamiento esperado de los clientes y ajustar dinámicamente el grado de semiasincronía M_{Dyn} de forma informada.

B.2 Simulación de ronda para un M fijo

El valor heurístico M_{Dyn} se evalúa mediante un proceso de simulación estocástica que permite analizar su comportamiento en distintos escenarios de ejecución. En este proceso se realizan N simulaciones independientes, en las que los tiempos de entrenamiento de los clientes se generan a partir de distribuciones lognormales parametrizadas con las estimaciones de media y desviación estándar obtenidas mediante el modelo de Holt.

Este enfoque introduce variabilidad en los tiempos de ejecución, lo que permite capturar posibles retrasos e imprevistos inherentes a entornos distribuidos heterogéneos. De este modo, cada simulación representa una realización plausible del sistema.

Formalmente, el tiempo de entrenamiento simulado del cliente n se denota como $\hat{t}_n$ y se modela como:

$$\hat{t}_n \sim \text{LogNormal}(\hat{y}_n, \sigma^2_n) \tag{11}$$

donde $\hat{y}_n$ y σ^2_n corresponden a la media y varianza en escala logarítmica estimadas mediante el modelo de Holt. Para cada simulación, estos parámetros se fijan como una instantánea del estado del sistema en la ronda actual, de modo que permanecen constantes durante la simulación.

En cada simulación se reproduce una ronda semiasíncrona condicionada por el estado previo del sistema. En primer lugar, se ajustan los tiempos de los clientes que continúan su ejecución desde la ronda anterior como:

$$\tilde{t}_n = \text{máx}(0, \hat{t}_n - T^{prev}_{\text{máx}}) \tag{12}$$

donde $T^{prev}_{\text{máx}}$ representa la duración de la ronda anterior.

A partir de estos tiempos ajustados, se selecciona el conjunto de los M clientes con menor tiempo de finalización, que denotamos como C_M:

$$C_M = \text{argmin}_{|C|=M}\{\tilde{t}_n\} \tag{13}$$

La duración de la ronda se define entonces como el máximo tiempo dentro de este conjunto:

$$T^{round} = \max_{n \in C_M} \tilde{t}_n \tag{14}$$

Adicionalmente, en cada simulación se incorpora la penalización asociada a la contribución de cada cliente durante la agregación. Para ello, se emplea la misma formulación definida para la penalización de clientes, denotada como μ_n (Ec. 3). De este modo, la penalización se aplica de manera consistente durante el proceso de simulación, lo que afecta directamente la contribución efectiva de cada cliente al cálculo del rendimiento del sistema.

Finalmente, la utilidad asociada a cada valor de M se estima en cada simulación mediante la acumulación de las contribuciones penalizadas de los clientes seleccionados. En particular, se calcula una velocidad acumulada ponderada de la forma

$$\sum_{n \in C_M} \text{velocity}_n \cdot \mu_n \cdot \frac{|D_n|}{|D|}$$

donde μ_n representa la penalización del cliente y $\frac{|D_n|}{|D|}$ es un factor de normalización que refleja la importancia relativa del conjunto de datos de cada cliente.

De este modo, cada valor de M queda asociado, en una única simulación, a una medida de utilidad que combina el progreso efectivo del sistema con el impacto de la heterogeneidad de los clientes.

B.3 Selección de M_{Dyn} para la siguiente ronda

Para cada valor candidato de M, se realizan N simulaciones independientes del sistema. En cada simulación i, se obtiene una velocidad agregada del sistema v^i_M y un tiempo total de ejecución t^i_M, calculados según el procedimiento descrito en la sección anterior.

A partir de estas realizaciones se obtienen las estimaciones empíricas:

$$\bar{v}_M = \frac{1}{N}\sum_{i=1}^{N} v^i_M, \quad \bar{t}_M = \frac{1}{N}\sum_{i=1}^{N} t^i_M. \tag{15}$$

Dado que ambas métricas han sido normalizadas al intervalo $[0, 1]$, se define una puntuación basada en la media armónica:

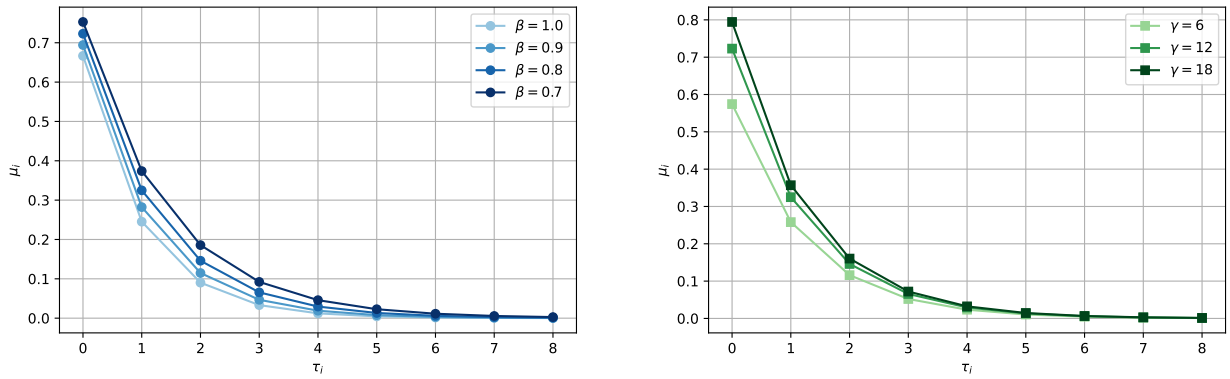
$$score(M) = \frac{2\,\bar{v}_M\,\bar{t}_M}{\bar{v}_M + \bar{t}_M}, \tag{16}$$

que favorece configuraciones que equilibran eficiencia agregada y tiempo de ejecución.

Finalmente, el sistema selecciona:

$$M_{Dyn} = \arg\max_M score(M), \tag{17}$$

maximizando el compromiso entre rendimiento global y latencia de la ronda.



(a) Efecto de β sobre μ_n con $\gamma = 12$ y $B = 6$.

(b) Efecto de γ sobre μ_n con $\beta = 0{,}8$ y $B = 6$.

Fig. 2: Análisis de sensibilidad de los parámetros β y γ sobre la función μ_n en función del *staleness* τ_n.

IV. Evaluación experimental

En esta sección se presentan los resultados experimentales con el objetivo de evaluar el comportamiento del sistema propuesto bajo distintos niveles de heterogeneidad y configuraciones de sincronización. A diferencia de enfoques tradicionales en FL, donde la evaluación se realiza en función del número de rondas, en este trabajo se adopta una métrica basada en tiempo. Esto se debe a que, en un entorno semiasíncrono, la duración de cada ronda varía en función del número de clientes participantes y de la dinámica de ejecución, por lo que el tiempo total de entrenamiento constituye una medida más representativa del rendimiento del sistema.

Además, se analiza el impacto de los principales hiperparámetros del sistema, en particular β y γ, así como el efecto del grado de semiasincronía sobre la convergencia del modelo. El estudio se estructura en distintos escenarios de heterogeneidad, lo que permite evaluar progresivamente la robustez del sistema ante clientes con tiempos de ejecución diversos. Asimismo, se comparan distintas configuraciones de sincronización, incluyendo esquemas síncronos y semiasíncronos con distintos valores de M, así como la variante adaptativa estudiada, con el fin de analizar tanto su eficiencia temporal como su estabilidad de convergencia.

A. Análisis preliminar de hiperparámetros

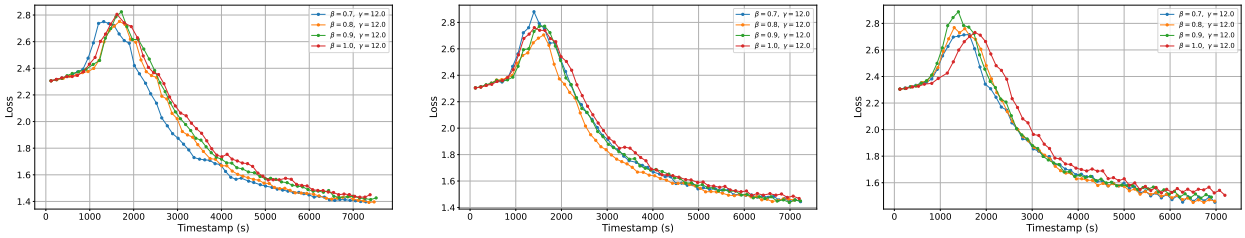
Antes de analizar en detalle el comportamiento del sistema bajo distintos escenarios de heterogeneidad, se realiza un estudio preliminar de los principales hiperparámetros, en particular de β y γ. El objetivo de este análisis es identificar rangos de valores adecuados que permitan un comportamiento estable del sistema y faciliten la interpretación de los resultados experimentales posteriores.

En la Ecuación 3, el parámetro β controla la intensidad global de la penalización aplicada a cada cliente, determinando la rapidez con la que μ_n decrece en función del retardo temporal τ_n y de la participación acumulada b_n. Por su parte, γ regula el equilibrio entre ambos términos, modulando la influencia relativa de la frecuencia de participación frente a la obsolescencia de las actualizaciones (ver Figura 2).

Para estudiar el efecto de β, se fija γ y se comparan distintos valores de β en los escenarios *slow1*, *slow2* y *slow3*, todos ellos con un grado de semiasincronía $M = 5$ (ver Figura 3). Los resultados muestran que β tiene un impacto significativo en el rendimiento del sistema, especialmente en presencia de heterogeneidad. En general, valores elevados de β tienden a degradar el comportamiento, al introducir una penalización excesiva sobre las contribuciones de los clientes. Este efecto se acentúa en entornos más heterogéneos, donde la variabilidad en los tiempos de entrenamiento y en la frecuencia de participación amplifica las consecuencias de una penalización agresiva.

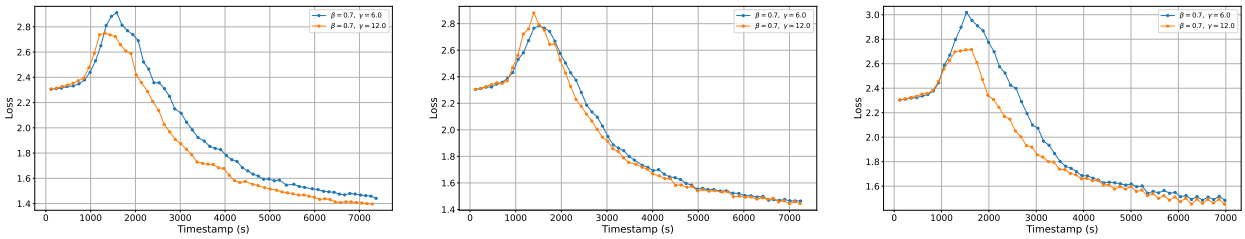
En los escenarios con mayor heterogeneidad, los valores más bajos de β, en particular 0,7 y 0,8, ofrecen los mejores resultados y muestran un comportamiento muy similar entre sí. Esta cercanía indica que, dentro de ese rango, el sistema mantiene una respuesta estable y robusta frente a las diferencias entre los clientes. En cambio, a medida que el entorno se aproxima a condiciones más homogéneas, las diferencias entre los valores de β se reducen y el efecto de este parámetro se vuelve menos determinante, especialmente cuando el grado de semiasincronía es alto. En consecuencia, para el resto de los experimentos se adoptan valores de β en torno a 0,7, ya que permiten una penalización suficiente sin comprometer la estabilidad del entrenamiento.

Una vez fijado $\beta = 0{,}7$, se analiza el efecto de γ comparando dos valores representativos, $\gamma = 6$ y $\gamma = 12$ (ver Figura 4). En este caso, los resultados muestran que los valores más altos de γ suelen ofrecer un comportamiento mejor y más estable, al reducir el peso de la frecuencia de participación y dar mayor protagonismo al retardo temporal. Este ajuste favorece una evolución más equilibrada del sistema, especialmente cuando se desea evitar que los clientes con una alta participación acumulada queden excesivamente penalizados. En general, $\gamma = 12$ tiende a proporcionar mejores resultados que $\gamma = 6$, tanto en términos de rendimiento como de estabilidad durante el entrenamiento. Aunque en los escenarios más heterogéneos las diferencias entre ambos valo-



(a) Estudio de valores de β: *slow0*. (b) Estudio de valores de β: *slow1*. (c) Estudio de valores de β: *slow2*.

Fig. 3: Estudio de los valores de β con γ fijo en entornos heterogéneos.



(a) Estudio de valores de γ: *slow0*. (b) Estudio de valores de γ: *slow1*. (c) Estudio de valores de γ: *slow2*.

Fig. 4: Estudio de los valores de β con γ fijo en entornos heterogéneos.

res se suavizan, la tendencia se mantiene y confirma que un γ más elevado resulta más adecuado para las configuraciones estudiadas.

En conjunto, este análisis preliminar permite acotar el espacio de búsqueda de hiperparámetros y establecer una configuración de referencia razonable para el resto de los experimentos. En particular, los resultados sugieren que un valor bajo de β, como 0,7, combinado con un γ relativamente alto, como 12, proporciona un equilibrio adecuado entre sensibilidad, estabilidad y capacidad de adaptación ante la heterogeneidad de los clientes. Esta elección se adopta como configuración de referencia para los experimentos posteriores.

B. Evaluación de FedMDyn

Tal y como se ha indicado previamente, se ha evaluado el comportamiento de FedMDyn utilizando $\beta = 0{,}7$ y $\gamma = 12$, con el objetivo de favorecer la estabilidad del sistema y simplificar el análisis comparativo.

En primer lugar, se presenta una comparativa directa entre FedMDyn y sus contrapartes deterministas ($M = 5, 6$ y 7) en distintos escenarios de heterogeneidad progresiva (Figura 5). De forma complementaria, la Figura 6 muestra la evolución del parámetro M_{Dyn} a lo largo de las rondas en cada uno de los escenarios considerados (a: *slow0*, b: *slow1*, c: *slow2*).

En el escenario *slow0* (Figura 5a), se observa que la estrategia dinámica se adapta de manera efectiva al entorno, en el que la sincronía es óptima debido a la homogeneidad de las velocidades de los clientes. FedMDyn reproduce este comportamiento, seleccionando sistemáticamente valores de M_{Dyn} cercanos al total de clientes, como se aprecia en la Figura 6a. Las ligeras fluctuaciones observadas pueden atribuirse a la variabilidad inherente al proceso de entrenamiento.

En el escenario *slow1* (Figura 5b), caracterizado por una heterogeneidad moderada, se observa un empeoramiento general de la velocidad de convergencia en todos los métodos. En este contexto, el enfoque completamente síncrono ($M = 7$) continúa ofreciendo los mejores resultados, tanto en el tiempo de convergencia como en el valor final de la función de pérdida. Este comportamiento puede explicarse por la mayor consistencia de las actualizaciones al incorporar toda la información disponible en cada ronda, a pesar del mayor coste temporal.

Por su parte, FedMDyn muestra una adaptación menos precisa que en el caso anterior. Tal y como refleja la Figura 6b, el valor de M_{Dyn} se mantiene mayoritariamente en torno a $M = 6$, lo que sugiere que el sistema tiende a alinearse con el subconjunto de clientes más rápidos. No obstante, se observan variaciones puntuales en M_{Dyn}, que pueden interpretarse como mecanismos de corrección ante acumulaciones de tiempo entre clientes. En determinadas rondas, estas diferencias pueden inducir fenómenos de *staleness*, lo que provoca ajustes temporales en la estrategia de agregación. Aunque este comportamiento puede parecer subóptimo en entornos estacionarios, puede resultar beneficioso en escenarios dinámicos, donde la disponibilidad o la capacidad de los clientes puede variar abruptamente (por ejemplo, debido a desconexiones o a escenarios de *multi-tenancy*).

Finalmente, en el escenario *slow2* (Figura 5c), con un mayor grado de heterogeneidad, se observan tendencias similares, aunque con una mejora relativa en la capacidad de adaptación de FedMDyn. En particular, el enfoque dinámico logra una convergencia más rápida que algunas configuraciones deterministas (notablemente $M = 6$), aunque no supera por completo al caso síncrono ($M = 7$) en todos los aspectos.

El análisis de la evolución de M_{Dyn} (Figura 6c) revela un comportamiento cíclico característico. Ini-

cialmente, el sistema selecciona valores de M elevados, próximos al número de clientes rápidos. Sin embargo, conforme se acumulan diferencias en los tiempos de finalización, debido a la ejecución reiterada de los clientes rápidos mientras los lentos aún completan sus iteraciones, FedMDyn incrementa temporalmente M_{Dyn} hasta alcanzar configuraciones cercanas a la sincronía. Este ajuste permite reabsorber el desfase acumulado. Posteriormente, el sistema vuelve a reducir M_{Dyn}, priorizando de nuevo a los clientes más rápidos, e incluso finaliza algunas rondas con valores bajos de M_{Dyn} para acelerar la convergencia. Este patrón refleja la capacidad del método para equilibrar dinámicamente la eficiencia temporal y la consistencia de las actualizaciones.

Para completar el análisis, se estudia la Figura 7, que muestra una comparativa directa de los tiempos totales de ejecución entre las distintas estrategias de semiasincronía consideradas (tres configuraciones deterministas y el enfoque dinámico FedMDyn), representadas mediante barras, en los escenarios de *slow0* a *slow2*.

Se observa que FedMDyn requiere menos tiempo total para completar las 60 rondas del experimento. Este comportamiento se explica por su capacidad de adaptación dinámica, que le permite realizar actualizaciones con un número menor de clientes cuando resulta conveniente, reduciendo así el coste computacional por ronda. Además, en las fases finales del entrenamiento, en las que algunas configuraciones pueden incurrir en sobreajuste, se realizan actualizaciones que aportan una mejora escasa pero mantienen un alto coste computacional. En este sentido, FedMDyn mitiga este efecto al evitar agregaciones innecesariamente costosas. De forma complementaria, la reducción del número de clientes participantes en determinadas rondas implica también un menor uso de la red, tanto en comunicación ascendente como en descendente.

Por otro lado, la Tabla I presenta una descomposición temporal detallada para el escenario *slow2*, mostrando el porcentaje medio de tiempo que cada configuración dedica a las fases de evaluación (*eval*), inactividad (*idle*) y ejecución (*exec*), junto con su desviación típica (promediado sobre todos los clientes). Cabe destacar que tanto las fases de evaluación como las de inactividad pueden considerarse tiempo no productivo para efectos de cómputo efectivo.

A partir de estos resultados, se observa que FedMDyn dedica, en promedio, una mayor proporción de su tiempo a la fase de evaluación que las estrategias deterministas. Sin embargo, este incremento se compensa con una reducción significativa del tiempo en estado *idle*, lo que indica una utilización más eficiente de los recursos disponibles. En particular, la estrategia dinámica minimiza los periodos de espera mediante una selección adaptativa de los clientes participantes en cada ronda.

Como consecuencia, FedMDyn alcanza el mayor porcentaje de tiempo dedicado a la ejecución efectiva (78.90 %), superando al resto de configuraciones evaluadas. Este resultado pone de manifiesto una mejor aprovechamiento del tiempo total disponible para el entrenamiento.

Config	Eval. (% ± %)	Idle (% ± %)	Exec. (% ± %)
sync5	13.98 ± 0.30	22.87 ± 1.32	68.64 ± 1.03
sync6	14.02 ± 0.08	11.27 ± 0.44	69.71 ± 0.81
sync7	13.84 ± 0.02	17.58 ± 0.19	70.93 ± 0.24
FedMDyn	16.16 ± 0.02	5.94 ± 0.30	78.90 ± 0.24

Tabla I: Descomposición del tiempo en porcentaje del tiempo medio total.

En cuanto a la variabilidad, FedMDyn presenta, en general, una desviación típica menor que la de otras estrategias, lo que sugiere un comportamiento más estable. La única excepción se observa en la fase *idle*, donde es ligeramente superado por el caso completamente síncrono ($M = 7$). No obstante, FedMDyn iguala o mejora la estabilidad en las fases de evaluación y ejecución, consolidando su robustez global.

V. Conclusiones

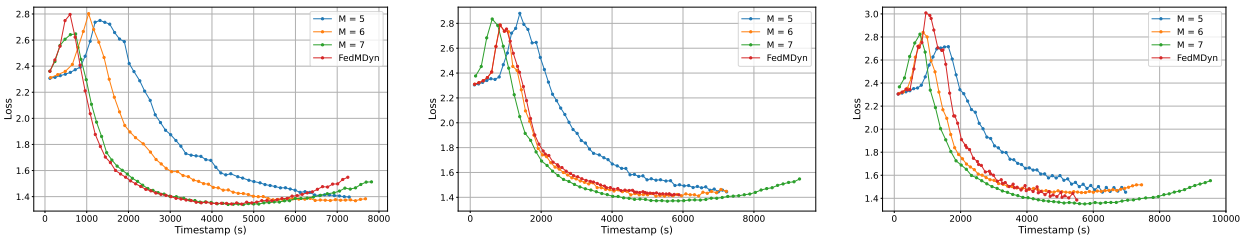
Hoy en día, el diseño de estrategias eficientes para el entrenamiento federado en entornos heterogéneos continúa siendo un desafío abierto, con una amplia literatura que aborda este problema más allá de la técnica presentada en este trabajo. En este contexto, se han analizado distintos esquemas de semiasincronía y se ha propuesto y evaluado FedMDyn como una estrategia dinámica para la selección del número de clientes a agregar en cada ronda, así como el estudio de sus hiperparámetros.

A partir de los resultados experimentales, se pueden extraer varias conclusiones relevantes. En primer lugar, se observa que un mayor tiempo total de ejecución no implica necesariamente una mejor convergencia. Por el contrario, resulta fundamental maximizar la eficiencia del uso del tiempo de cómputo, evitando actualizaciones redundantes o de bajo impacto, especialmente en fases avanzadas del entrenamiento.

En segundo lugar, la estrategia propuesta demuestra ser capaz de reducir de forma consistente el tiempo total de ejecución mediante una adaptación dinámica al estado del sistema. Este comportamiento introduce, no obstante, un compromiso entre la velocidad y la calidad de la convergencia, ya que las actualizaciones sobre subconjuntos de clientes pueden reducir la consistencia global del modelo en determinadas situaciones.

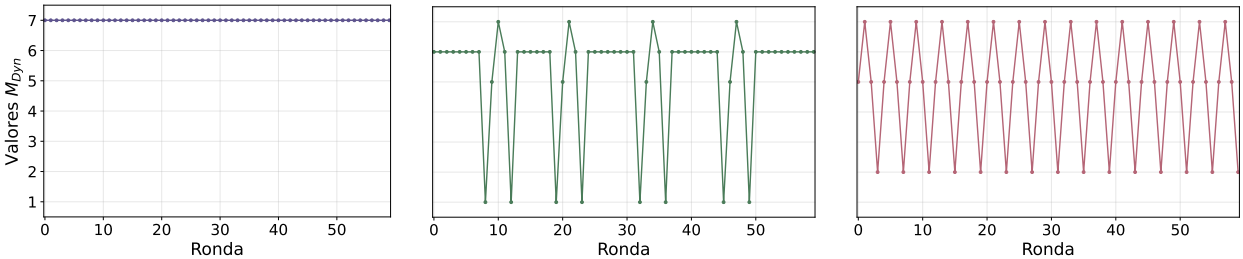
Asimismo, los resultados evidencian que el rendimiento de este tipo de enfoques depende en gran medida de la configuración de los hiperparámetros. En particular, parámetros como β y γ influyen de manera significativa en la estabilidad y la capacidad de adaptación del sistema, lo que hace recomendable realizar un análisis exploratorio previo para cada escenario de aplicación.

Adicionalmente, se ha observado que la dinámica de selección de clientes no solo impacta en la convergencia, sino también en la utilización efectiva de los recursos, lo que reduce los tiempos de inactividad y mejora el aprovechamiento global del sistema. Este



(a) Comparativa de métodos: *slow0*. (b) Comparativa de métodos: *slow1*. (c) Comparativa de métodos: *slow2*.

Fig. 5: Comparativa de métodos deterministas vs. dinámicos en entornos heterogéneos con $\beta = 0{,}7$ y $\gamma = 12$



(a) Selección de valores de M_{Dyn}: *slow0*. (b) Selección de valores de M_{Dyn}: *slow1*. (c) Selección de valores de M_{Dyn}: *slow2*.

Fig. 6: Comparativa de selección de valores de M del método determinista FedMDyn en entornos heterogéneos

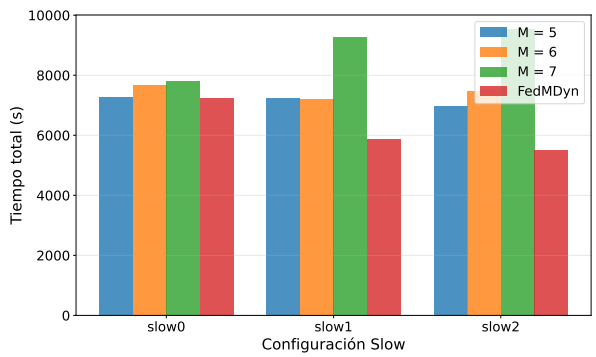


Fig. 7: Comparativa de tiempo total de ejecución en entornos heterogéneos

aspecto resulta especialmente relevante en entornos reales, donde la heterogeneidad de los recursos y la variabilidad temporal son factores predominantes.

Por otro lado, los experimentos sugieren que la adaptación dinámica resulta más beneficiosa en escenarios altamente heterogéneos o no estacionarios, mientras que en entornos estables y homogéneos, estrategias más simples pueden ser suficientes o incluso preferibles.

Como líneas de trabajo futuras, se proponen varias direcciones. En primer lugar, extender el mecanismo de selección de M incorporando información adicional del sistema, como la heterogeneidad de los datos entre clientes o métricas asociadas a la calidad de las actualizaciones (por ejemplo, normas de gradiente o contribución al descenso de la pérdida). En segundo lugar, integrar técnicas complementarias presentes en el estado del arte, tales como el ajuste adaptativo de la tasa de aprendizaje o las estrategias de selección de clientes basadas en criterios informados.

Finalmente, una línea prometedora consiste en el desarrollo de modelos predictivos que permitan anticipar el comportamiento temporal de los clientes (por ejemplo, tiempos de entrenamiento o de disponibilidad), lo que podría mejorar la toma de decisiones del sistema y reducir aún más los efectos de la asincronía.

Agradecimientos

Esta publicación forma parte del proyecto de I+D+i PID2024-158682OB-C32, financiado por el MICIU/AEI/10.13039/501100011033 y el FEDER/UE. También ha sido financiado por la beca 2025-GRIN-38312 otorgada por la Universidad de Castilla-La Mancha para Grupos de Investigación Consolidados. El primer autor también cuenta con un contrato predoctoral financiado por la Universidad de Castilla-La Mancha.

Declaración sobre IA asistida en la elaboración del manuscrito

Durante la elaboración de este trabajo, los autores utilizaron las herramientas ChatGPT de OpenAI y Perplexity de Perplexity AI con el fin de mejorar el lenguaje y la legibilidad. Tras el uso de estas herramientas, los autores revisaron y editaron el contenido según fue necesario y asumen la responsabilidad total del contenido publicado.

Referencias

[1] Alexandros Pantelopoulos and Nikolaos G. Bourbakis, "A survey on wearable sensor-based systems for health monitoring and prognosis," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 1, pp. 1–12, 2010.

[2] Omair Rashed Abdulwareth Almanifi, Chee-Onn Chow, Mau-Luen Tham, Joon Huang Chuah, and Jeevan Kanesan, "Communication and computation efficiency in federated learning: A survey," *Internet of Things*, vol. 22, pp. 100742, 2023.

[3] Qianpiao Ma, Yang Xu, Hongli Xu, Zhida Jiang, Liusheng Huang, and He Huang, "FedSA: A semi-asynchronous federated learning mechanism in heterogeneous edge computing," *IEEE JOURNAL ON SELEC-*

TED AREAS IN COMMUNICATIONS, vol. 39, no. 12, pp. 3654–3672, 2021.

[4] Zhixiong Chen, Wenqiang Yi, Hyundong Shin, and Arumugam Nallanathan, "Adaptive semi-asynchronous federated learning over wireless networks," *IEEE TRANSACTIONS ON COMMUNICATIONS*, vol. 73, no. 1, pp. 394–409, 2025.

[5] Peng Zhao, Zhuhua Liao, Yijiang Zhao, Jianbo Xu, and Aiping Yi, "A semi-asynchronous federated learning method integrating personalization and staleness awareness for traffic flow prediction in dynamic Internet of Vehicles," *Journal of Supercomputing*, vol. 81, June 2025.

[6] Zhixiong Chen, Wenqiang Yi, Hyundong Shin, and Arumugam Nallanathan, "Adaptive Semi-Asynchronous Federated Learning Over Wireless Networks," *IEEE TRANSACTIONS ON COMMUNICATIONS*, vol. 73, pp. 394–409, 2025.

[7] Jieling Yu, Ruiting Zhou, Chen Chen, Bo Li, and Fang Dong, "ASFL: Adaptive Semi-asynchronous Federated Learning for Balancing Model Accuracy and Total Latency in Mobile Edge Networks," in *Proceedings Of The 52ND International Conference On Parallel Processing, ICPP 2023*, 2023, pp. 443–451.

[8] Miri Yu, Jiheon Choi, Jaehyun Lee, and Sangyoon Oh, "Staleness aware semi-asynchronous federated learning," *Journal of Parallel and Distributed Computing*, vol. 193, pp. 104950, 2024.

[9] Víctor Hidalgo, Maria Blanca Caminero, and María del Carmen Carrión, "Despliegue de aprendizaje federado semiasíncrono sobre una infraestructura distribuida de bajo coste," in *Actas de las Jornadas SARTECO*, 2025.

[10] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Hei Li Kwing, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2023.

[11] Everette S. Gardner, "Exponential smoothing: The state of the art," *Journal of Forecasting*, 1985.

Framework de Edge AI para la Monitorización de Crisis Epilépticas Mediante Dispositivos Wearables en Microcontroladores

J. Navarro-Lázaro[1], M. Lupión[1], V. González-Ruiz[1], J.F. Sanjuan[1] y P.M. Ortigosa[1]

Resumen— **La monitorización fiable de las crisis epilépticas en entornos reales sigue siendo uno de los principales retos en la atención de la epilepsia, especialmente fuera del ámbito clínico, donde la electroencefalografía (EEG) convencional resulta poco práctica. Las soluciones *wearables* existentes ofrecen un soporte parcial mediante modalidades de sensores limitadas y, con frecuencia, dependen de teléfonos inteligentes para el procesamiento, lo que restringe la autonomía y la capacidad de respuesta en tiempo real. Además, las plataformas actuales carecen de soporte para la anotación clínica sincronizada y la generación de conjuntos de datos estructurados, elementos esenciales para el entrenamiento y la validación de algoritmos de detección en las fases iniciales de desarrollo. Para abordar estas limitaciones, presentamos un *framework* completo, de bajo consumo y extremo a extremo, basado en el Internet de las Cosas (IoT), orientado a la detección de crisis epilépticas mediante dispositivos *wearables*. El sistema integra la detección fisiológica multimodal: actividad electrodérmica (EDA), electromiografía (EMG) y temperatura (TEMP), con una arquitectura modular y en contenedores *Docker* para la ingesta de datos en tiempo real, el almacenamiento histórico y la anotación por expertos.**

A diferencia de soluciones anteriores, específicas para cada aplicación o vinculadas a *hardware* propietario, nuestra plataforma permite una implementación flexible y admite flujos de trabajo de anotación desacoplados, lo que facilita que los equipos clínicos y los investigadores construyan conjuntamente conjuntos de datos de alta calidad a partir de datos reales de pacientes. Partiendo de esta base, demostramos la implementación de un modelo de red neuronal convolucional (CNN) cuantizada capaz de realizar inferencias en menos de 15 milisegundos directamente en microcontroladores ESP32, lo que posibilita una detección autónoma y en tiempo real sin necesidad de procesamiento externo. Asimismo, se introduce un método de posprocesamiento para reducir los falsos negativos durante episodios de crisis prolongados. Validado con datos clínicos del Hospital Universitario Regional de Málaga, el *framework* propuesto ilustra una vía escalable y práctica hacia sistemas de detección de crisis portátiles de nueva generación que son a la vez inteligentes y autónomos.

Palabras clave— **IA embebida, IoT, Epilepsia, Redes Neuronales.**

I. Introducción

UNO de los principales retos en el tratamiento de la epilepsia es la ausencia de herramientas fiables para la monitorización continua de las crisis en entornos reales. Aunque la electroencefalografía (EEG) constituye el método de referencia clínica por su alta sensibilidad y especificidad, su uso fuera del ámbito hospitalario se ve limitado por el elevado coste, la necesidad de supervisión especializada y su escasa aceptabilidad social. Para hacer frente a estas dificultades, se están desarrollando dispositivos *wearables* de detección de crisis que permitan una monitorización autónoma, faciliten una intervención oportuna y mejoren los resultados del tratamiento.

Los dispositivos *wearables* se clasifican en cinco fases de validación que reflejan niveles crecientes de madurez clínica y complejidad [1]. La fase 0 emplea datos simulados; la fase 1 incorpora pacientes reales con registros anotados; la fase 2 requiere pruebas con más de 10 usuarios con garantías de seguridad; la fase 3 exige alertas en tiempo real validadas en entornos clínicos con más de 30 usuarios; y la fase 4 se centra en la usabilidad sin vídeo de referencia ni EEG.

Según esta escala, el presente estudio corresponde a la fase 2, al basarse en datos de pacientes reales, anotados clínicamente y recopilados en un entorno hospitalario, e incluir una validación con múltiples usuarios (12 pacientes en total, con un análisis detallado de tres de ellos). Sin embargo, aún no implica el despliegue a gran escala y en tiempo real con más de 30 usuarios que caracteriza la fase 3.

De acuerdo con la revisión sistemática de [2], de los 16 dispositivos de detección de crisis disponibles en el mercado, solo *Empatica* [3] y *EpiCare* [4] han obtenido la certificación de la fase 4, mientras que *Nightwatch* [5], *SPEAC* [6] y *EDDi* [7] permanecen en la fase 3. Estos dispositivos satisfacen los criterios para la generación de alertas en tiempo real y han sido validados en entornos clínicos o ambulatorios con más de 30 usuarios y eventos convulsivos verificados.

Los distintos tipos de crisis epilépticas se diferencian en sus manifestaciones fisiológicas, lo que determina la selección de sensores adecuada. Las crisis tónicas implican contracciones musculares sostenidas y suelen monitorizarse mediante electromiografía (EMG), actividad electrodérmica (EDA), temperatura cutánea (TEMP) y electrocardiogramas (ECG). Las crisis clónicas y mioclónicas, que conllevan espasmos musculares rítmicos o breves, se detectan mediante combinaciones similares, que con frecuencia incluyen también el EEG. Las crisis atónicas, caracterizadas por una pérdida repentina del tono muscular, se monitorizan mediante EEG y ECG, mientras que los tipos más complejos como las crisis focales discognitivas o las crisis autonómicas, requieren una detección multimodal (p. ej., EEG, ECG, EDA, espectroscopia de infrarrojo cercano, NIRS). Comprender estas relaciones entre los sensores y los tipos de crisis resulta clave para el desarrollo de sistemas de

[1]Universidad de Almería, 04120 Almería, España, e-mail: jnl941@inlumine.ual.es, {marcoslupion,vruiz,jsanjuan,ortigosa}@ual.es

detección eficaces y aplicables en la vida cotidiana.

La mayoría de los dispositivos comerciales se basan en la acelerometría (ACM), a menudo con un uso limitado de modalidades complementarias. Por ejemplo, *Empatica* combina ACM, EDA, TEMP y fotopletismografía (PPG); *Nightwatch* emplea ACM con PPG; y tanto *SPEAC* como *EDDi* se basan exclusivamente en EMG. Sin embargo, la sinergia entre EMG, EDA y TEMP permanece insuficientemente explorada, a pesar de su potencial para detectar una gama más amplia de tipos de crisis.

Por otra parte, a pesar de los avances alcanzados, los dispositivos actuales suelen carecer de capacidad de inferencia autónoma integrada debido a las limitaciones computacionales de los microcontroladores. La mayoría de los sistemas delegan el procesamiento en teléfonos inteligentes o *gateways*, lo que restringe la capacidad de respuesta en tiempo real y la autonomía. Este cuello de botella se agrava por la necesidad de modelos de detección multimodal basados en aprendizaje profundo, que demandan una elevada capacidad computacional. Para superarlo, se requieren estrategias de inferencia optimizadas para plataformas de bajo consumo y memoria limitada. En este trabajo, proponemos y evaluamos dichas técnicas con el objetivo de posibilitar la detección de crisis en tiempo real directamente en el dispositivo y en el borde.

En los estudios de fase 1, se requieren datos anotados que combinen señales de EEG y de sensores para el entrenamiento de los modelos de detección. Esta tarea se realiza habitualmente con dispositivos como Shimmer [8] o Biosignals Plux [9], que carecen de funcionalidades de base de datos personalizadas para la anotación sincronizada y la generación de conjuntos de datos.

Se han propuesto varios *frameworks* de IoT para la monitorización y anotación de señales biológicas. Awais et al. [10] presentan una configuración IoT inalámbrica optimizada para la clasificación de emociones mediante protocolos MAC de bajo consumo y *hardware* personalizado, aunque sin la robustez necesaria para su despliegue en entornos hospitalarios. Ham et al. [11] describen un sistema basado en Android construido sobre el *middleware* oneM2M, estrechamente acoplado a plataformas específicas y sin modularidad ni soporte para contenedores. Murhe et al. [12] utilizan *hardware* basado en Arduino, pero omiten la integración en la nube y el soporte de datos históricos. Nada et al. [13] integran datos de EEG y de dispositivos *wearables* en un *framework* que cumple con el RGPD y se centra en la precisión de la detección, aunque carece de ingestión modular o gestión de anotaciones. Estos sistemas suelen ser específicos para cada aplicación, no están contenedorizados y están vinculados a *stacks* fijos de *hardware* y *software*. Ninguno separa las anotaciones humanas de los datos de los sensores, admite el registro de datos históricos ni ofrece canalizaciones de ingestión modulares y flexibles.

Estas limitaciones ponen de manifiesto la necesidad de sistemas de bajo coste, eficientes energéticamente y autónomos que permitan una detección más amplia y el procesamiento en tiempo real en el borde. Este trabajo presenta un *framework* integral para la detección de crisis mediante dispositivos *wearables* que da respuesta a estas carencias. Incluye: (1) la capacidad de detectar distintos tipos de crisis a partir de señales multimodales (EMG, EDA, TEMP); (2) modelos de aprendizaje profundo optimizados para la inferencia en tiempo real en microcontroladores; y (3) una plataforma modular y en contenedores para la adquisición y visualización de datos con anotación sincronizada y almacenamiento histórico. Todo ello contribuye al avance de los sistemas portátiles de nueva generación para la monitorización de crisis en entornos reales.

Para validar el *framework* propuesto, se ha adoptado una metodología sistemática que abarca el diseño de *hardware*, la adquisición de datos, la optimización de modelos y su implementación. Las siguientes secciones desarrollan este proceso. La sección II describe la arquitectura del sistema, el protocolo de recopilación de datos clínicos, la estrategia de diseño de modelos y los métodos de posprocesamiento. La sección III presenta los resultados experimentales sobre los datos de los pacientes, incluyendo pruebas de rendimiento de las redes neuronales optimizadas en dispositivos integrados. Por último, la sección IV resume los hallazgos principales y el trabajo futuro orientado a mejorar la integración clínica y las capacidades de análisis en tiempo real.

II. Metodología

En esta sección se presenta la metodología seguida para desarrollar el *framework* propuesto basado en el IoT y sus modelos de IA asociados para la detección de crisis epilépticas. Se comienza con una descripción de la arquitectura del sistema, que incluye un dispositivo portátil que captura señales fisiológicas multimodales y las transmite a una plataforma de visualización basada en la nube. A continuación, se detalla el proceso de adquisición de datos, que incluye la colaboración con el Hospital Universitario Regional de Málaga para la obtención de registros de crisis sincronizados y anotados clínicamente. Se describe asimismo la arquitectura para el procesamiento de datos y la visualización en tiempo real, seguida de la estrategia de aumento de datos que incrementa la diversidad de las muestras de crisis mediante ventanas deslizantes temporales. Posteriormente, se expone el diseño y la optimización de los modelos de aprendizaje profundo, desde una red neuronal convolucional (CNN) con memoria a corto y largo plazo (LSTM) hasta una CNN ligera compatible con microcontroladores ESP32. Por último, se introduce una técnica de posprocesamiento para mejorar la robustez de la clasificación mediante la reducción de los falsos negativos durante episodios de crisis prolongados.

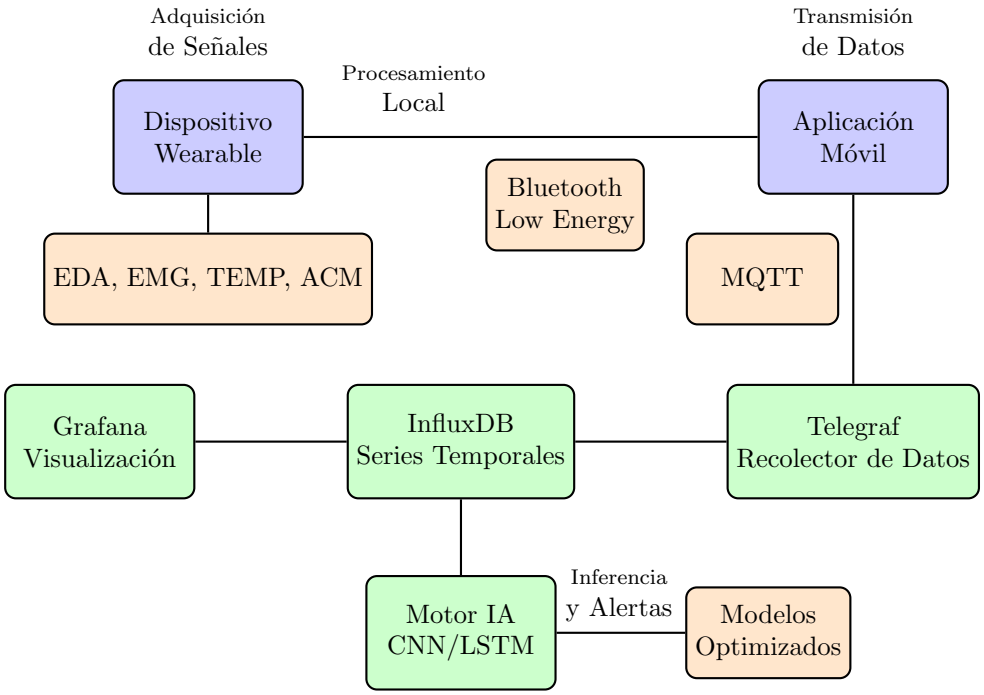


Fig. 1: Arquitectura inicial del wearable con procesamiento de IA centralizado.

A. Arquitectura del sistema

La arquitectura general del sistema se fundamenta en una solución distribuida basada en el IoT para la monitorización continua de las crisis epilépticas. La Figura 1 muestra el flujo de datos desde la adquisición de señales en los sensores hasta la inferencia en tiempo real.

El sistema incluye sensores multimodales (EDA, EMG, TEMP y ACM) para la extracción de señales fisiológicas de los pacientes en una unidad de vídeo-EEG. Estas señales se transmiten mediante *Bluetooth Low Energy* (BLE) a una aplicación móvil personalizada, que utiliza el protocolo *Message Queuing Telemetry Transport* (MQTT) para almacenar los datos en una base de datos temporal. El personal médico puede anotar los datos mediante el panel de control de Grafana, que presenta todas las señales de forma integrada. En paralelo, se aplica un enfoque de ventana deslizante para la generación de conjuntos de datos, con los que se entrenan los modelos de aprendizaje profundo sobre el conjunto anotado. Finalmente, dichos modelos se optimizan para su ejecución en un dispositivo integrado con capacidades computacionales limitadas.

B. Adquisición de datos

Dado que ningún conjunto de datos público incluye las señales fisiológicas multimodales requeridas, se llevó a cabo una fase de registro en el Hospital Universitario Regional de Málaga. Los pacientes fueron sometidos a una monitorización de vídeo-EEG de 24 horas, el método de referencia clínica para el registro de las crisis epilépticas. Además de los sistemas de EEG y ECG del hospital, se integraron sensores externos (Biosignals Plux para EMG, EDA y TEMP; Polar Verity Sense para ACM). Todas las secuencias se sincronizaron con las anotaciones clínicas de las crisis del hospital, lo que garantizó la alineación de los datos fisiológicos con los eventos convulsivos verificados, tal y como se ilustra en la Figura 2.

C. Sistema IoT para la visualización de datos

La arquitectura del sistema IoT mostrada en la Figura 1 proporciona un flujo de datos completo, desde la adquisición en los sensores hasta la visualización en tiempo real. Un dispositivo *wearable* recoge señales fisiológicas (EDA, EMG, TEMP, ACM) y las envía mediante BLE a una aplicación móvil, la cual reenvía los datos a un *broker* MQTT Mosquitto para una transmisión eficiente de alto rendimiento. Un agente Telegraf se suscribe al *topic* MQTT, almacena los datos en búfer y los escribe por lotes en una base de datos de series temporales InfluxDB, optimizando el rendimiento tanto de la red como de la base de datos.

Grafana se conecta a InfluxDB para proporcionar paneles con vistas superpuestas e individuales de las señales brutas, junto con métricas estadísticas (media, mínimo/máximo y desviación estándar) para cada señal. Las anotaciones clínicas se almacenan junto con los datos del electroencefalograma en archivos de formato europeo de datos (EDF), se analizan mediante un script de Python y se cargan a través de MQTT en una base de datos MariaDB. Un segundo script sincroniza estas anotaciones con Grafana a través de su interfaz de programación de aplicaciones (API), integrando las crisis clínicas en los paneles de control.

La Figura 3 muestra un ejemplo de visualización de múltiples crisis del Paciente 8: la sección izquierda presenta tres señales fisiológicas brutas superpuestas, y la sección derecha las muestra individualmente para facilitar estudios específicos. Se representan cinco

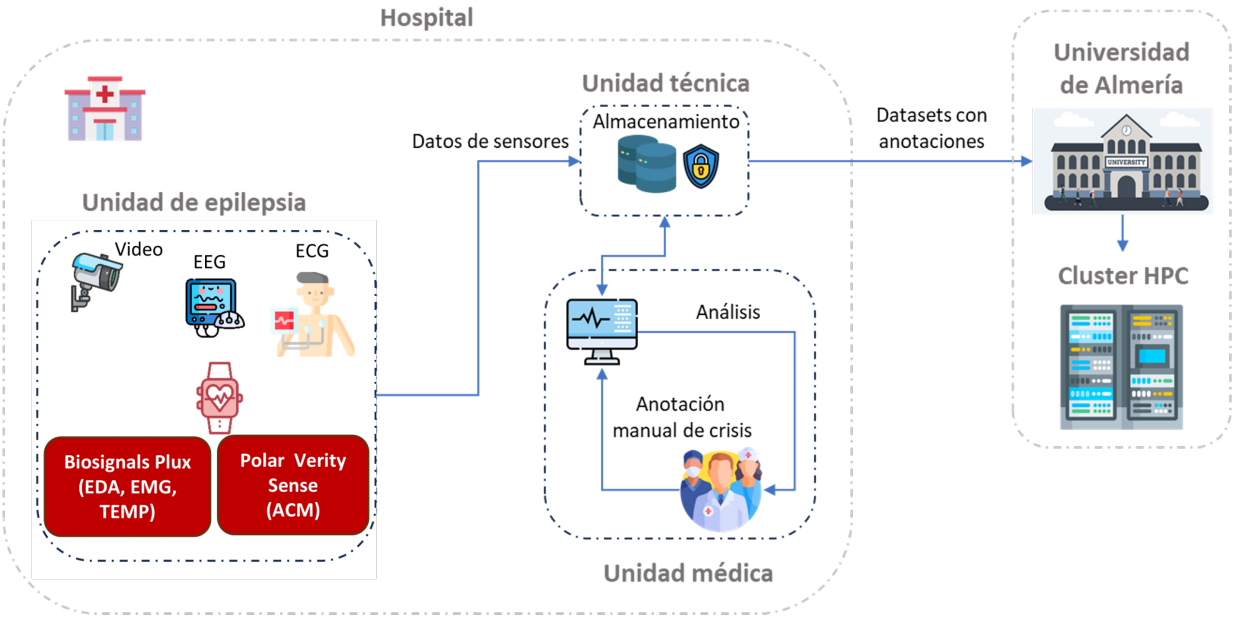


Fig. 2: Proceso de adquisición de datos fisiológicos en la unidad de vídeo-EEG.

crisis en áreas rectangulares verticales de color azul.

D. Aumento de datos

Para hacer frente al desequilibrio inherente entre clases en los conjuntos de datos de detección de crisis epilépticas, se implementó una estrategia de aumento de datos basada en ventanas deslizantes de longitud fija con desplazamientos temporales inferiores a dicha longitud. Esta técnica extrae múltiples segmentos solapados a partir de la misma señal original mediante desplazamientos temporales. Solo se conservan para el entrenamiento los segmentos que contienen más del 50 % de la duración de la crisis, salvo que formen parte de los datos sin desplazamiento, ya que es necesario incluir los datos de base sin crisis. Este enfoque incrementa significativamente la diversidad y la frecuencia de las muestras de crisis sin requerir una recopilación adicional de datos, mejorando así la robustez y la capacidad de generalización del modelo de clasificación.

La Figura 4 ilustra este proceso de aumento de datos. La señal EDA original (parte superior) contiene dos episodios convulsivos marcados en rojo mediante ventanas deslizantes de 30 segundos. Mediante desplazamientos temporales de 7,5 s y 15 s, se generan múltiples segmentos solapados a partir de los mismos datos de origen. Solo las ventanas cuyos desplazamientos temporales contienen más del 50 % de contenido convulsivo se conservan como muestras de entrenamiento válidas (representadas en verde), mientras que las ventanas con contenido convulsivo insuficiente se descartan (representadas en gris). Los datos sin desplazar, tanto de eventos sin crisis como de crisis, no se descartan.

En este contexto, puede surgir un posible problema de fuga de datos, dado que los desplazamientos temporales en ventanas consecutivas podrían capturar involuntariamente partes del conjunto de validación. Para mitigar este riesgo, el punto de separación entre los datos de entrenamiento y los de validación se estableció cuidadosamente de forma que ningún evento convulsivo del conjunto de validación quedase incluido en ninguna de las ventanas desplazadas utilizadas para el entrenamiento, lo que garantiza que no se añaden puntos de datos aumentados del conjunto de validación.

E. Arquitectura y optimización del modelo de IA

El desarrollo de un sistema de inferencia en el borde requirió importantes modificaciones arquitectónicas para adaptarse a las restricciones de recursos de los microcontroladores. Inicialmente, se diseñó una arquitectura híbrida CNN-LSTM por sus capacidades de aprendizaje temporal. Sin embargo, debido a las limitaciones de *hardware* y a la incompatibilidad de las capas LSTM con el *framework* TensorFlow Lite en microcontroladores, la arquitectura se rediseñó como una CNN más sencilla con tres capas Conv1D.

Las Tablas I y II muestran una comparación de ambas arquitecturas. El modelo CNN+LSTM procesa 7 características de entrada, incorporando 4 atributos temporales (día, hora, minuto y día de la semana) junto con las 3 señales fisiológicas (EDA, EMG, TEMP), mientras que el modelo CNN simplificado conserva únicamente las señales fisiológicas.

A pesar de la simplificación, la arquitectura CNN optimizada presenta un mayor número de parámetros (31.457) que el híbrido CNN+LSTM original (21.282), debido al incremento de la profundidad y del tamaño de las capas convolucionales y densas.

Se eliminaron las capas de normalización por lotes, dado que los datos están preescalados, lo que reduce la carga computacional y de memoria. Entre las optimizaciones más relevantes se incluyen la cuantización de `FLOAT32` a `INT8` y la compatibilidad con la biblioteca ESP-NN, que posibilitan un ahorro de memoria de $\times 4$ y una inferencia más rápida. Se consideró la poda (*pruning*), aunque con frecuen-



Fig. 3: Visualización de las crisis del Paciente 8 a través de Grafana.

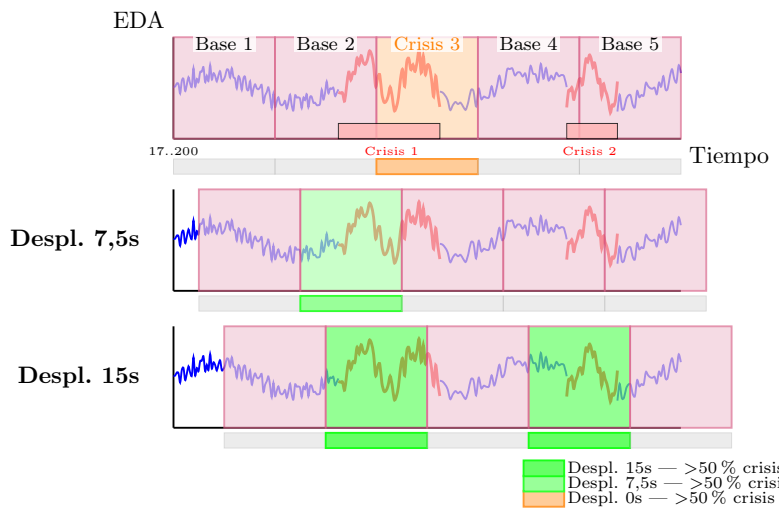


Fig. 4: Estrategia de aumento de datos mediante ventanas deslizantes con desplazamientos temporales.

cia genera dispersión irregular con una aceleración mínima en microcontroladores; la destilación del conocimiento (*knowledge distillation*) se plantea como una estrategia prometedora para trabajos futuros, que permitiría equilibrar aún más la precisión y la eficiencia [14].

F. Posprocesamiento

Para reducir los falsos negativos, se aplicó una estrategia de posprocesamiento temporal. Tal como se muestra en la Figura 5, las predicciones se agrupan en intervalos de 30 segundos. Si más del 50 % de las predicciones de un intervalo indican una crisis, el intervalo completo se reclasifica como crisis. Esta estrategia mitiga los falsos negativos aislados debidos a predicciones imprecisas en períodos de crisis naturalmente continuos y mejora la sensibilidad global del sistema.

III. Resultados

En esta sección se presentan los resultados obtenidos tras aplicar el *framework* propuesto a los datos fisiológicos recopilados en el Hospital Universitario Regional de Málaga. Se comienza detallando las características del conjunto de datos, incluidos los retos que plantea la escasez de datos de pacientes validados, y se describe la estrategia de división temporal empleada para garantizar un entrenamiento y una evaluación del modelo sin sesgos. A continuación, se destacan los resultados del modelo de aprendizaje profundo CNN+LSTM, tanto en el contexto general como en los modelos específicos por paciente. Sin embargo, debido a las limitaciones de *hardware*, únicamente fue posible implementar el modelo CNN en el dispositivo integrado final, el microcontrolador Arduino ESP32. Por ello, se presentan también los resultados de la cuantización y la validación de este modelo CNN optimizado, demostrando su eficacia para la inferencia en tiempo real con recursos limi-

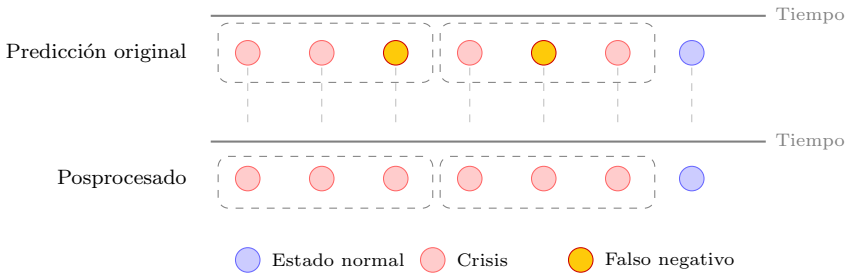


Fig. 5: Corrección de falsos negativos mediante posprocesamiento con ventana temporal.

Capa	Forma de Salida	Params.
InputLayer	(10, 7)	0
Conv1D	(10, 32)	1.152
MaxPooling1D	(5, 32)	0
BatchNormalization	(5, 32)	128
Dropout	(5, 32)	0
Conv1D	(5, 64)	6.208
MaxPooling1D	(2, 64)	0
BatchNormalization	(2, 64)	256
Dropout	(2, 64)	0
LSTM	(32)	12.416
Dropout	(32)	0
Dense	(32)	1.056
Dropout	(32)	0
Dense	(2)	66
Total		**21.282**

Tabla I: CNN+LSTM para señales multivariantes.

Capa	Forma de Salida	Params.
InputLayer	(10, 3)	0
Conv1D	(10, 32)	416
MaxPooling1D	(5, 32)	0
Conv1D	(5, 64)	6.208
MaxPooling1D	(2, 64)	0
Conv1D	(2, 128)	16.512
MaxPooling1D	(1, 128)	0
Flatten	(128)	0
Dense	(64)	8.256
Dense	(1)	65
Total		**31.457**

Tabla II: CNN optimizada para ESP32 sin capas LSTM.

tados.

A. Validación del modelo y dataset

El *framework* y los modelos IoT se validaron utilizando un subconjunto de datos de 12 pacientes monitorizados en el Hospital Universitario Regional de Málaga. El conjunto de datos de cada paciente incluye varias crisis epilépticas anotadas y períodos normales prolongados. Todas las señales fisiológicas (EDA, EMG, TEMP, ACM) se sincronizaron con los datos de EEG y se organizaron en carpetas específicas para cada paciente dentro del flujo de datos en contenedores.

La validación se centró en los pacientes 5, 8 y 9, al ser los únicos casos con datos EDA+EMG+TEMP médicamente validados y totalmente sincronizados. Se implementó una estrategia de separación temporal para garantizar que los datos de entrenamiento y de prueba permanecieran separados cronológicamente, preservando la secuencia natural de los eventos y evitando la fuga de datos, lo que resulta esencial en aplicaciones sanitarias en tiempo real.

La validación del modelo siguió dos estrategias: un modelo general y modelos específicos por paciente. El paciente P5, con una única crisis epiléptica anotada, contribuyó exclusivamente al entrenamiento del modelo general. Por el contrario, los pacientes P8 y P9, con múltiples episodios convulsivos, se emplearon para desarrollar y evaluar modelos individuales específicos por paciente, con lo que se validó la escalabilidad y la adaptabilidad del sistema a perfiles de datos de pacientes variables.

B. Resultados de aprendizaje profundo

La arquitectura CNN+LSTM muestra un rendimiento sólido y consistente con la combinación de sensores EDA+EMG+TEMP en el modelo general, tal y como recoge la Tabla III. Esta combinación alcanza una precisión de 0,9800 para la clase sin crisis (C0) y de 1,0000 para la clase con crisis (C1), con valores de Sensibilidad de 1,0000 y 0,7500 respectivamente, lo que evidencia una capacidad de detección equilibrada.

Los modelos específicos por paciente ponen de relieve la variabilidad en el rendimiento de la detección según las distintas combinaciones de sensores. El modelo del Paciente 8, que utiliza únicamente TEMP, alcanza una alta precisión y un Sensibilidad similares a los del modelo general, mientras que el modelo del Paciente 9, que emplea EDA+EMG, presenta una excelente precisión para la detección de eventos no convulsivos, pero muestra dificultades en la precisión de detección de crisis, lo que sugiere la presencia de posibles falsos positivos. Estos resultados subrayan la importancia de la calibración personalizada y la selección de sensores para mejorar la fiabilidad de la detección.

C. Despliegue de la CNN embebida

La CNN optimizada se implementó con éxito en microcontroladores ESP32, lo que demostró la capacidad del *framework* para el procesamiento autóno-

Modelo	Sensores	Precisión C0	Precisión C1	Sensibilidad C0	Sensibilidad C1
CNN+LSTM (General)	EDA+EMG+TEMP	0,9800	1,0000	1,0000	0,7500
CNN+LSTM (Paciente 8)	TEMP	0,9444	1,0000	1,0000	0,7500
CNN+LSTM (Paciente 9)	EDA+EMG	1,0000	0,0281	0,6581	1,0000

Tabla III: Mejores resultados del modelo general y de los modelos específicos por paciente (CNN+LSTM).

Modelo	Sensores	Precisión C0	Precisión C1	Sensibilidad C0	Sensibilidad C1
CNN+LSTM (General)	EDA+EMG+TEMP	0,9800	1,0000	1,0000	0,7500
CNN (Paciente 8)	TEMP	0,9255	1,0000	1,0000	0,7500
CNN+LSTM (Paciente 9)	EDA+EMG	1,0000	0,0281	0,6581	1,0000

Tabla IV: Mejores resultados del modelo general y de los modelos específicos por paciente, incluyendo el modelo CNN embebido.

Grupo	ESP32	Python	Desv.	Val. Real	Pred.
1	0,459	0,460	0,0039	Normal	Normal
2	0,461	0,461	0,0034	Normal	Normal
3	0,459	0,459	0,0025	Normal	Normal
4	0,567	0,574	0,0059	Crisis	Crisis
5	0,573	0,578	0,0078	Crisis	Crisis
6	0,556	0,556	0,0000	Crisis	Crisis
Total	–	–	<0,78 %	**Exactitud: 100 %**	

Tabla V: Validación del despliegue: consistencia entre la implementación en ESP32 y Python.

mo en el borde. A modo de demostración, el modelo se entrenó con los datos del Paciente 8, el sujeto con el mayor número de crisis epilépticas, utilizando los modelos CNN de mejor rendimiento.

La cuantización redujo el tamaño del modelo de 284,3 KB a 45,5 KB, lo que permitió su despliegue en dispositivos ESP32 con recursos limitados sin pérdida de precisión. Se codificaron de forma fija seis filas del conjunto de datos del Paciente 8 en el código de inferencia del ESP32 para la realización de pruebas comparativas, con lo que se confirmó la coherencia con la implementación en Python; no obstante, el modelo desplegado puede procesar señales arbitrarias en tiempo real.

La validación recogida en la Tabla V reveló desviaciones inferiores al 0,78 % entre los resultados obtenidos en el ESP32 y en Python, con tiempos de inferencia inferiores a 15 ms. Estos resultados confirman tanto la fiabilidad del modelo cuantizado como la eficiencia de las estrategias de optimización integradas en el *framework* IoT.

IV. Conclusiones y trabajo futuro

Este trabajo presenta un *framework* modular de IoT e IA para la monitorización de la salud mediante dispositivos *wearables* en el contexto de las crisis epilépticas, que abarca el proceso completo desde la adquisición de datos de los sensores hasta la inferencia en el borde. La arquitectura en contenedores integra MQTT, InfluxDB y Grafana para la monitorización en tiempo real y del historial, estableciendo un puente entre el manejo de datos clínicos y la implementación de IA embebida.

La validación con datos multimodales sincronizados de pacientes confirma la gestión fiable de conjuntos de datos sensibles con integridad temporal. Las redes neuronales optimizadas en microcontroladores ESP32 lograron tiempos de inferencia inferiores a 15 ms y una pérdida mínima de precisión tras la cuantización, lo que demuestra la viabilidad de la monitorización en el borde en tiempo real y de bajo consumo. La modularidad, la escalabilidad y la compatibilidad con los estándares de EEG del *framework* ponen de relieve su potencial para la atención sanitaria inteligente.

El estudio presenta como limitación el reducido tamaño del *dataset*, compuesto por 12 pacientes epilépticos, debido a las restricciones en la recopilación de datos sobre las crisis, aunque los resultados son coherentes con las validaciones propias de las fases iniciales. Los trabajos futuros incluyen la incorporación de nuevos pacientes para lograr una mayor solidez estadística y avanzar hacia la validación clínica de las fases 3 y 4.

El desarrollo posterior contempla la mejora de los análisis con visualizaciones avanzadas de Grafana, la ampliación de las modalidades de sensores y el perfeccionamiento de la integración clínica mediante interfaces intuitivas para el registro de crisis y la sincronización en tiempo real. Estas actuaciones tienen como objetivo facilitar una monitorización escalable, autónoma y multiusuario basada en IA de vanguardia.

Por último, se explorará la destilación del conocimiento (*knowledge distillation*) junto con la cuantización para el diseño de modelos compactos que mantengan el rendimiento predictivo reduciendo al mismo tiempo las exigencias computacionales en dispositivos con recursos limitados.

Agradecimientos

Este trabajo ha sido financiado por los proyectos de I+D+i PID2021-123278OB-I00 y PDC2022-133370-I00, con el número de referencia MCIN/AEI/10.13039/501100011033/, y con fondos del FEDER; así como por el Departamento de Informática de la Universidad de Almería.

Referencias

[1] Sándor Beniczky and Philippe Ryvlin, "Standards for testing and clinical validation of seizure detection devices," *Epilepsia*, vol. 59, no. suppl 1, pp. 9–13, 2018.

[2] Julie Verdru and Wim Van Paesschen, "Wearable seizure detection devices in refractory epilepsy," *Acta Neurol Belg*, vol. 120, pp. 1271–1281, 2020.

[3] Giulia Regalia, Francesco Onorati, Matteo Lai, Chiara Caborni, and Rosalind W. Picard, "Multimodal wrist-worn devices for seizure detection and advancing research: Focus on the empatica wristbands," *Epilepsy Res*, vol. 153, pp. 79–82, 2019.

[4] Pirgit Meritam, Philippe Ryvlin, and Sándor Beniczky, "User-based evaluation of applicability and usability of a wearable accelerometer device for detecting bilateral tonic-clonic seizures: a field study," *Epilepsia*, vol. 59, no. suppl 1, pp. 48–52, 2018.

[5] Johan Arends, Roland D. Thijs, Thea Gutter, Constantin Ungureanu, Pierre Cluitmans, Johannes van Dijk, Judith van Andel, Francis Tan, Al de Weerd, Ben Vledder, Wytske Hofstra, Richard Lazeron, Ghislaine van Thiel, Kit C. B. Roes, Frans Leijten, and Dutch Tele-Epilepsy Consortium, "Multimodal nocturnal seizure detection in a residential care setting: A long-term prospective trial," *Neurology*, vol. 91, no. 21, pp. e2010–e2019, 2018.

[6] Jonathan J. Halford, Michael R. Sperling, Dileep R. Nair, Dennis J. Dlugos, William O. Tatum, Jay Harvey, Jacqueline A. French, John R. Pollard, Edward Faught, Katherine H. Noe, Thomas R. Henry, Gina M. Jetter, Octavian V. Lie, Lola C. Morgan, Michael R. Girouard, Damon P. Cardenas, Luke E. Whitmire, and Jose E. Cavazos, "Detection of generalized tonic-clonic seizures using surface electromyographic monitoring," *Epilepsia*, vol. 58, no. 11, pp. 1861–1869, 2017.

[7] Sándor Beniczky, Isa Conradsen, Oliver Henning, Martin Fabricius, and Peter Wolf, "Automated real-time detection of tonic-clonic seizures using a wearable emg device," *Neurology*, vol. 90, no. 5, pp. e428–e434, 2018.

[8] Shimmer Sensing, "Shimmer wearable wireless sensors," 2025.

[9] PLUX Wireless Biosignals, "biosignalsplux: Wireless biometric signal acquisition," 2025.

[10] Muhammad Awais, Mohsin Raza, Nishant Singh, Kiran Bashir, Umar Manzoor, Saif Ul Islam, and Joel J. P. C. Rodrigues, "Lstm-based emotion detection using physiological signals: Iot framework for healthcare and distance learning in covid-19," *IEEE Internet of Things Journal*, vol. 8, no. 23, pp. 16863–16871, 2021.

[11] SeungMi Ham, HyeMin Lee, JaeHyun Lim, and Jeongwook Seo, "A negative emotion recognition system with internet of things-based multimodal biosignal data," *Electronics*, vol. 12, no. 20, pp. 4321, 2023.

[12] Varsha Murhe, Sandip Nagpure, and Dadas Anandrao Bhanudas, "Iot-convnet + lamb: A deep learning based emotion recognition framework using smart iot systems," *International Journal of Information Technology*, vol. 17, no. 5, pp. 2871–2876, 2025.

[13] Zaki Rashad et al., "An iot framework for emotion detection and behavior analysis with gdpr/hipaa compliance," *Americas PG*, vol. 3, no. 3, pp. 144–163, 2025.

[14] Minh Tri Lê, Pierre Wolinski, and Julyan Arbel, "Efficient neural networks for tiny machine learning: A comprehensive review," *ArXiv*, vol. abs/2311.11883, 2023.

Asignación de carga orientada a la eficiencia energética en entrenamiento distribuido heterogéneo

Daniel Suárez [1], Vicente Blanco [1], Pedro Toledo [1] y Francisco Almeida [1]

Resumen— **La heterogeneidad de dispositivos es una fuente importante de ineficiencia en el entrenamiento distribuido, ya que los puntos de sincronización obligan a los participantes más rápidos a esperar a los más lentos. Este efecto es especialmente visible en entornos de bajo consumo y de arquitecturas mixtas, donde las diferencias de rendimiento y comportamiento energético pueden ser sustanciales. Presentamos una estrategia ligera de reasignación de muestras en tiempo de ejecución que ajusta la cantidad de trabajo asignada a cada participante a partir de mediciones recogidas durante la ejecución. El método no es intrusivo y no requiere modificar ni el objetivo de entrenamiento ni el procedimiento de optimización.**

Evaluamos la estrategia propuesta en aprendizaje federado, *Federated Learning* (FL), entrenamiento distribuido paralelo por datos, *Distributed Data Parallel* (DDP), y un régimen híbrido FL+DDP con agregación por rondas sobre una plataforma heterogénea compuesta por dispositivos Arm, RISC-V y x86. En lugar de asumir una partición fija e igualitaria de la carga de entrenamiento, el sistema redistribuye las muestras entre participantes para ajustarse mejor a las capacidades observadas de cada dispositivo. Los resultados experimentales muestran que esta asignación dinámica reduce el desequilibrio en los puntos de sincronización y mejora la eficiencia global frente a una partición estática. Dependiendo del objetivo seleccionado, el método puede reducir el tiempo de ejecución o la energía consumida por los dispositivos manteniendo resultados de entrenamiento comparables bajo el mismo presupuesto de trabajo.

Estos resultados sugieren que la adaptación dinámica de la carga en tiempo de ejecución es una técnica de sistemas simple y eficaz para el entrenamiento distribuido heterogéneo.

Palabras clave— **entrenamiento distribuido, asignación de carga, computación heterogénea, eficiencia energética, aprendizaje federado.**

I. Introducción

El entrenamiento de un modelo compartido sobre múltiples máquinas es hoy habitual tanto en entornos *edge* como en infraestructuras de servidor, pero la heterogeneidad del *hardware* sigue dificultando una ejecución eficiente. Esto es especialmente cierto cuando los dispositivos difieren no solo en velocidad bruta, sino también en arquitectura, capacidad de memoria y comportamiento energético. Bajo estas condiciones, la sincronización se convierte en una fuente principal de ineficiencia: los participantes más rápidos esperan repetidamente a los más lentos, alargando el tiempo de ejecución y aumentando el consumo energético sin aportar progreso efectivo al entrenamiento.

[1]Dpto. de Ingeniería Informática y de Sistemas, Universidad de La Laguna, España, e-mail: {dsuarezl,vblanco,petode,falmeida}@ull.es.

Este problema aparece tanto en aprendizaje federado, *Federated Learning* (FL), como en entrenamiento distribuido paralelo por datos, *Distributed Data Parallel* (DDP), aunque de formas distintas. En aprendizaje federado, un servidor central distribuye el modelo actual, los clientes realizan actualizaciones locales y la agregación se produce al final de cada ronda [1], [2]. En entrenamiento distribuido paralelo por datos, los trabajadores procesan diferentes minilotes en paralelo y sincronizan gradientes tras cada paso del optimizador [3]. El primer escenario introduce esperas en los límites de ronda, mientras que el segundo puede sufrirlas con mucha más frecuencia porque la sincronización ocurre durante todo el entrenamiento. En ambos casos, la heterogeneidad del *hardware* hace que una distribución equilibrada de la carga termine traduciéndose en una ejecución desequilibrada.

Una respuesta habitual consiste en dividir la carga de entrenamiento de forma uniforme entre todos los participantes. Esta elección es simple y natural cuando los dispositivos son similares, pero resulta discutible cuando la plataforma mezcla procesadores con prestaciones y características energéticas muy distintas. Asignar la misma cantidad de trabajo a todos los dispositivos puede dejar infrautilizados los nodos más potentes, mientras que los más lentos o menos eficientes determinan el ritmo global del entrenamiento. El resultado es un aprovechamiento pobre de la plataforma y un gasto energético innecesario.

Diversas líneas de investigación han intentado reducir estos efectos. Entre las aproximaciones existentes se incluyen la participación selectiva, la ejecución asíncrona, modificaciones del proceso de optimización y adaptaciones del modelo para dispositivos más débiles [4]. Aunque efectivas en muchos escenarios, estas soluciones suelen alterar el propio proceso de aprendizaje o dependen de supuestos que no siempre resultan deseables en la práctica. Una alternativa es mantener intacto el algoritmo de aprendizaje y adaptar únicamente la carga asignada a cada participante durante la ejecución.

Este trabajo estudia precisamente esa alternativa. Proponemos un controlador ligero en tiempo de ejecución que reasigna muestras de entrenamiento entre participantes utilizando medidas recogidas en rondas previas. El método no modifica el modelo, el optimizador, la función de pérdida ni la semántica de sincronización. Su función es puramente operativa: decide cuánto trabajo debe recibir cada participante

en la siguiente ronda a partir del tiempo y la energía observados por muestra. De esta forma, el sistema puede reaccionar ante la heterogeneidad preservando el procedimiento original de entrenamiento.

Analizamos este mecanismo en tres regímenes de ejecución: aprendizaje federado, entrenamiento distribuido paralelo por datos puro y un escenario agrupado que combina ejecución local paralela por datos con agregación por rondas entre grupos. Aplicar el mismo principio de asignación a los tres regímenes permite estudiar cómo la granularidad de sincronización interactúa con la heterogeneidad y la redistribución de carga. También permite comparar directamente entrenamiento con sincronización laxa y estricta bajo las mismas condiciones de *hardware*.

La evaluación se realiza sobre una plataforma heterogénea de bajo consumo compuesta por dispositivos Arm, RISC-V y x86. Este entorno refleja la diversidad arquitectónica que cada vez aparece con más frecuencia en sistemas distribuidos de pequeña escala y entornos *edge*. En lugar de obligar a todos los dispositivos a procesar cargas idénticas, el controlador propuesto desplaza muestras hacia los participantes más favorables desde el punto de vista del objetivo seleccionado. Dependiendo de la configuración, el método puede priorizar una finalización más rápida o una menor energía consumida por los dispositivos.

Los resultados muestran que la reasignación dinámica de muestras es un mecanismo de sistemas eficaz para el entrenamiento distribuido heterogéneo. A lo largo de los regímenes estudiados, reduce el desequilibrio en los puntos de sincronización y mejora la eficiencia global frente a un reparto estático e igualitario, manteniendo una calidad de entrenamiento comparable bajo el mismo presupuesto total de trabajo. Estos resultados indican que es posible obtener mejoras significativas sin cambiar el método de aprendizaje subyacente, adaptando simplemente la distribución de la carga al comportamiento medido de los dispositivos participantes.

El resto del artículo se organiza de la siguiente forma. La Sección II revisa los trabajos relacionados con aprendizaje federado, entrenamiento distribuido y asignación de carga en sistemas heterogéneos. La Sección III describe la plataforma experimental, los regímenes de ejecución y el controlador adaptativo propuesto. La Sección IV presenta los resultados de tiempo, energía y calidad del modelo. Finalmente, la Sección V resume las principales conclusiones del trabajo.

II. Estado del arte

El aprendizaje federado ha sido ampliamente estudiado como una forma de entrenar modelos sobre clientes distribuidos sin centralizar los datos en bruto. Trabajos tempranos como FedAvg establecieron el flujo básico basado en rondas de entrenamiento local seguido de agregación en el servidor [2], mientras que estudios posteriores organizaron el área alrededor de retos clave como la eficiencia en comunicación, la heterogeneidad estadística, la privacidad y la variabilidad del sistema [1], [4]. Estos problemas se acentúan cuando los clientes difieren significativamente en capacidad de cómputo o comportamiento energético.

Una amplia línea de investigación se ha centrado en mitigar la heterogeneidad en entornos federados. Algunos métodos modifican el proceso de optimización para mejorar la robustez ante datos no independientes e idénticamente distribuidos, conocidos como datos no IID, y deriva entre clientes, como FedProx y SCAFFOLD [5], [6]. Otras aproximaciones adaptan la propia participación. Oort prioriza clientes combinando utilidad y velocidad del sistema [7], TiFL reduce los rezagados, habitualmente denominados *stragglers*, mediante planificación por niveles [8], y otros métodos seleccionan solo los clientes que probablemente terminarán dentro de un plazo objetivo [9]. Estas estrategias pueden reducir la duración de las rondas, pero normalmente lo hacen excluyendo o rebajando el peso de una parte de la plataforma disponible.

La sobrecarga de comunicación y ejecución también ha motivado estrategias alternativas de orquestación. Sistemas orientados a producción como Papaya reducen cuellos de botella entre rondas mediante una coordinación más flexible [10], mientras que los esquemas asíncronos relajan las barreras estrictas de ronda para que los clientes más tardíos no retrasen siempre el progreso global [11]. En paralelo, los métodos de compresión y cuantización reducen el coste de comunicación disminuyendo el tamaño de las actualizaciones transmitidas [12], [13], [14]. Estas contribuciones abordan cuellos de botella importantes, pero no se centran directamente en la asignación de carga como mecanismo dinámico para equilibrar dispositivos heterogéneos.

Ideas relacionadas también aparecen fuera del aprendizaje federado clásico. En clústeres heterogéneos, se han utilizado métodos de planificación y reequilibrado basados en búsqueda para optimizar conjuntamente rendimiento y energía [15]. Más recientemente, técnicas adaptativas de reducción de datos como AdaDP disminuyen el número de muestras procesadas por clientes más lentos para acelerar el entrenamiento [16]. El aprendizaje federado consciente de la energía también ha ganado atención, con trabajos que analizan compromisos entre radio y cómputo, participación descentralizada y costes energéticos a nivel de cliente [17], [18], [19]. Estos estudios muestran que la energía debe tratarse como un objetivo de sistemas de primer nivel y no como una métrica secundaria.

Nuestro trabajo se aproxima a esta línea orientada a sistemas, pero difiere tanto en alcance como en mecanismo. No modificamos el optimizador, el modelo ni la pérdida, y no dependemos de participación selectiva. En su lugar, mantenemos activos a todos los participantes y adaptamos únicamente el número de muestras asignadas a cada uno de una ronda a otra. La misma regla de control guiada por mediciones se evalúa en aprendizaje federado, entrenamiento distri-

buido paralelo por datos puro y un escenario híbrido agrupado. Esto nos permite estudiar la asignación de carga como principal mecanismo de control para reducir el desequilibrio de sincronización y la energía consumida por los dispositivos en plataformas heterogéneas.

III. Metodología

Esta sección resume la configuración experimental utilizada para evaluar la asignación adaptativa de carga en dispositivos heterogéneos. El estudio combina un flujo de entrenamiento común, medidas externas de tiempo y energía, una plataforma heterogénea de bajo consumo y tres regímenes de ejecución con distinto comportamiento de sincronización. En todos los experimentos, la participación es completa, la cantidad global de trabajo asignada por ronda es fija y el horizonte de entrenamiento se mantiene constante para que las comparaciones de tiempo y energía sean significativas.

A. Orquestación y medidas

Las ejecuciones federadas y agrupadas se coordinan con Flower [20]. En nuestra implementación, el servidor de Flower se ejecuta en un nodo dedicado y todos los clientes se conectan remotamente mediante el mismo protocolo. Esto proporciona un mecanismo común de coordinación para *hardware* heterogéneo y permite reutilizar el mismo código del modelo en distintos dispositivos. En cada ronda, el servidor difunde el modelo actual y la configuración de la ronda, los clientes realizan entrenamiento local con los datos asignados y el servidor agrega las actualizaciones recibidas mediante FedAvg [2], [4]. Extendemos este flujo de trabajo basado en rondas con un controlador de asignación que utiliza el tiempo y la energía observados en la ronda actual para decidir cuántas muestras debe procesar cada participante en la siguiente.

Los experimentos de DDP puro utilizan PyTorch Distributed en lugar de Flower. Para mantener la comparación alineada con el resto de regímenes, las ejecuciones de DDP con controlador se dividen en rondas lógicas. Estas rondas lógicas no alteran la semántica de DDP. Su función es únicamente crear puntos explícitos de planificación donde se puede difundir la siguiente asignación por *rank* antes de continuar el entrenamiento. Dentro de cada ronda lógica, los gradientes siguen sincronizándose en cada paso del optimizador.

El tiempo y la energía se miden con granularidad de ronda. El servidor registra la duración total de cada ronda desde el envío hasta la finalización de la agregación, y cada participante informa de su tiempo local transcurrido y del número de muestras procesadas. También recogemos la energía consumida por cada dispositivo en cada ronda. La latencia de comunicación no se emplea como señal directa de planificación porque es compartida, ruidosa y está fuertemente condicionada por los efectos de sincronización. En ejecución federada, la sobrecarga de red y agregación no refleja la capacidad intrínseca de un único dispositivo. En ejecución basada en DDP, la latencia colectiva viene determinada en gran medida por el *rank* más lento. Por este motivo, el controlador se apoya en medidas centradas en el coste computacional, mientras que los efectos de comunicación permanecen visibles en los resultados finales de tiempo y energía extremo a extremo.

Tabla I: Dispositivos utilizados en los experimentos.

Dispositivo	SoC/CPU	Arquitectura	Núcleos	RAM
Turing RK1 (4x)	RK3588	Armv8-A	8 (4A76+4A55)	32 GB LPDDR4x
Raspberry Pi CM4 (4x)	BCM2711	Armv8-A	4 (A72 1.5 GHz)	8 GB LPDDR4
Rock960 (1x)	RK3399	Armv8-A	2A72+4A53	4 GB LPDDR4
VisionFive 2 (1x)	JH7110	RISC-V RV64	4 (U74)	8 GB LPDDR4
MLLSE G2 Pro (1x)	Intel N150	x86-64	4 (Gracemont)	12 GB LPDDR5

La energía se mide externamente con AccelPowerCape [21], utilizando un host BeagleBone Black [22], sensores INA219 [23] y una canalización de recogida personalizada basada en `pmlib` [24], [25]. Las trazas de potencia se alinean con los límites de ronda reportados por el proceso de entrenamiento, y la energía por ronda se obtiene integrando la potencia sobre el intervalo correspondiente. Este procedimiento se aplica de forma consistente en todos los dispositivos, lo que hace que las medidas resultantes sean directamente comparables. Solo se incluye la energía consumida por los dispositivos. La infraestructura compartida, como el *switch* Ethernet, queda excluida.

B. Plataformas y organización de datos

La plataforma *hardware* abarca sistemas Arm, RISC-V y x86 con diferentes capacidades de cómputo y tamaños de memoria. La Tabla I resume los dispositivos utilizados en los experimentos. La plataforma incluye Turing RK1, Raspberry Pi CM4, Rock960, VisionFive 2 y un mini PC Intel N150. Todos los nodos están conectados mediante Ethernet cableado a un único *switch*, y todas las ejecuciones utilizan entrenamiento en CPU con multihilo local.

Para evitar que las transferencias en tiempo de ejecución se conviertan en un factor de confusión, el conjunto completo de entrenamiento se copia en todos los dispositivos antes de comenzar la ejecución. El servidor controla entonces el acceso lógico asignando índices de muestras a cada participante. Este diseño permite distinguir entre dos políticas lógicas de datos. En el caso estático, cada participante conserva el mismo fragmento durante toda la ejecución, lo que aproxima la suposición habitual en aprendizaje federado de datos locales fijos. En el caso adaptativo, el servidor puede cambiar tanto la cantidad de datos como los índices seleccionados de una ronda a otra, lo que asume un conjunto de datos compartido dentro de un único dominio de confianza. Dado que todos los dispositivos almacenan físicamente el conjunto de datos, la configuración no preserva la privacidad en el sentido más estricto, pero permite estudiar el impacto en sistemas de la partición estática y adaptativa sin añadir movimiento de datos durante la ejecución.

C. Regímenes de ejecución

Todos los experimentos comparten el mismo procedimiento supervisado de entrenamiento. En cada iteración, los datos se procesan en minilotes, el modelo realiza la pasada hacia delante, se calcula la pérdida, se obtienen los gradientes mediante retropropagación y el optimizador actualiza los parámetros. Por tanto, el procedimiento local de aprendizaje es común a todos los regímenes. Lo que cambia entre ellos es el patrón de sincronización.

Estudiamos tres modos de ejecución. En *aprendizaje federado*, cada dispositivo actúa como un cliente independiente y envía una actualización del modelo al servidor al final de cada ronda. La sincronización ocurre, por tanto, únicamente en los límites de ronda. En *entrenamiento distribuido paralelo por datos*, todos los dispositivos pertenecen al mismo trabajo DDP y los gradientes se sincronizan en cada paso del optimizador. En *FL+DDP híbrido*, los dispositivos se agrupan en equipos locales fijos de DDP; la sincronización ocurre en cada paso dentro de cada grupo, mientras que las actualizaciones de grupo se agregan en el servidor una vez por ronda. Este diseño híbrido combina ejecución local estrictamente sincronizada con coordinación entre grupos basada en rondas.

La participación es siempre completa. En FL y DDP, un participante corresponde a un único dispositivo. En el régimen híbrido, un participante corresponde a un grupo fijo de dispositivos. El presupuesto total de muestras por ronda es constante durante toda la ejecución. En FL y DDP, dicho presupuesto se distribuye entre todos los dispositivos. En la configuración híbrida, el presupuesto total se divide de forma uniforme entre grupos, y el controlador de asignación se aplica únicamente dentro de cada grupo. No se mueven muestras entre grupos en tiempo de ejecución.

D. Controlador adaptativo de asignación

El controlador de asignación se implementa como un módulo externo ligero que opera en los límites de ronda. No altera el modelo, el optimizador, la función de pérdida ni la semántica de sincronización del régimen subyacente. En su lugar, observa cuánto tiempo y energía necesitó cada participante en la ronda previa y utiliza esa información para decidir cuánto trabajo debe asignarse en la siguiente.

Al final de cada ronda, el servidor recoge de cada participante tres valores: tiempo transcurrido, energía consumida y número de muestras procesadas. A partir de estas medidas, el controlador estima el coste relativo de procesar una muestra en cada dispositivo y redistribuye el presupuesto fijo de muestras de la siguiente ronda. De este modo, la cantidad total de trabajo por ronda permanece constante, pero su reparto puede adaptarse al comportamiento observado de la plataforma.

Se evalúan dos políticas de asignación. La primera es una línea base de *reparto uniforme*, que divide el presupuesto de la ronda de la forma más equilibrada posible entre participantes. Esta configuración sirve como referencia en todos los regímenes y representa el caso habitual en el que se asume que todos los dispositivos deben procesar aproximadamente la misma cantidad de datos.

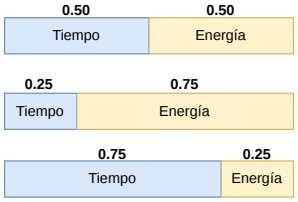


Fig. 1: Interpretación conceptual de los pesos de la política proporcional. Arriba, compromiso equilibrado entre tiempo y energía. En el centro, mayor énfasis en energía. Abajo, mayor énfasis en tiempo.

La segunda política es una *asignación proporcional ponderada*. En este caso, el trabajo no se reparte por igual, sino de manera inversamente proporcional al coste observado en la ronda anterior. Los dispositivos que muestran un menor coste por muestra reciben más trabajo en la siguiente ronda, mientras que aquellos que resultan más costosos reciben menos. La particularidad de esta política es que dicho coste puede construirse dando más importancia al tiempo de ejecución, a la energía consumida, o a un compromiso entre ambos.

Para ello, el controlador utiliza dos parámetros de peso, uno asociado al tiempo y otro a la energía. Cuando el peso del tiempo es mayor, la política favorece a los dispositivos que completan antes el trabajo asignado, incluso si no son los más eficientes energéticamente. Cuando el peso de la energía es mayor, la asignación tiende a desplazarse hacia los dispositivos que consumen menos por muestra procesada, aunque no necesariamente sean los más rápidos. Valores intermedios permiten explorar soluciones de compromiso entre ambos objetivos.

La Fig. 1 ilustra esta idea de forma conceptual. En la parte superior se representa un reparto equilibrado entre tiempo y energía. En la parte central, la política da mayor importancia a la energía, por lo que el criterio de decisión queda dominado por ese término. En la parte inferior ocurre lo contrario: el tiempo pasa a tener mayor peso y condiciona en mayor medida la asignación. En todos los casos, la suma de ambos pesos permanece normalizada, de modo que el controlador siempre trabaja con una combinación relativa de los dos objetivos y no con magnitudes absolutas.

Esta formulación hace posible utilizar el mismo controlador con distintos objetivos sin modificar el procedimiento de entrenamiento. Si se desea minimizar principalmente el tiempo total, puede emplearse una configuración dominada por el peso temporal. Si el objetivo prioritario es reducir la energía consumida por los dispositivos, puede darse mayor importancia al término energético. Entre ambos extremos aparecen configuraciones intermedias que permiten

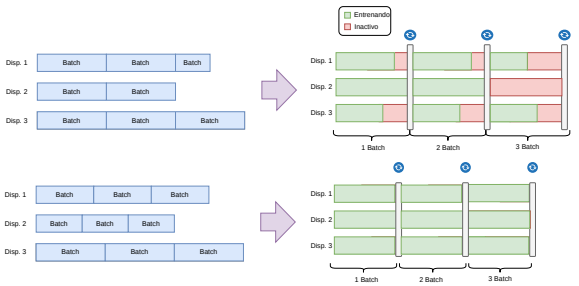


Fig. 2: Ajuste dinámico del tamaño de lote en DDP. Arriba, una asignación desigual de muestras con tamaño de lote fijo provoca tiempos de espera en los puntos de sincronización. Abajo, el ajuste dinámico del tamaño de lote mantiene alineado el número de pasos entre *ranks*, reduciendo los tiempos de espera mientras cada dispositivo procesa su carga asignada.

recorrer distintos puntos de compromiso entre rendimiento y eficiencia.

La primera ronda utiliza siempre reparto uniforme para que todos los participantes generen un perfil inicial de medidas. A partir de la segunda ronda, el controlador actualiza la asignación en cada límite de ronda. Dado que solo consume medidas agregadas por ronda y devuelve la siguiente asignación de muestras, añade poca complejidad de integración y permanece independiente de la implementación concreta del modelo.

E. Ajuste dinámico del tamaño de lote en DDP

Las asignaciones desiguales de muestras son directas en FL porque los clientes entrenan de forma independiente dentro de cada ronda. En ejecución basada en DDP, requieren mayor cuidado. Si todos los *ranks* utilizaran el mismo tamaño de lote local mientras procesan cantidades distintas de datos, algunos terminarían su trabajo útil antes que otros y pasarían el resto de la ronda esperando dentro de operaciones colectivas. Esto preservaría la corrección, pero ocultaría el beneficio de la asignación adaptativa detrás de la ineficiencia de sincronización.

La Fig. 2 ilustra este problema y la solución adoptada en este trabajo. En la parte superior se muestra el caso en el que una asignación desigual de muestras se combina con un tamaño de lote fijo. En esa situación, algunos *ranks* agotan antes su carga útil y permanecen en espera en los puntos de sincronización mientras esperan a los más lentos. En la parte inferior se representa el enfoque utilizado en nuestra propuesta, donde el tamaño de lote se ajusta dinámicamente para mantener alineado el número de pasos entre *ranks* y reducir así los tiempos muertos, sin renunciar a una distribución desigual del trabajo.

Para evitar este problema, las ejecuciones DDP con controlador emplean un esquema de tamaño de lote variable que mantiene alineado el número de pasos del optimizador entre los *ranks* activos y, al mismo tiempo, permite que cada *rank* procese exactamente su número asignado de muestras. En la práctica, esto significa que el sistema ajusta la forma en que las muestras locales se agrupan en lotes para que todos los *ranks* permanezcan sincronizados en número de pasos. Este mecanismo se utiliza tanto en DDP independiente como dentro de los grupos DDP del régimen híbrido.

Un segundo problema consiste en definir la entrada del controlador en DDP. El tiempo completo extremo a extremo por paso incluye esperas a *ranks* más lentos, lo que lo convierte en un mal indicador del rendimiento intrínseco de un dispositivo concreto. Por ello, el controlador utiliza señales ajustadas por espera a nivel de *rank* derivadas de la ejecución de la pasada hacia delante y de las ventanas energéticas correspondientes, mientras que el tiempo total de pared sigue registrándose para el informe final. Esto evita que el planificador reaccione principalmente a retrasos de sincronización en lugar de al coste local de cómputo.

F. Tarea, modelo y protocolo de evaluación

La evaluación se realiza sobre una tarea de clasificación de imágenes con MNIST [26]. Este conjunto de datos contiene 60 000 imágenes de entrenamiento y 10 000 de prueba en escala de grises, de tamaño 28×28, distribuidas en 10 clases. Para simplificar la ejecución sobre dispositivos heterogéneos y evitar transferencias durante el entrenamiento, el conjunto de datos se descarga una única vez, se almacena localmente en todos los nodos y se reutiliza en todas las ejecuciones. Las muestras de entrenamiento se particionan de forma determinista entre participantes, mientras que la evaluación se realiza sobre el conjunto común de prueba de MNIST.

El modelo utilizado en todos los experimentos es una red convolucional compacta, adecuada para esta tarea y suficientemente ligera para ejecutarse en la plataforma heterogénea considerada. La arquitectura consta de dos capas convolucionales con 16 y 32 canales, respectivamente, cada una seguida de activación ReLU, e incorpora etapas de *max pooling*. La salida convolucional se aplana y se conecta a una capa totalmente conectada de 64 neuronas, seguida de la capa de salida de 10 clases. Este diseño permite mantener un coste computacional moderado y, al mismo tiempo, resulta lo bastante representativo como para estudiar el efecto de la asignación adaptativa de carga sobre tiempo, energía y calidad final del modelo.

El entrenamiento utiliza pérdida de entropía cruzada y el optimizador Adam con tasa de aprendizaje 10^{-3} [27]. El tamaño de lote de referencia es 256. En los regímenes basados en DDP, cuando la asignación desigual de muestras lo requiere, este tamaño se ajusta dinámicamente según el mecanismo descrito anteriormente para mantener alineado el número de pasos sincronizados entre *ranks*. La calidad del modelo se reporta como pérdida media de prueba sobre MNIST, donde valores menores indican mejor rendimiento.

Todos los experimentos utilizan un horizonte fijo de entrenamiento equivalente a veinte rondas federadas. En FL y en la ejecución híbrida, esto corresponde directamente a veinte rondas de servidor. En DDP

con controlador, se introducen veinte rondas lógicas para crear puntos de actualización de la asignación entre fases de ejecución. La línea base de DDP puro se ejecuta de forma continua con el mismo número total de pasos del optimizador que la configuración DDP con controlador a lo largo de esas veinte rondas lógicas. Esto preserva la comparabilidad y asegura que cualquier diferencia en tiempo o energía proceda de la asignación de carga y no de una alteración de la semántica de DDP.

Las líneas base se definen por separado para cada régimen. En FL, la referencia es FedAvg síncrono con un reparto estático e igualitario del presupuesto de ronda entre clientes. En el régimen híbrido FL+DDP, la línea base utiliza los mismos grupos fijos, pero con reparto uniforme estándar por *rank* dentro de cada grupo, seguido de la misma agregación federada por rondas entre grupos. En DDP puro, la línea base es DDP continuo estándar sin puntos externos de planificación. Bajo este protocolo, la tarea MNIST con esta red convolucional se utiliza para analizar cómo la asignación adaptativa de muestras afecta al desequilibrio de sincronización, al tiempo de ejecución, a la energía consumida por los dispositivos y a la calidad final del modelo en sistemas heterogéneos.

IV. Resultados experimentales

A. Dinámica de asignación del controlador

En primer lugar, analizamos cómo el controlador redistribuye la carga de trabajo a lo largo de las rondas. La Fig. 3 muestra las asignaciones de muestras producidas en el régimen DDP para la tarea MNIST bajo tres valores representativos del peso energético E. En todos los casos, el presupuesto total de muestras por ronda se mantiene fijo en $S = 60\,000$, por lo que las diferencias entre curvas reflejan únicamente cómo el controlador reparte el trabajo entre dispositivos a partir de las medidas observadas en la ronda anterior.

Se observan dos patrones principales. En primer lugar, la mayor parte del reajuste ocurre en las primeras rondas. Tras la asignación inicial uniforme, el controlador desplaza rápidamente trabajo desde los dispositivos con mayor coste observado hacia aquellos con menor coste, y después converge a una distribución relativamente estable con ajustes menores en rondas posteriores. En segundo lugar, el parámetro E condiciona claramente la forma de la asignación. Cuando $E = 0$, la política favorece sobre todo a los dispositivos que minimizan el tiempo de ejecución, lo que produce una distribución más sesgada y concentra una fracción importante de las muestras en unos pocos nodos. A medida que E aumenta, la asignación se desplaza hacia dispositivos con mejor comportamiento energético por muestra, dando lugar a configuraciones de equilibrio distintas aunque el presupuesto total de trabajo permanezca constante.

Estas dinámicas concuerdan con los resultados agregados de tiempo y energía que se presentan en las subsecciones siguientes. Los pesos orientados al tiempo tienden a concentrar más trabajo en los dispositivos con mayor rendimiento, mientras que los pesos orientados a la energía favorecen aquellos con menor coste energético por muestra, aunque ello pueda modificar el equilibrio de sincronización y espera entre participantes.

B. Tiempo de ejecución

La Fig. 4 compara el tiempo total de ejecución para las distintas estrategias de asignación bajo los tres regímenes considerados: FL, DDP y FL+DDP híbrido. Las estrategias están indexadas por el peso energético E, de modo que $E = 0$ prioriza tiempo y valores mayores incrementan la importancia de la energía. Valores menores implican una finalización más rápida bajo el mismo horizonte de entrenamiento y el mismo presupuesto de muestras por ronda. La barra roja corresponde a la línea base: reparto uniforme en FL y ejecución DDP estándar en los regímenes DDP e híbrido.

En FL, la sincronización se produce una sola vez por ronda, por lo que los *stragglers* no bloquean el progreso en cada paso del optimizador. Aun así, la asignación adaptativa reduce el tiempo total al disminuir la dispersión entre tiempos de finalización de los clientes y, con ello, el tiempo ocioso del servidor en la barrera de ronda. En esta tarea, las configuraciones con pesos bajos o intermedios de energía son muy parecidas entre sí, mientras que $E = 1$ empeora el tiempo total respecto a las mejores opciones, al desplazar más trabajo hacia dispositivos energéticamente favorables pero no necesariamente más rápidos. La línea base queda claramente por encima del resto.

En DDP, la sincronización ocurre en cada paso del optimizador, de modo que la línea base amplifica con mucha más fuerza el efecto de los *stragglers*. Cualquier dispositivo más lento retrasa todas las sincronizaciones y aumenta el tiempo ocioso acumulado. Las estrategias adaptativas reducen de forma muy notable el tiempo total al asignar menos muestras a los nodos más lentos y más trabajo a los de mayor rendimiento. Este efecto es más pronunciado que en FL precisamente por la mayor frecuencia de sincronización. En los resultados de MNIST, los valores bajos de E son los que producen los menores tiempos, mientras que $E = 1$ vuelve a introducir parte del desequilibrio al priorizar más la energía que la velocidad.

El régimen híbrido FL+DDP presenta un comportamiento intermedio. Dentro de cada grupo local, DDP introduce sincronización a nivel de paso, mientras que el servidor impone además una barrera entre grupos al final de cada ronda. La asignación adaptativa reduce el tiempo respecto a la línea base al mejorar el equilibrio dentro de cada grupo y también entre grupos. De nuevo, configuraciones con excesivo énfasis en la energía pueden degradar parcialmente el resultado si favorecen dispositivos más eficientes pero menos rápidos. Aun así, todas las configuraciones adaptativas superan claramente a la línea base.

En conjunto, la Fig. 4 muestra un compromiso cla-

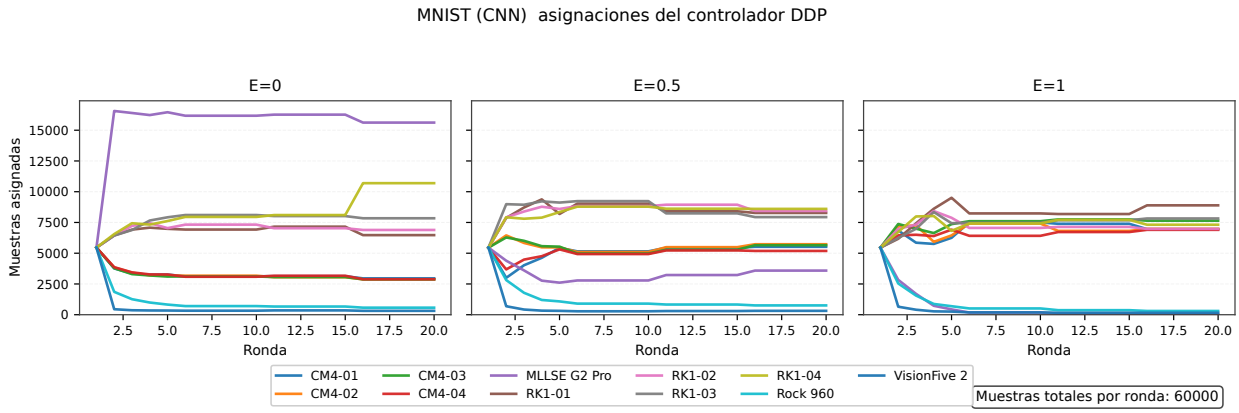


Fig. 3: Asignaciones del controlador en DDP para MNIST a lo largo de 20 rondas. Cada panel corresponde a un valor del peso energético E y muestra el número de muestras asignadas a cada dispositivo en cada ronda bajo un presupuesto fijo de $S = 60\,000$ muestras por ronda.

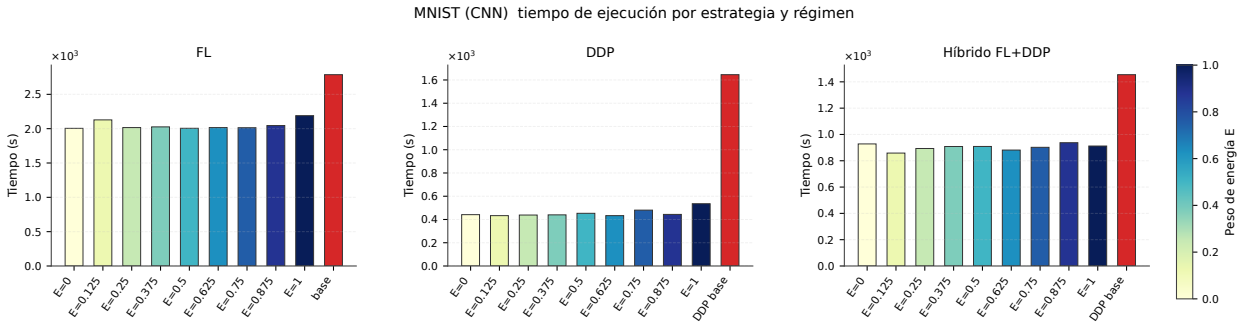


Fig. 4: Tiempo total de ejecución para MNIST en los tres regímenes evaluados. Las estrategias están indexadas por el peso energético E. La barra roja representa la línea base: reparto uniforme en FL y DDP estándar en DDP y FL+DDP híbrido. Valores menores son mejores.

ro entre tiempo y energía gobernado por E. Los valores pequeños de E preservan la mayor parte de la mejora temporal, mientras que los valores altos pueden reintroducir desequilibrio y espera en sincronización.

C. Consumo energético

La Fig. 5 muestra la energía total consumida por los dispositivos para completar el mismo horizonte fijo de entrenamiento bajo las mismas estrategias y regímenes. De nuevo, valores menores indican mejor eficiencia energética, y la barra roja representa la línea base correspondiente a cada régimen.

La reasignación adaptativa reduce el consumo energético total en los tres regímenes al disminuir el tiempo ocioso y desplazar trabajo desde dispositivos menos favorables hacia otros con mejor coste observado por muestra. Sin embargo, minimizar tiempo y minimizar energía no conduce exactamente al mismo punto de operación.

En FL, la reducción de energía proviene sobre todo de reducir la dispersión temporal entre clientes y, con ello, el tiempo de espera del servidor al final de cada ronda. En los resultados de MNIST, los menores consumos aparecen con valores intermedios de E, mientras que $E = 1$ incrementa la energía frente a las mejores configuraciones al favorecer en exceso el criterio energético local, lo que puede empeorar el balance global de tiempos. La línea base vuelve a ser claramente peor.

En DDP, la espera inducida por *stragglers* resulta especialmente costosa desde el punto de vista energético porque los nodos más rápidos permanecen activos mientras están detenidos en operaciones colectivas. Al reducir el desequilibrio entre *ranks*, las estrategias adaptativas disminuyen tanto el tiempo como la energía desaprovechada en sincronización. En esta tarea, los mejores resultados energéticos se obtienen con valores intermedios de E, mientras que los extremos no son necesariamente óptimos. Esto confirma que la solución con menor energía no coincide siempre con la de menor tiempo.

El régimen híbrido vuelve a situarse entre FL y DDP. La asignación adaptativa reduce la energía al mejorar el equilibrio dentro de los grupos DDP y al mismo tiempo redistribuir el trabajo entre grupos de manera más favorable. También aquí se observa que un énfasis excesivo en la energía no garantiza el mejor resultado global si incrementa los tiempos de espera dentro o entre grupos. Aun así, todas las variantes adaptativas superan con claridad a la línea base.

En resumen, la Fig. 5 muestra que incorporar la energía como objetivo explícito en la política de asignación permite reducir el consumo total, pero también que el punto energéticamente óptimo no tiene por qué coincidir con el punto temporalmente óptimo.

D. Calidad del modelo

La Fig. 6 presenta la métrica final de prueba tras el horizonte fijo de entrenamiento para la tarea MNIST. En este caso se reporta pérdida de entropía cruzada, por lo que valores menores indican mejor calidad final

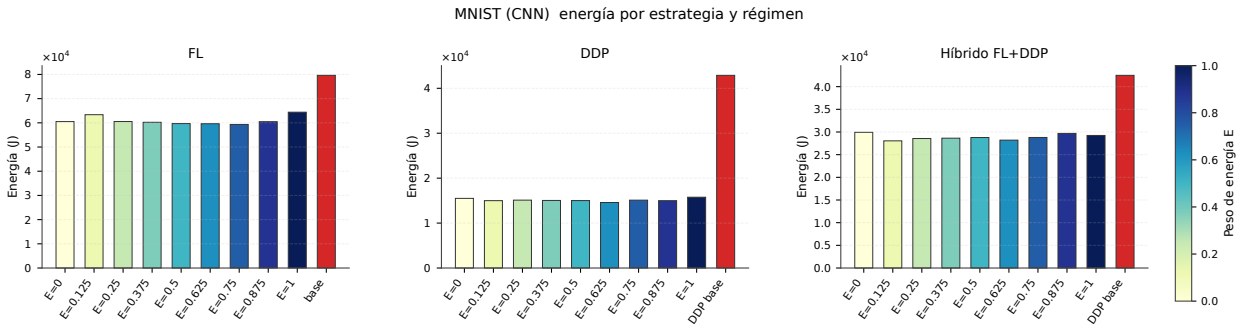


Fig. 5: Energía total consumida para MNIST en los tres regímenes evaluados. Las estrategias están indexadas por el peso energético E. La barra roja representa la línea base: reparto uniforme en FL y DDP estándar en DDP y FL+DDP híbrido. Valores menores son mejores.

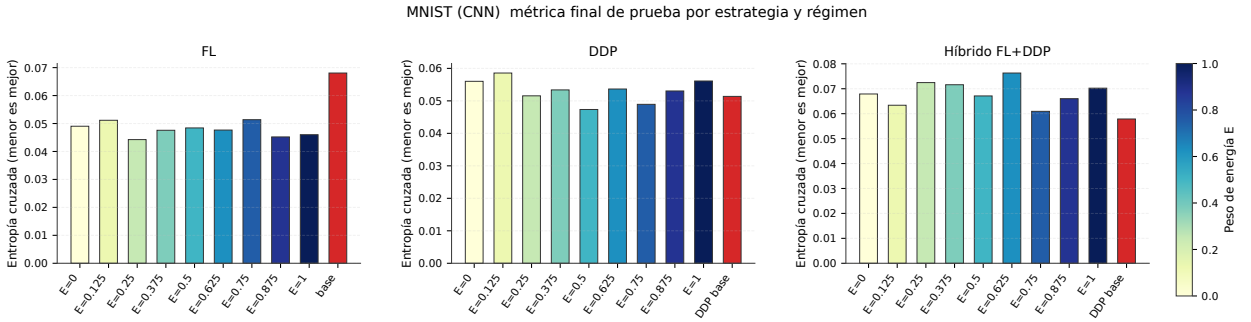


Fig. 6: Métrica final de prueba para MNIST en los tres regímenes evaluados. Se reporta pérdida de entropía cruzada, donde valores menores indican mejor calidad del modelo.

del modelo.

En términos generales, la reasignación adaptativa no altera de forma importante la calidad final del modelo bajo el mismo presupuesto global de trabajo y el mismo horizonte de entrenamiento. En FL, varias configuraciones adaptativas incluso mejoran ligeramente la pérdida respecto a la línea base, aunque las diferencias siguen siendo pequeñas y no modifican la conclusión principal. En DDP e híbrido, las pérdidas finales se mantienen en rangos comparables entre configuraciones, sin mostrar degradaciones sistemáticas asociadas al controlador.

Este comportamiento era esperable, ya que el controlador no modifica ni el modelo, ni la función de pérdida, ni el optimizador, ni el número de épocas locales por ronda. Su efecto se limita a redistribuir el número de muestras procesadas por cada participante según su comportamiento observado. Como resultado, las mejoras obtenidas en tiempo y energía responden principalmente a una mejor adecuación entre carga y capacidad del dispositivo, manteniendo un rendimiento final del modelo comparable al de las configuraciones base.

V. Conclusiones

Este trabajo ha abordado la ineficiencia provocada por los participantes rezagados, o *stragglers*, en entrenamiento distribuido sobre dispositivos heterogéneos de bajo consumo, considerando tres regímenes de coordinación: aprendizaje federado (FL), entrenamiento distribuido paralelo por datos (DDP) y un esquema híbrido FL+DDP. En estos entornos, los puntos de sincronización introducen tiempos de espera cuando los participantes presentan rendimientos distintos, lo que incrementa el tiempo total de ejecución y desperdicia energía en los dispositivos.

Para mitigar este problema sin alterar la semántica del aprendizaje, se ha propuesto un controlador ligero guiado por medidas que opera en los límites de ronda. Aunque cada ronda mantiene un presupuesto global de muestras fijo, el controlador reasigna dinámicamente la cantidad de trabajo de cada participante a partir del tiempo y la energía observados por muestra en la ronda anterior. De esta forma, el método actúa como un mecanismo externo de balanceo adaptativo: en FL reduce la dispersión de tiempos en la barrera de ronda, mientras que en DDP y en el régimen híbrido disminuye el desequilibrio que, de otro modo, amplificaría las esperas en las sincronizaciones a nivel de paso.

Los resultados obtenidos con la tarea MNIST sobre una plataforma heterogénea con dispositivos Arm, RISC-V y x86 muestran que la reasignación adaptativa mejora de forma consistente la eficiencia del sistema. En los tres regímenes evaluados, las estrategias dinámicas reducen el tiempo total de ejecución y el consumo energético respecto a las configuraciones base, especialmente en DDP y en el modo híbrido, donde la frecuencia de sincronización hace que el impacto de los *stragglers* sea mayor. Al mismo tiempo, la calidad final del modelo se mantiene en rangos comparables, lo que indica que las mejoras observadas provienen del mejor reparto del trabajo y no de cambios en el proceso de aprendizaje.

Otro resultado relevante es que el mismo controlador permite recorrer distintos puntos de compromiso entre tiempo y energía mediante el parámetro de ponderación utilizado en la política proporcional.

Esto confirma que la asignación de datos a nivel de ronda constituye un mecanismo de control suficientemente potente para adaptar el comportamiento del sistema a distintos objetivos de eficiencia sin necesidad de modificar ni el modelo, ni el optimizador, ni la función de pérdida.

Desde un punto de vista práctico, esta aproximación resulta especialmente útil en despliegues donde existe heterogeneidad de rendimiento y donde las condiciones de compartición de datos permiten redistribuir dinámicamente la carga. En esos escenarios, la asignación adaptativa evita el desequilibrio persistente que suele aparecer con repartos estáticos e igualitarios y reduce el tiempo ocioso inducido por la sincronización. En conjunto, los resultados apoyan que la redistribución dinámica de trabajo por ronda es una técnica simple, de bajo coste de integración y efectiva para mejorar el aprovechamiento de recursos en entrenamiento distribuido heterogéneo.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación de España mediante el proyecto PID2023-151073NB-I00.

Referencias

[1] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 12:1–12:19, 2019.

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Aguera y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017, pp. 1273–1282.

[3] S. Li et al., "PyTorch distributed: Experiences on accelerating data parallel training," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3005–3018, 2020.

[4] Peter Kairouz, H. Brendan McMahan, and et al., "Advances and open problems in federated learning," *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[5] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith, "Federated optimization in heterogeneous networks," in *Proceedings of MLSys*, 2020.

[6] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh, "SCAFFOLD: Stochastic controlled averaging for federated learning," in *Proceedings of ICML*, 2020, vol. 119 of *PMLR*, pp. 5132–5143.

[7] Fan Lai, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury, "Oort: Efficient federated learning via guided participant selection," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. July 2021, pp. 19–35, USENIX Association.

[8] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng, "TiFL: A tier-based federated learning system," in *Proceedings of HPDC*, 2020.

[9] Takayuki Nishio and Ryo Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. May 2019, IEEE.

[10] D. Huba et al., "Papaya: Practical, private, and scalable federated learning," in *Proceedings of Machine Learning and Systems (MLSys)*, 2022.

[11] Cong Xie, Sanmi Koyejo, and Indranil Gupta, "Asynchronous federated optimization," 2020.

[12] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon, "Federated learning: Strategies for improving communication efficiency," arXiv:1610.05492, 2016.

[13] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *NeurIPS*, 2017.

[14] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek, "Robust and communication-efficient federated learning from non-IID data," *IEEE Transactions on Neural Networks and Learning Systems*, 2020, Early version arXiv:1903.02891.

[15] Alberto Cabrera, Alejandro Acosta, Francisco Almeida, and Vicente Blanco, "A dynamic multi-objective approach for dynamic load balancing in heterogeneous systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 11, pp. 2421–2434, 2020.

[16] X. Yang and et al., "Adaptive data pruning for efficient federated learning on heterogeneous clients," *Cluster Computing*, 2025.

[17] Qunsong Zeng, Yuqing Du, Kaibin Huang, and Kin K. Leung, "Energy-efficient radio resource allocation for federated edge learning," in *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2020, pp. 1–6.

[18] Elia Guerra, Francesc Wilhelmi, Marco Miozzo, and Paolo Dini, "The cost of training machine learning models over distributed data sources," 2023.

[19] Alessandro Buratto, Elia Guerra, Marco Miozzo, Paolo Dini, and Leonardo Badia, "Energy minimization for participatory federated learning in IoT analyzed via game theory," in *2024 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 2024, pp. 249–254.

[20] D. J. Beutel et al., "Flower: A friendly federated learning framework," 2020.

[21] Group of Architecture and Technology of Computing Systems (ArTeCS) of the Complutense University of Madrid, "AccelPowerCape reference," `https://artecs.dacya.ucm.es/tools/accelpowercape/`.

[22] Coley G., "BeagleBone Black system reference manual," `https://www.farnell.com/datasheets/1685587.pdf`.

[23] Adafruit, "INA219 public repository," `https://github.com/adafruit/Adafruit_INA219`.

[24] María Barreda, Sergio Barrachina Mir, Sandra Catalán, Manuel F. Dolz, G. Fabregat, Rafael Mayo, and Enrique Orti, "An integrated framework for power-performance analysis of parallel scientific applications," Jan. 2013.

[25] Alberto Cabrera, Francisco Almeida, Javier Arteaga, and Vicente Blanco, "Measuring energy consumption using EML (Energy Measurement Library)," *Computer Science - Research and Development*, vol. 30, May 2014.

[26] Yann LeCun, Corinna Cortes, and Christopher J. C. Burges, "The MNIST database of handwritten digits," `http://yann.lecun.com/exdb/mnist/`, 1998.

[27] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," 2017.

Lenguajes, compiladores y herramientas de programación y ejecución paralela (1/2)

Análisis Comparativo del Rendimiento de Algoritmos Paralelos de Ordenamiento usando Intel TBB

Mario Rossainz-López[1], Bárbara Sánchez-Rinza[1], Gabriela Martínez Rosas[1], Manuel I. Capel-Tuñón[2]

***Resumen*— Este trabajo presenta un análisis comparativo del rendimiento de tres algoritmos clásicos de ordenamiento: Quicksort, Mergesort y Shellsort. Se implementaron versiones secuenciales y paralelas usando Intel TBB, evaluando su desempeño sobre vectores de gran tamaño (hasta ochenta millones de datos) con datos aleatorios (en un rango de números reales entre 0.0 y 1.0). Los resultados muestran que Quicksort paralelo es el más eficiente, logrando las mayores aceleraciones (speedup) en comparación con sus contrapartes secuenciales y respecto a los otros algoritmos paralelos de ordenación implementados. Shellsort mostró un rendimiento intermedio con buena escalabilidad, mientras que Mergesort fue el menos eficiente. Los resultados encontrados permiten identificar qué algoritmos se benefician más del cómputo en plataformas multicore.**

Palabras Clave— **Ordenamiento paralelo, Intel TBB, Quicksort, Mergesort, Shellsort, speedup**

I. Introducción

La eficiencia de los algoritmos de ordenamiento es crucial en aplicaciones que manipulan grandes volúmenes de datos. Con el avance de los procesadores multicore, el aprovechamiento del paralelismo se ha convertido en una estrategia clave para mejorar el rendimiento de éstos. En este contexto, la biblioteca Intel Threading Building Blocks (TBB) ofrece herramientas para desarrollar software paralelo de manera eficiente y portable (para mayores detalles ver [1]). Este trabajo se centra en el análisis comparativo de tres algoritmos de ordenamiento bien conocidos: Quicksort, Mergesort y Shellsort, cuyos detalles algorítmicos y de complejidad se pueden encontrar en [2], evaluando tanto su rendimiento secuencial como su capacidad de escalamiento al entorno paralelo provisto por TBB. La motivación principal es, a través de un análisis comparativo, determinar cuál de estos algoritmos se beneficia más del paralelismo ofrecido por la biblioteca Intel TBB.

[1]Benemérita Universidad Autónoma de Puebla, Avenida. San Claudio y 14 Sur, San Manuel, Puebla, Puebla, 72000, México. E-mail: {mario.rossainz, barbara.sanchez}@correo.buap.mx, 20240806-01@viep.com.mx

[2]DPT. Ingeniería de Software, Colegio de Informática y Telecomunicaciones, Universidad de Granada, ETSIIT Calle Periodista Daniel Saucedo Aranda s/n, 18071 Granada, España. E-mail: manuelcapel@ugr.es

II. Marco Teórico

El ordenamiento eficiente de datos es crucial para múltiples aplicaciones computacionales, desde bases de datos hasta algoritmos científicos y financieros. Además, muchos procesos de cómputo, como la búsqueda binaria, requieren que los datos estén ordenados previamente. De ahí la importancia de estudiar, comparar y optimizar algoritmos de ordenamiento, tanto en contextos secuenciales como en entornos paralelos. La posibilidad de paralelizar algoritmos clásicos en arquitecturas multicore permite aprovechar mejor los recursos disponibles, reduciendo significativamente los tiempos de procesamiento.

A. Algoritmo de Ordenación Quicksort

Quicksort es uno de los algoritmos más utilizados en la práctica por su velocidad y simplicidad, fue propuesto por Tony Hoare en 1960 y es uno de los algoritmos de ordenamiento más eficientes. Emplea la técnica de divide y vencerás, seleccionando un pivote y particionando el arreglo de modo que todos los elementos menores al pivote queden a un lado, y los mayores al otro, repitiendo el proceso recursivamente. Tiene un tiempo promedio de ejecución de $O(n \log n)$, aunque su peor caso es $O(n^2)$, cuando los pivotes son seleccionados de manera desfavorable. En entornos paralelos, Quicksort es altamente adaptable gracias a la independencia de sus llamadas recursivas, lo que lo convierte en una excelente opción para entornos multicore [3].

B. Algoritmo de Ordenación Mergesort

Mergesort, desarrollado por John von Neumann en 1945, también utiliza la estrategia de divide y vencerás, dividiendo el arreglo en mitades recursivamente hasta llegar a subarreglos de tamaño uno, para luego combinarlos en orden. A diferencia de Quicksort, Mergesort es estable y siempre tiene una complejidad de $O(n \log n)$ en el mejor caso, el caso promedio y peor caso. No obstante, su consumo de memoria es mayor, ya que necesita espacio adicional para realizar las fusiones; aun así, su diseño recursivo lo hace apto para la paralelización. Sin embargo, la fase de fusión puede generar cuellos de botella en implementaciones paralelas si no se administra adecuadamente la sincronización [4].

C. Algoritmo de Ordenación Shellsort

Shellsort fue propuesto por Donald Shell en 1959 como una mejora del ordenamiento por inserción. Su principio es comparar elementos separados por un cierto intervalo o "gap", que disminuye en cada iteración. Aunque su rendimiento depende de la secuencia de incrementos utilizada, Shellsort puede alcanzar complejidades en torno a O(n^1.5) o incluso mejores con optimizaciones, aunque en el peor caso puede llegar a $O(n^2)$. Este algoritmo es considerado como intermedio en cuanto a su complejidad y facilidad de paralelización respecto a Quicksort y Mergesort. Estudios recientes muestran que, aunque no es el más rápido, puede ser útil cuando se requiere un balance entre simplicidad y rendimiento, especialmente en entornos con restricciones de memoria [5].

III. Threading Building Block (TBB)

TBB es una librería implementada en C++ moderno que surge en el 2006 cuyo objetivo es proporcionar a los desarrolladores una manera fácil y poderosa de generar aplicaciones que continúen escalando a medida que se introducen nuevas plataformas, con diferentes arquitecturas y más núcleos. TBB proporciona plantillas de alto nivel que se adaptan a las necesidades de la mayoría de los programadores, centrándose en casos comunes [6]. Pero también proporciona interfaces de bajo nivel para profundizar y crear soluciones personalizadas. Esta librería cumple dos funciones clave:

1. llena los vacíos fundamentales en apoyo al paralelismo donde el estándar de C++ no ha evolucionado lo suficiente
2. Proporciona abstracciones de nivel superior para el paralelismo que está más allá del alcance de lo que probablemente alguna vez incluirá el estándar del lenguaje C++ [6].

TBB contiene características que se clasifican en dos grandes grupos: interfaces para expresar cálculos paralelos mediante tareas e interfaces que son independientes del modelo de ejecución incluyendo contenedores concurrentes [6].

A. Interfaces de Ejecución Paralela

Como desarrolladores de TBB, expresamos nuestra aplicación mediante tareas, ya sea directa o indirectamente a través de los algoritmos pre-empaquetados, y TBB computa estas tareas en los recursos de hardware de la plataforma por nosotros (ver Fig. 1). Debido a que, como desarrolladores, es posible que queramos expresar diferentes tipos de paralelismo, la librería nos ofrece tres capas de paralelismo que son las más comunes para programar aplicaciones paralelas. La capa message-driven basada en paso de mensajes. Los patrones de diseño comunes en esta capa incluyen grafos de transmisión, grafos de flujo de datos y grafos de dependencia. La capa fork-join que admite patrones de diseño en los que un cálculo en serie se bifurca (ramifica) en un conjunto de tareas paralelas y luego continúa solo cuando se completan los subcálculos paralelos. Y la capa SIMD donde se explota el paralelismo de datos aplicando la misma operación a múltiples elementos de datos simultáneamente [6], la Fig.2 muestra las estructuras de estas tres capas.

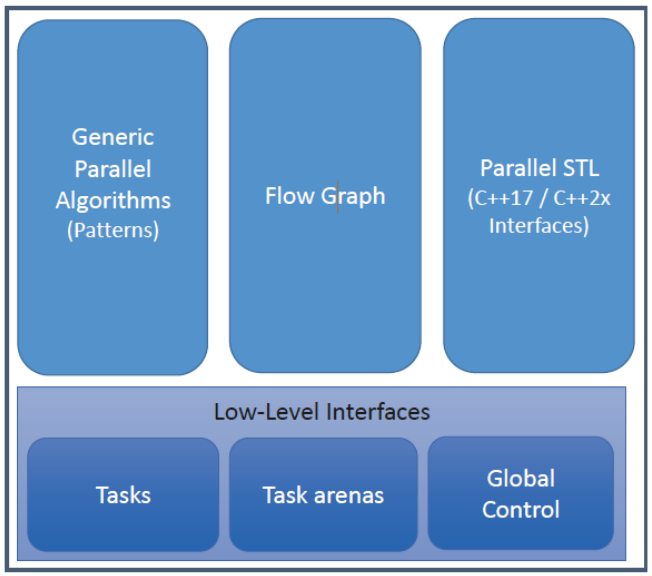


Fig.1. Características de la Librería TBB: Interfaces de Ejecución Paralela (Tomada de [6])

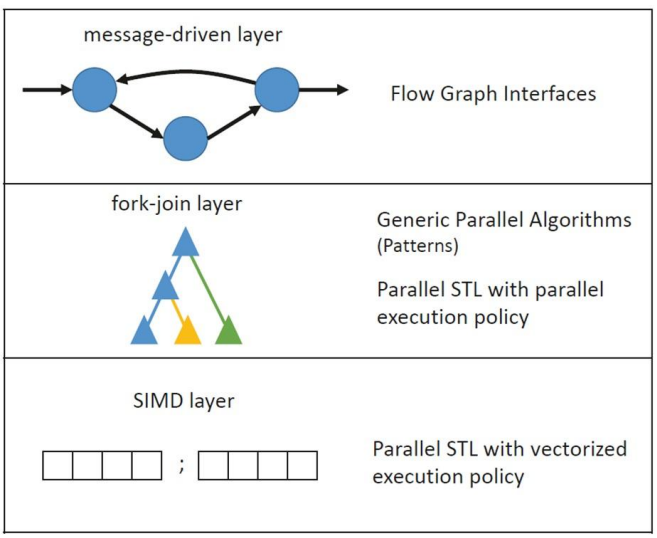


Fig.2. Las capas de Paralelismo en TBB: message-driven, fork-join y SIMD (Tomada de [6])

B. Interfaces Independientes del Modelo de Ejecución

Este segundo grupo de características de TBB es completamente independiente del modelo de ejecución y de las tareas de la librería. Estas características son útiles en aplicaciones que usan threads nativos, como pthreads o WinThreads, al igual que en aplicaciones que usan tareas TBB. Incluyen contenedores concurrentes que proporcionan interfaces amigables con threads para estructuras de datos comunes como tablas hash, colas y vectores. También incluyen funciones para asignación de memoria escalable y caché, además de características de nivel inferior, como primitivas de sincronización y almacenamiento local de threads [6], ver Fig. 3.

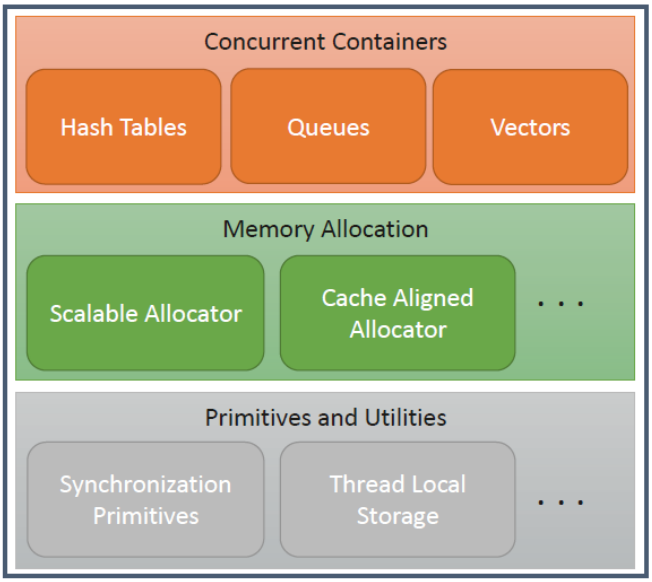


Fig.3. Características de la Librería TBB: Interfaces Independientes del Modelo de Ejecución (Tomada de [6])

IV. METODOLOGÍA

Se desarrollaron versiones secuenciales y paralelas de cada algoritmo de ordenación en C++, utilizando Intel TBB. Las versiones paralelas usaron *"parallel_invoke( )"* para dividir las llamadas recursivas y *"parallel_for( )"* para iteraciones independientes [7]. Las pruebas se realizaron sobre vectores con tamaños de 5, 10, 20, 40 y 80 millones de datos tipo double, generados aleatoriamente en el rango [0.0, 1.0]. Cada experimento se ejecutó en una máquina multicore en condiciones controladas. Se midió el tiempo de ejecución y se calculó el speedup como la razón entre el tiempo secuencial y el tiempo paralelo.

V. PROGRAMACIÓN SECUENCIAL DE LOS ALGORITMOS DE ORDENACIÓN

Se muestra la codificación secuencial de los algoritmos de ordenación Quicksort (Fig.4), Mergesort (Fig.5) y Shellsort (Fig.6); los tiempos de ejecución de cada uno de ellos con el mismo tamaño y conjunto de datos de tipo double entre 0.0 y 1.0; y el análisis comparativo que identifica cuál de estos algoritmos de ordenación es el mejor.

```cpp
#include <algorithm>
#include <cfloat>
#include <iostream>
#include <random>
#include <vector>
#include <tbb/tbb.h>

struct DataItem{
                int id;
                double value;
                DataItem(int i, double v):id{i}, value{v} {}
                };

using QSVector= std::vector<DataItem>;

void serialQuickSort(QSVector::iterator b, QSVector::iterator e){
    if(b>=e) return;
    double pivot_value= b->value;
    QSVector::iterator i=b, j=e-1;
    while(i!=j){
        while(i!=j && pivot_value < j->value) --j;
        while(i!=j && i->value <= pivot_value) ++i;
        std::iter_swap(i,j);
    }

    std::iter_swap(b,i);
    serialQuickSort(b,i);
    serialQuickSort(i+1,e);
}

static QSVector makeQSData(int N, unsigned seed = 123456789){
    QSVector v;
    std::default_random_engine g(seed);
    std::uniform_real_distribution<double> d(0.0,1.0);
    for(int i=0; i<N; ++i)
        v.push_back(DataItem{i,d(g)});
    return v;
}

int main(){
    const int N=50000000;
    QSVector v1=makeQSData(N);
    double serial_time=0.0;
    {
        tbb::tick_count t0= tbb::tick_count::now();
        serialQuickSort(v1.begin(),v1.end());
        serial_time=(tbb::tick_count::now()-t0).seconds();
    }
    std::cout << "tiempo serial = " << serial_time << " segundos" << std::endl;
    return 1;
}
```

Fig 4. Código Secuencial Quicksort en C++ Moderno

```cpp
#include <algorithm>
#include <cfloat>
#include <iostream>
#include <random>
#include <vector>
#include <tbb/tbb.h>

struct DataItem {
                int id;
                double value;
                DataItem(int i, double v) : id{i}, value{v} {}
                };
using QSVector = std::vector<DataItem>;
void merge(QSVector::iterator b, QSVector::iterator m, QSVector::iterator e) {
    QSVector temp;
    auto i = b;
    auto j = m;
    while (i != m && j != e) {
        if (i->value <= j->value) {
            temp.push_back(*i);
            ++i;
        } else {
            temp.push_back(*j);
            ++j;
          }
    }
    temp.insert(temp.end(), i, m);
    temp.insert(temp.end(), j, e);
    std::move(temp.begin(), temp.end(), b);
}
void serialMergeSort(QSVector::iterator b, QSVector::iterator e) {
    if (e - b <= 1) return;
    auto m = b + (e - b) / 2;
    serialMergeSort(b, m);
    serialMergeSort(m, e);
    merge(b, m, e);
}
static QSVector makeQSData(int N, unsigned seed = 123456789) {
    QSVector v;
    std::default_random_engine g(seed);
    std::uniform_real_distribution<double> d(0.0, 1.0);
    for (int i = 0; i < N; ++i) v.push_back(DataItem{i, d(g)});
    return v;
}
int main() {
    const int N = 50000000;
    QSVector v1 = makeQSData(N);
    double serial_time = 0.0;
    tbb::tick_count t0 = tbb::tick_count::now();
    serialMergeSort(v1.begin(), v1.end());
    serial_time = (tbb::tick_count::now() - t0).seconds();
    std::cout << "tiempo serial = " << serial_time << " segundos" << std::endl;
    return 1;
}
```

Fig 5. Código Secuencial Mergesort en C++ Moderno

```cpp
#include <algorithm>
#include <cfloat>
#include <algorithm>
#include <cfloat>
#include <iostream>
#include <random>
#include <vector>
#include <tbb/tbb.h>

struct DataItem {
                 int id;
                 double value;
                 DataItem(int i, double v) : id{i}, value{v} {}
                };
using QSVector = std::vector<DataItem>;

void serialShellSort(QSVector::iterator b, QSVector::iterator e) {
    int n = std::distance(b, e);
    int gap = 1;
    while (gap < n / 3) gap = 3 * gap + 1;
    while (gap >= 1) {
        for (auto i = b + gap; i != e; ++i) {
            auto temp = *i;
            auto j = i;
            while (j >= b + gap && (j - gap)->value > temp.value) {
                *j = *(j - gap);
                j -= gap;
            }
            *j = temp;
        }
        gap /= 3;
    }
}

static QSVector makeQSData(int N, unsigned seed = 123456789) {
    QSVector v;
    std::default_random_engine g(seed);
    std::uniform_real_distribution<double> d(0.0, 1.0);
    for (int i = 0; i < N; ++i) v.push_back(DataItem{i, d(g)});
    return v;
}

int main() {
    const int N = 50000000;
    QSVector v1 = makeQSData(N);
    double serial_time = 0.0;
    {
     tbb::tick_count t0 = tbb::tick_count::now();
     serialShellSort(v1.begin(), v1.end());
     serial_time = (tbb::tick_count::now() - t0).seconds();
    }
    std::cout << "tiempo serial = " << serial_time << " segundos" << std::endl;
    return 0;
}
```

Fig 6. Código Secuencial Shellsort en C++ Moderno

En la Tabla 1 se muestran los tiempos de ejecución de cada algoritmo de ordenación codificado en C++ moderno; todos ellos con los mismos vectores de datos de tipo double, generados aleatoriamente en el rango [0.0, 1.0] con tamaños de 5, 10, 20, 40 y 80 millones de datos respectivamente.

Tabla 1. Comparativo de los tiempos de ejecución en segundos de los algoritmos de ordenación Quicksort, Mergesort, Shellsort en su implementación secuencial

Tamaño N	Run Time SEQ/segs QuickSort	Run Time SEQ/Segs MergeSort	Run Time SEQ/segs ShellSort
5000000	0.508683	1.73975	1.04427
10000000	1.09436	3.72541	2.45262
20000000	2.20958	8.03072	5.693
40000000	4.58462	17.2532	11.6497
80000000	10.166	37.1085	29.7545

Estos resultados muestran claramente que Shellsort tuvo un tiempo medio y Mergesort fue el más lento, mientras Quicksort, en su versión secuencial, es el algoritmo más eficiente para los conjuntos de datos trabajados. Su tiempo de ejecución fue significativamente menor que el de los otros dos algoritmos (ver Fig.7).

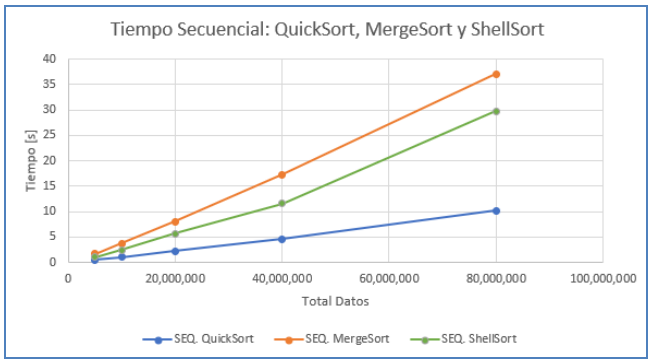


Fig 7. Gráfica comparativa de los tiempos de ejecución de las implementaciones de los algoritmos de ordenación Quicksort, Mergesort y Shellsort en sus versiones secuenciales con tamaños de 5 a 80 millones de números aleatorios de tipo double en un rango de [0.0,1.0]

En un entorno secuencial y para los tamaños de los datos de entrada que se trabajaron (ver Tabla 1), Quicksort es el más adecuado en términos de rapidez.

VI. Programación Paralela de los Algoritmos de Ordenación

La paralelización de algoritmos ha cobrado gran relevancia con el auge de procesadores multinúcleo. Intel Threading Building Blocks (TBB) es actualmente una de las bibliotecas cada vez más conocida y más utilizada para implementar tareas concurrentes en C++ y nos proporciona primitivas como *parallel_invoke( )* y *parallel_for( )* que facilitan la implementación de versiones paralelas de algoritmos clásicos. Su enfoque basado en tareas abstrae la complejidad del manejo directo de hilos, permitiendo escalar de forma eficiente en arquitecturas modernas [8]. Los algoritmos de ordenamiento paralelos deben considerar aspectos como granularidad, sincronización, balance de carga y sobrecarga de administración de threads. La eficiencia paralela se mide mediante métricas como el speedup y la escalabilidad, siendo Quicksort uno de los algoritmos de ordenación más destacados en este sentido [9]. Se muestra a continuación la codificación paralela de los algoritmos de ordenación Quicksort (Fig.8), Mergesort (Fig.9) y Shellsort (Fig.10); utilizando las primitivas *parallel_invoke( )* y *parallel_for( )* de la librería TBB de Intel, los tiempos de ejecución de cada uno de ellos con el mismo tamaño y conjunto de datos de tipo double entre 0.0 y 1.0; y el análisis comparativo que identifica cuál de estos algoritmos de ordenación es el mejor, respecto a las versiones secuenciales de la sección anterior y entre ellos mismos como algoritmos paralelos.

```cpp
#include <algorithm>
#include <cfloat>
#include <iostream>
#include <random>
#include <vector>
#include <tbb/tbb.h>

struct DataItem{
                int id;
                double value;
                DataItem(int i, double v):id{i}, value{v} {}
                };

using QSVector= std::vector<DataItem>;

void parallelQuickSort(QSVector::iterator b, QSVector::iterator e){
    if(b>=e) return;
    double pivot_value= b->value;
    QSVector::iterator i=b, j=e-1;
    while(i!=j){
        while(i!=j && pivot_value < j->value) --j;
        while(i!=j && i->value <= pivot_value) ++i;
        std::iter_swap(i,j);
    }
    std::iter_swap(b,i);
    tbb::parallel_invoke(
        [=](){parallelQuickSort(b,i);},
        [=](){parallelQuickSort(i+1,e);}
    );
}

static QSVector makeQSData(int N, unsigned seed = 123456789){
    QSVector v;
    std::default_random_engine g(seed);
    std::uniform_real_distribution<double> d(0.0,1.0);
    for(int i=0; i<N; ++i)
        v.push_back(DataItem{i,d(g)});
    return v;
}

int main(){
    const int N=10;
    QSVector v1=makeQSData(N);
    double serial_time=0.0;
    {
     tbb::tick_count t0= tbb::tick_count::now();
     parallelQuickSort(v1.begin(),v1.end());
     serial_time=(tbb::tick_count::now()-t0).seconds();
    }
    std::cout << "tiempo serial = " << serial_time << " segundos" << std::endl;
    return 1;
}
```

Fig 8. Quicksort Paralelo TBB

```cpp
#include <algorithm>
#include <cfloat>
#include <iostream>
#include <random>
#include <vector>
#include <tbb/tbb.h>
struct DataItem {
                int id;
                double value;
                DataItem(int i, double v) : id{i}, value{v} {}
};
using QSVector = std::vector<DataItem>;
void merge(QSVector::iterator b, QSVector::iterator m, QSVector::iterator e) {
    QSVector temp;
    auto i = b;
    auto j = m;
    while (i != m && j != e) {
        if (i->value <= j->value) {
            temp.push_back(*i);
            ++i;
        } else {
                temp.push_back(*j);
                ++j;
                }
    }
    temp.insert(temp.end(), i, m);
    temp.insert(temp.end(), j, e);
    std::move(temp.begin(), temp.end(), b);
}
void parallelMergeSort(QSVector::iterator b, QSVector::iterator e) {
    if (e - b <= 1) return;
    auto m = b + (e - b) / 2;
    tbb::parallel_invoke(
        [=](){parallelMergeSort(b, m);},
        [=](){parallelMergeSort(m, e);}
    );
    merge(b, m, e);
}
static QSVector makeQSData(int N, unsigned seed = 123456789) {
    QSVector v;
    std::default_random_engine g(seed);
    std::uniform_real_distribution<double> d(0.0, 1.0);
    for (int i = 0; i < N; ++i) v.push_back(DataItem{i, d(g)});
    return v;
}
int main() {
    const int N = 50000000;
    QSVector v1 = makeQSData(N);
    double parallel_time = 0.0;
     tbb::tick_count t0 = tbb::tick_count::now();
     parallelMergeSort(v1.begin(), v1.end());
     parallel_time = (tbb::tick_count::now() - t0).seconds();
    std::cout << "tiempo paralelo = " << parallel_time << " segundos" << std::endl;
    return 1;
}
```

Fig 9. Mergesort Paralelo TBB

```cpp
#include <algorithm>
#include <cfloat>
#include <iostream>
#include <random>
#include <vector>
#include <tbb/tbb.h>
struct DataItem {
                int id;
                double value;
                DataItem(int i, double v) : id{i}, value{v} {}
};
using QSVector = std::vector<DataItem>;
void insertionSortForGap(QSVector::iterator b, int n, int start, int gap) {
    for (int i = start + gap; i < n; i += gap) {
        auto temp = *(b + i);
        int j = i;
        while (j >= gap && (b + j - gap)->value > temp.value) {
            *(b + j) = *(b + j - gap);
            j -= gap;
        }
        *(b + j) = temp;
    }
}
void parallelShellSort(QSVector::iterator b, QSVector::iterator e) {
    int n = std::distance(b, e);
    int gap = 1;
    while (gap < n / 3) gap = 3 * gap + 1;
    while (gap >= 1) {
        tbb::parallel_for(tbb::blocked_range<size_t>(0, gap),
        [&](const tbb::blocked_range<size_t>& r) {
        for (size_t start = r.begin(); start != r.end(); ++start)
            insertionSortForGap(b, n, start, gap);
        });
        gap /= 3;
    }
}
static QSVector makeQSData(int N, unsigned seed = 123456789) {
    QSVector v;
    std::default_random_engine g(seed);
    std::uniform_real_distribution<double> d(0.0, 1.0);
    for (int i = 0; i < N; ++i) v.push_back(DataItem{i, d(g)});
    return v;
}
int main() {
    const int N = 80000000;
    QSVector v1 = makeQSData(N);
    double parallel_time = 0.0;
    {
     tbb::tick_count t0 = tbb::tick_count::now();
     parallelShellSort(v1.begin(), v1.end());
     parallel_time = (tbb::tick_count::now() - t0).seconds();
    }
    std::cout << "tiempo paralelo = " << parallel_time << " segundos" << std::endl;
return 1;
}
```

Fig 10. Shellsort Paralelo TBB

En la Tabla 2 se muestran los tiempos de ejecución de cada algoritmo de ordenación paralelo codificados en C++ moderno utilizando la librería TBB de Intel; todos ellos con los mismos vectores de datos de tipo double, generados aleatoriamente en el rango [0.0, 1.0] con tamaños de 5, 10, 20, 40 y 80 millones de datos respectivamente.

Tabla 2. Comparativo de tiempos de ejecución en segundos de los algoritmos paralelos de ordenación Quicksort, Mergesort, Shellsort utilizando la librería TBB de Intel

Tamaño N	Run Time PAR/segs Quicksort Parallel	Run Time PAR/Segs Merge Sort Parallel	Run Time PAR/segs Shell Sort Parallel
5000000	0.28866	0.840184	0.662559
10000000	0.618755	1.69563	1.40603
20000000	1.41982	3.56097	3.17416
40000000	2.66412	8.36681	7.6652
80000000	5.82909	19.202	15.1919

Estos resultados muestran que todos los algoritmos paralelos mejoran considerablemente respecto a sus versiones secuenciales en sus tiempos de ejecución: QuickSort paralelo es el más rápido en tiempo absoluto; ShellSort paralelo también mejora mucho, reduciendo casi a la mitad su tiempo de ejecución y MergeSort paralelo mejora bastante, pero sigue siendo el más lento en valores absolutos. (ver Fig.11).

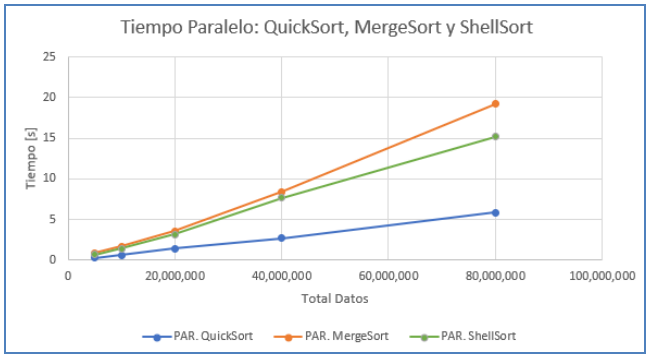


Fig 11. Gráfica comparativa de los tiempos de ejecución de las implementaciones paralelas de los algoritmos de ordenación Quicksort, Mergesort y Shellsort utilizando la librería TBB de Intel con tamaños de 5 a 80 millones de números aleatorios de tipo double en un rango de [0.0,1.0]

VII. Comparativo del Análisis de Rendimiento de Quicksort, Mergesort y Shellsort Paralelos

Para el análisis del rendimiento respecto de la aceleración o speedup de los algoritmos paralelos de ordenación Quicksort, Mergesort y Shellsort versus sus versiones secuenciales para el ordenamiento de 5 a 80 millones de datos aleatorios de tipo double en un rango de [0.0,1.0], se llevaron a cabo en una computadora con procesador Intel CORE i7 de 20 procesadores lógicos y se realizó el cálculo del speedup a través de los siguientes cruces: Quicksort Parallel vs (Quicksort Seq, Mergesort Seq, Shellsort Seq) tal como lo muestra la Tabla 3, Mergesort Parallel vs (Quicksort Seq, Mergesort Seq, Shellsort Seq) cuyos resultados se muestran en la Tabla 4 y Shellsort Parallel vs (Quicksort Seq, Mergesort Seq, Shellsort Seq) como lo muestra la Tabla 5.

Tabla 3. Comparativo del speedup o aceleración del algoritmo paralelo de ordenación Quicksort Parallel respecto de los algoritmos secuenciales de ordenación Quicksort, Mergesort, Shellsort

Tamaño N	Speedup Quicksort Parallel vs Quicksort Seq	Speedup Quicksort Parallel vs Merge sort Seq	Speedup Quicksort Parallel vs Shell sort Seq
5000000	1.762221991	6.026986766	3.617647059
10000000	1.768648334	6.020815993	3.963798272
20000000	1.556239523	5.656153597	4.009663197
40000000	1.720875937	6.476134709	4.372813537
80000000	1.744011501	6.366088017	5.104484576

Tabla 4. Comparativo del Speedup o aceleración del algoritmo paralelo de ordenación Mergesort Parallel respecto de los algoritmos secuenciales de ordenación Quicksort, Mergesort, Shellsort

Tamaño N	Speedup Merge sort Parallel vs Quicksort Seq	Speedup Merge sort Parallel vs Merge sort Seq	Speedup Merge sort Parallel vs Shell sort Seq
5000000	0.605442379	2.070677375	1.242906316
10000000	0.645400235	2.197065398	1.446435838
20000000	0.620499471	2.255205745	1.598721697
40000000	0.547953163	2.062100131	1.392370569
80000000	0.529424018	1.932533069	1.54955213

Tabla 5. Comparativo del Speedup o aceleración del algoritmo paralelo de ordenación Shellsort Parallel respecto de los algoritmos secuenciales de ordenación Quicksort, Mergesort, Shellsort

Tamaño N	Speedup Merge sort Parallel vs Quicksort Seq	Speedup Merge sort Parallel vs Merge sort Seq	Speedup Merge sort Parallel vs Shell sort Seq
5000000	0.767755023	2.625803891	1.57611624
10000000	0.778333321	2.649594959	1.744358228
20000000	0.696114878	2.530029992	1.793545379
40000000	0.598108334	2.250847988	1.519816835
80000000	0.669172388	2.44265036	1.95857661

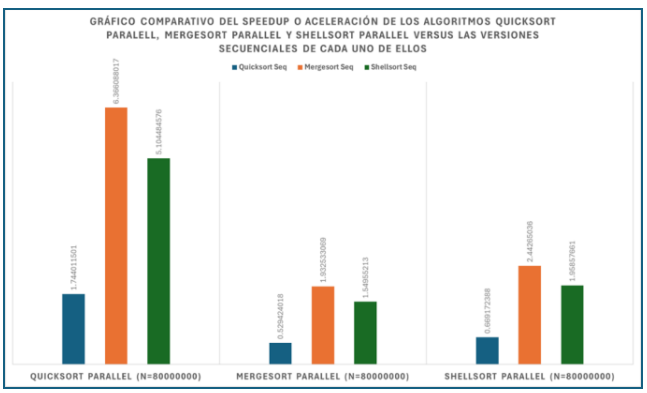


Fig 12. Gráfica comparativa del speedup o aceleración de los algoritmos paralelos de ordenación Quicksort Parallel, Mergesort Parallel y Shellsort Parallel respecto de las versiones secuenciales de ellos mismos en una comparación todos contra todos.

Del análisis de los resultados obtenidos en las tablas anteriores se observa que Quicksort paralelo muestra su mayor ventaja frente a Mergesort y Shellsort, y su menor ventaja frente a su versión secuencial en cuanto a aceleración se refiere. Hay una escalabilidad limitada frente a Quicksort secuencial, posiblemente por limitaciones del hardware o bien la elección del pivote no óptimo para dividir recursivamente. No obstante, aunque el speedup frente a Quicksort secuencial es menor (por la similitud en su estructura y eficiencia base), el algoritmo Quicksort paralelo vale la pena especialmente frente a algoritmos secuenciales más costosos como Mergesort o Shellsort, en particular cuando se trabaja con grandes volúmenes de datos. Por otro lado, el rendimiento y escalabilidad de Mergesort Paralelo es muy similar al que presenta Shellsort y muy pobres respecto al speedup encontrado en Quicksort Paralelo. Esto último se puede ver mejor en la gráfica de la Fig.12 donde se grafican los speedup encontrados en la última línea de cada una de las tres tablas anteriores.

VIII. Conclusiones

Hemos presentado la implementación secuencial y paralela (utilizando *"parallel_invoke( )"* y *"parallel_for( )"* de la librería de clases llamada Threading Building Blocks – TBB, para la programación paralela con paso de mensajes en C++ [10]) de los algoritmos de ordenación Quicksort, Mergesort y Shellsort para realizar un análisis comparativo respecto a la eficiencia, aceleración y escalabilidad de dichas implementaciones paralelas. Con un tamaño lo suficientemente grande (de 5 a 80 millones de datos de tipo

double en un rango de [0.0,1.0] en una computadora con procesador Intel CORE i7 de 20 procesadores lógicos, la ejecución de estas implementaciones arrojó que Quicksort paralelo es claramente el más eficiente, pues supera por mucho a las otras versiones secuenciales y paralelas que se han mostrado. Mergesort y Shellsort paralelos no escalan tan bien como se esperaría pues a pesar de tener speedup frente a sus versiones secuenciales, no justifican su uso frente a Quicksort. La comparación de los otros algoritmos de ordenación contra Quicksort secuencial es útil como referencia ya que muchos algoritmos paralelos no lo superan, lo que resalta su eficiencia base. La efectividad del paralelismo varía según el algoritmo, por ejemplo, Quicksort tiene mejor estructura para dividir y conquistar en paralelo, mientras que Mergesort y Shellsort posiblemente enfrentan mayor overhead o dificultades de balance de carga. Finalmente, los resultados mostrados en la sección VII reflejan claramente que QuickSort paralelo ofrece el mejor desempeño bajo las condiciones evaluadas, tanto por su velocidad como por su eficiencia al distribuir el trabajo entre múltiples threads. Esto lo posiciona como la opción más adecuada para tareas de ordenamiento cuando se cuenta con procesamiento paralelo.

REFERENCIAS

[1] Rehder, D.: Pro TBB: C++ parallel programming with Threading Building Blocks. Apress. https://doi.org/10.1007/978-1-4842-6180-3. (2021).

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C.: Introduction to algorithms (4th ed.). The MIT Press. (2022).

[3] Sedgewick, R., & Wayne, K.: Algorithms (4th ed.). Addison-Wesley (2016).

[4] Blelloch, G. E., & Reid-Miller, M.: Fast Parallel Algorithms for Sorting and Prefix Sums. Carnegie Mellon University (2010).

[5] Skiena, S.: The Algorithm Design Manual (3rd ed.). Springer. (2020).

[6] Voss M., Asenjo R., Reinders J.: Pro TBB. C++ Parallel Programming with Threading Building Blocks. Apress Open. USA (2019).

[7] Smirnov, A., Kozhevnikov, M., & Sidorov, D.: Parallelization of scientific applications using Intel TBB library. Procedia Computer Science, 136, 442–451. (2018).

[8] Reinders, J.: Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism. Intel Press (2021).

[9] Acar, U. A., Blelloch, G. E., & Tangwongsan, K.: Parallel Sorting with Optimal Speedup. ACM Transactions on Parallel Computing, 1(1), 1–28 (2013).

[10] Amini, L., He, Y., Nicol, D., Sanders, R., & Shukla, S.: Platform-independent dynamic parallelism with Intel Threading Building Blocks. Concurrency and Computation: Practice and Experience, 19(18), 2281–2299. (2007).

¿Highway para álgebra lineal dispersa? Estudio de Rendimiento y Portabilidad

Patricia Siwinska[1], Adrián Castelló[1], Héctor Martínez[2], Andrés E. Tomás[1] y Enrique S. Quintana-Ortí[1]

Resumen— **Lograr un rendimiento portable en la vectorización SIMD sigue siendo un desafío fundamental en el álgebra lineal dispersa. Aunque las intrínsecas específicas de cada arquitectura ofrecen una gran eficiencia, limitan la portabilidad entre las extensiones vectoriales de CPU, que evolucionan constantemente. Este trabajo evalúa Google Highway, una capa de abstracción de código abierto en C++ para la programación SIMD portable, dentro del desarrollo de kernels de alto rendimiento en álgebra lineal dispersa, en particular para la multiplicación matriz dispersa–vector (SpMV). Se comparan implementaciones vectoriales de SpMV basadas en los formatos CSR y sELL-c con la autovectorización del compilador y con intrínsecas específicas del hardware en x86-64 AVX2, Arm NEON, Arm SVE2 y RISC-V RVV 1.0. Los experimentos realizados con matrices de la colección SuiteSparse muestran que el rendimiento varía en función de la plataforma de destino.**

Palabras clave— **Paralelismo SIMD, Intrínsecas Vectoriales, Google Highway, Álgebra Lineal Dispersa, Alto Rendimiento**

I. Introducción

LOS procesadores modernos incorporan unidades SIMD (Single Instruction, Multiple Data) para mejorar el rendimiento ejecutando una misma instrucción sobre múltiples datos de forma simultánea [1]. Con este objetivo, las arquitecturas amplían su ISA (*instruction set architecture*) base con extensiones vectoriales que introducen registros vectoriales, operaciones aritméticas y mecanismos especializados de carga y almacenamiento (*load/store*).

Las intrínsecas vectoriales (*vector intrinsics*) ofrecen gran parte de las ventajas de rendimiento de las instrucciones vectoriales de bajo nivel, a la vez que se integran en lenguajes de alto nivel como C/C++, lo que facilita la legibilidad, la comprobación de tipos y la optimización por parte del compilador. Sin embargo, la mayoría de las extensiones de intrínsecas vectoriales, así como todas las instrucciones vectoriales a nivel ensamblador, plantean serios problemas de portabilidad al estar ligadas a arquitecturas de procesador concretas y a sus extensiones vectoriales asociadas.

Google Highway (GHW)[1] aborda este problema introduciendo una biblioteca de código abierto que implementa una abstracción portable sobre la programación SIMD, orientada a las extensiones vectoriales de múltiples ISAs. No obstante, GHW evita deliberadamente exponer ciertas características de bajo nivel propias de cada ISA con el fin de preservar la portabilidad, lo que implica que las garantías de rendimiento son conservadoras. Como resultado, GHW aún no puede servir como un reemplazo general para los intrínsecas específicas de arquitectura en núcleos (*kernels*) altamente críticos para el rendimiento.

En este trabajo, se investiga GHW como alternativa de alto rendimiento para desarrollar núcleos de álgebra lineal dispersa, acelerados mediante vectorización en CPUs modernas, con vistas a su integración en *instancias portables* de bibliotecas de álgebra lineal numérica como PETSc [2], Ginkgo [3] o Trilinos [4]. Nuestro trabajo se centra en la multiplicación de matriz dispersa–vector (SpMV), un núcleo especialmente difícil de vectorizar de forma eficiente debido a sus patrones de acceso indirecto a la memoria. En concreto, las contribuciones de este artículo son las siguientes:

- Se presentan implementaciones de alto rendimiento de SpMV mediante GHW para el formato de matriz CSR [5] y la variante sELL-*c* de ELLPACK [6], [7]. Aunque existe una amplia variedad de formatos específicos orientados a distintas aplicaciones y/o arquitecturas (véase, p.ej., [8], [9], [10], [11], [12], [7], [13], [14]), estas representaciones matriciales ofrecen un equilibrio entre flexibilidad, generalidad y eficiencia en memoria, siendo las opciones preferidas en numerosas aplicaciones científicas y de ingeniería.
- Se expone una evaluación completa que compara GHW frente a la autovectorización del compilador y cuatro extensiones vectoriales específicas del hardware: x86-64 AVX2, Arm NEON, Arm SVE2 y RISC-V RVV 1.0, sobre distintas plataformas.
- Se integran las rutinas vectoriales desarrolladas en versiones prototipo de PETSc y Ginkgo, dos bibliotecas de código abierto ampliamente utilizadas para la resolución escalable de problemas de computación científica.

El resto del artículo se organiza de la siguiente manera. En la Sección II se ofrece una introducción a las intrínsecas vectoriales de GHW. En las Secciones III y IV, se detallan nuestras implementaciones de SpMV basadas en GHW y se evalúan en las plataformas objetivo. Finalmente, la Sección V recoge las conclusiones del trabajo.

[1]Universitat Politècnica de València, e-mail: {psiwins,adcastel,antodo,quintana}@disca.upv.es
[2]Universidad de Córdoba, e-mail: el2mapeh@uco.es
[1]https://github.com/google/highway

II. Extensiones Vectoriales en las ISAs Modernas

En esta sección, utilizamos un núcleo vectorial simple para introducir brevemente la programación paralela con intrínsecas vectoriales y resaltar las diferencias entre GHW y otras extensiones de ISA específicas.

GHW (versión 1.3.0) actualmente soporta ocho plataformas (incluyendo Armv7+, POWER, RISC-V, WebAssembly y x86-64) y un total de 27 objetivos (*targets*) diferentes. Para lograr esto, GHW modela las instrucciones vectoriales como funciones de C++ sobrecargadas y libres que operan sobre variables de tipos vectoriales. Además, GHW permite el *dynamic dispatch*, donde el código se compila para múltiples extensiones vectoriales y se selecciona la más adecuada en tiempo de ejecución.

La Figura 1 ilustra las diferencias entre GHW y otras extensiones vectoriales mediante el núcleo AXPY ($y = \alpha x + y$) de BLAS,[2] vectorizado con x86-64 AVX2, Arm SVE2, RISC-V RVV 1.0 y GHW. El escalar α se almacena en la variable `alpha`, y las direcciones de inicio de los dos vectores, ambos de dimensión `n`, en `x` e `y` respectivamente. El tipo de dato base es `double` (FP64) y los códigos están formateados para facilitar su comparación visual.

Conviene destacar algunas diferencias importantes:

- AVX2 es dependiente de la arquitectura y específico en cuanto a la longitud vectorial. En concreto, AVX2 opera con registros vectoriales de 256 bits, con capacidad para 4 valores FP64. Por ello, el código procesa los elementos del vector en bloques de 4 doubles (bucle en la línea 6) e incluye un código residual (líneas 17–18) para gestionar los elementos restantes cuando el tamaño del problema no es un múltiplo entero de 4.
- SVE2 y RVV implementan extensiones vectoriales independientes de la longitud vectorial: ambas emplean un bucle (línea 7) que procesa los elementos en bloques cuyo tamaño se determina en tiempo de ejecución mediante las intrínsecas correspondientes. En SVE2 esto se realiza a través de una máscara (línea 9), mientras que en RVV se hace mediante la configuración de la longitud vectorial (línea 9).
- Además de ser agnóstico a la arquitectura, GHW es (*i*) agnóstico a la longitud del vector y (*ii*) agnóstico al tipo de datos. De manera similar a Arm SVE2, la propiedad (*i*) se logra declarando un tipo de vector escalable cuya anchura subyacente se determina en tiempo de ejecución (línea 3); consultando la longitud del vector de hardware en elementos sin asumir un número fijo de carriles (*lanes*) (línea 17); y enmascarando dinámicamente los elementos restantes para que el mismo código funcione para cualquier longitud de vector de hardware (línea 9). La propiedad (*ii*) se logra definiendo una etiqueta (*tag*) para el tipo de datos (línea 2), lo que permite que la misma estructura de bucle opere sobre cualquier tipo soportado por GHW simplemente cambiando la etiqueta.

Las implementaciones vectoriales de AXPY basadas en SVE2, RVV 1.0 y GHW podrían optimizarse siguiendo la misma estrategia que en AVX2: un bucle inicial que opera sobre bloques de ancho de vector completo, seguido de un bucle final que gestione los elementos restantes. Se ha optado por no adoptar ese enfoque con el fin de destacar con mayor claridad las diferencias en la filosofía de programación de cada ISA vectorial.

En la Tabla I se resumen los tipos de datos vectoriales y las intrínsecas más comunes a modo de comparativa final.

III. Núcleos vectoriales para álgebra lineal dispersa

En esta sección, se presentan nuestras implementaciones vectoriales para SpMV utilizando GHW. Nuestro interés en este núcleo es doble. Primero, sus patrones de acceso indirecto a memoria dificultan la utilización directa de SIMD, por lo que la mayoría de las implementaciones de alto rendimiento dependen de optimizaciones específicas del hardware en lugar de abstracciones portables como GHW. Segundo, SpMV constituye un componente central para muchos algoritmos de análisis de datos y computación científica en general, y para la resolución iterativa de sistemas lineales dispersos en particular.

A continuación, se revisan varios esquemas de almacenamiento de matrices dispersas que destacan por su flexibilidad, generalidad y eficiencia en memoria. Junto a ellos, se presentan sus núcleos asociados para SpMV $y = Ax$, donde A es una matriz dispersa de $m \times n$ que contiene $n_z(\text{nz})$ elementos no nulos, mientras que x e y son vectores de dimensiones compatibles.

A. Formatos de almacenamiento de matrices dispersas y SpMV

CSR. Este formato proporciona una representación compacta de matrices dispersas, facilitando el acceso secuencial por filas durante la ejecución del núcleo SpMV para optimizar la localidad de datos en memoria. En este diseño, los elementos no nulos se almacenan siguiendo un orden por filas (*row-major*) en el vector `values`. Además, CSR emplea el vector `col_idx` para los índices de columna de las entradas no nulas y un vector `row_ptr` que consta de $m + 1$ enteros; estos últimos delimitan las posiciones de inicio y fin de cada fila dentro de los vectores `values` y `col_idx`, ambos de longitud n_z. En la parte superior de la Figura 2 se ilustra un ejemplo del formato CSR a partir de una matriz dispersa de 4×4.

ELLPACK. Este formato rellena todas las filas para que cada una contenga exactamente $n_{zr}(= \text{nzr})$ entradas, donde n_{zr} denota el número máximo de

[2] `https://www.netlib.org/blas/`

```x86-64 AVX2
// AXPY using x86-64 AVX2
__m256d va, vx, vy;
// Broadcast alpha to all lanes
va = __mm256_set1_pd(alpha);

size_t i = 0; while (i < (n - 3)) {
  // Load x and y
  vx = __mm256_load_pd(&x[i]);
  vy = __mm256_load_pd(&y[i]);
  // y = alpha * x + y
  vy = __mm256_fmadd_pd(va, vx, vy);
  // Store result
  __mm256_store_pd(&y[i], vy);
  i += 4; // Advance by vlen
}
// Scalar tail handling
for (; i<n; ++i)
  y[i] += alpha*x[i];
```

(a) x86-64 AVX2

```Arm SVE2
// AXPY using Arm SVE2
svfloat64_t va, vx, vy;
// Broadcast alpha to all lanes
va = svdup_f64(alpha);


size_t i = 0; while (i < n) {
  // Mask active vector lanes
  svbool_t mask = svwhilelt_b64(i,n);
  // Load x and y
  vx = svld1(mask, &x[i]);
  vy = svld1(mask, &y[i]);
  // y = alpha * x + y
  vy = svmla_f64_x(mask, vy, va, vx);
  // Store result
  svst1(mask, &y[i], vy);
  i += svcntd(); // Advance by vlen
}
```

(b) Arm SVE2

```RISC-V RVV 1.0
// AXPY using RISC-V RVV 1.0

vfloat64m1_t va, vx, vy;
// Broadcast alpha to all lanes
va = __riscv_vfmv_vf_f64m1(alpha,
                  __riscv_vsetvl_e64m1(n));
int i = 0; while (i < n) {
  // Set active vector length
  size_t vl = __riscv_vsetvl_e64m1(n-i);
  // Load x and y
  vx = __riscv_vle64_v_f64m1(&x[i], vl);
  vy = __riscv_vle64_v_f64m1(&y[i], vl);
  // y = alpha * x + y
  vy = __riscv_vfmacc_vv_f64m1(vy,va,vx,vl);
  // Store result
  __riscv_vse64_v_f64m1(&y[i], vy, vl);
  i += vl;  // Advance by vlen
}
```

(c) RISC-V RVV 1.0

```GHW
// AXPY using Highway
const ScalableTag<double> d;
Vec<decltype(d)> va, vx, vy;
// Broadcast alpha to all lanes
va = Set(d, alpha);

int i = 0; while (i < n) {
  // Mask active vector lanes
  const auto mask = FirstN(d, n-i);
  // Load x and y
  vx = MaskedLoad(mask, d, &x[i]);
  vy = MaskedLoad(mask, d, &y[i]);
  // y = alpha * x + y
  vy = MaskedMulAdd(vx, va, vy);
  // Store result
  BlendedStore(vy, mask, d, &y[i]);
  i += Lanes(d); // Advance by vlen
}
```

(d) GHW

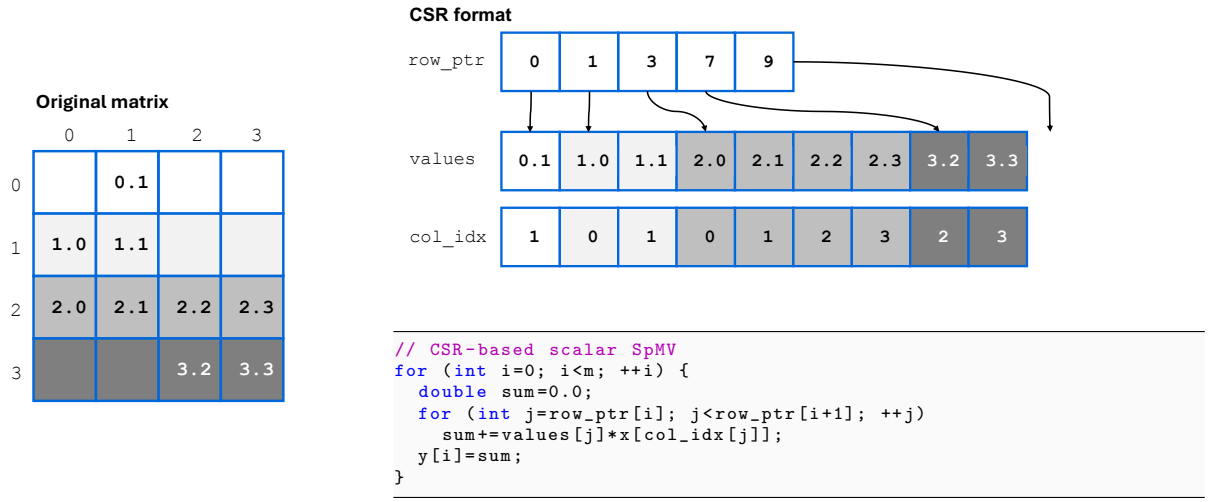
Fig. 1: Núcleo vectorial AXPY usando x86-64 AVX2 (arriba izquierda), Arm SVE2 (arriba derecha), RISC-V RVV 1.0 (abajo izquierda) y GHW (abajo derecha).

Tabla I: Correspondencia de tipos vectoriales (arriba) e intrínsecas vectoriales (abajo) en GHW con x86-64 AVX2, Arm Neon, Arm SVE2 y RISC-V RVV 1.0. *Todos los nombres de intrínsecas vectoriales incluyen el prefijo `__mm256_`.+Todos los nombres de intrínsecas vectoriales incluyen el prefijo `__riscv_`. †Emulado.

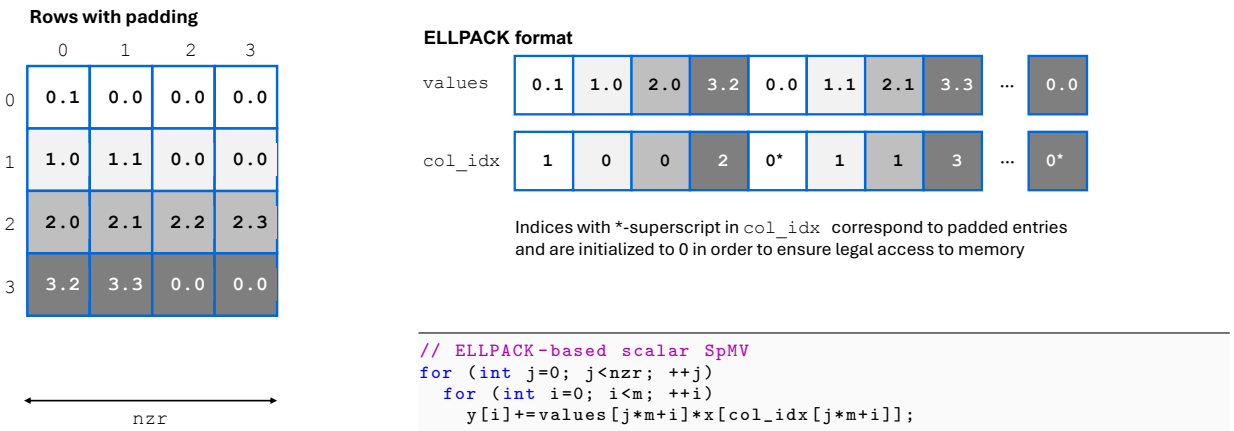
Operación	**AVX2***	**Neon**	**SVE2**	**RVV 1.0+**	**GHW**
Vector fp64	`__m256d`	`float64x2_t`	`svfloat64_t`	`vfloat64m1_t`	`Vec<d>`
Vector int64	`__m256i`	`int64x2_t`	`svint64_t`	`vint64m1_t`	`Vec<int64_t>`
Tipo predicado/ máscara	`__m256d mask`	`uint64x2_t`†	`svbool_t`	`vbool64_t`	`Mask<d>`
Ajuste vlen/ gestión del tail	–	–	`svwhilelt_b64`	`vsetvl_e64m1`	`FirstN()`
Inic. vector	`setzero_pd`	`vdupq_n_f64`	`svdup_f64`	`vfmv_v_f_f64m1`	`Zero`
Load (fp64)	`loadu_pd`	`vld1q_f64`	`svld1`	`vle64_v_f64m1`	`Load`
Store (fp64)	`storeu_pd`	`vst1q_f64`	`svst1`	`vse64_v_f64m1`	`Store`
Gather (fp64)	`i64gather_pd`	–	`svld1_gather_s64index`	`vloxei64_v_f64m1`	`Gather`
FMA	`fmadd_pd`	`vfmaq_f64`	`svmla_f64_x`	`vfmacc_vv_f64m1`	`MulAdd`
Reducción	–	–	`svaddv`	–	`ReduceSum`

elementos no nulos en cualquier fila de la matriz dispersa A. La matriz se representa entonces mediante dos vectores: (i) `values` de tamaño $m \times n_{zr}$ que almacena los valores no nulos de la matriz junto con el relleno; y (ii) `col_idx`, que es un vector de las mismas dimensiones que contiene los índices de columna correspondientes a las entradas en `values`. La parte inferior de la Figura 2 ilustra este diseño para la misma matriz de ejemplo 4×4 utilizada anteriormente para el formato CSR, y también presenta la implementación correspondiente para SpMV basada en ELLPACK. En nuestro caso, los vectores lógicos 2D se almacenan físicamente en memoria en orden por columnas (*column-major*). La rutina recorre sus entradas en ese orden y, por lo tanto, con salto unitario (*unit stride*).

sELL-*c*. Esta variante particiona la matriz en fran-

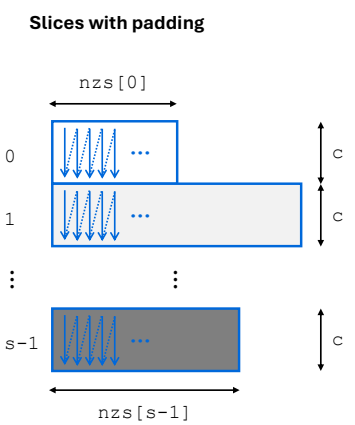


(a) Matriz dispersa 4×4 de ejemplo (izquierda) y formato CSR con el núcleo SpMV basado en CSR (derecha). Los ceros se omiten en la matriz original.



(b) Filas con relleno de la matriz dispersa de ejemplo (izquierda) y formato ELLPACK con el núcleo SpMV basado en ELLPACK (derecha). Por simplicidad, el vector `y` se asume inicializado a ceros.

Fig. 2: Formatos de matrices dispersas y núcleos SpMV asociados.



(a) Rebanadas con relleno de una matriz dispersa, con las entradas de los arrays `values` y `col_idx` almacenadas en orden *column-major*.

```
// sELL-c-based scalar SpMV
int offset=0;
// Loop over matrix slices: s = m/c
for (int i=0; i<s; i++) {
  // Loop over slice columns
  for (int j=0; j<nzs[i]; ++j)
    // Loop over slice rows
    for (int k=0; k<c; ++k)
       y[i*c+k]+=values[offset+j*c+k]
                *x[col_idx[offset+j*c+k];
  offset+=nzs[i]*c;
}
```

(b) Núcleo SpMV basado en sELL-c. Por simplicidad, `y` se asume inicializado a ceros.

Fig. 3: Formato sELL-c para matrices dispersas y núcleo SpMV asociado.

jas o bloques de c filas consecutivas, y cada franja de la matriz se convierte individualmente a ELLPACK.

```
1  // CSR-based vector SpMV using Highway
2  const ScalableTag<double> d;
3  Vec<decltype(d)> vy, vvalues, vx;
4  const int LANES = Lanes(d);
5  const Rebind<int32_t, decltype(d)> di32;
6  Vec<decltype(di32)> vcol32;
7  const Rebind<int64_t, decltype(d)> di64;
8  Vec<decltype(di64)> vcol64;
9
10 for (int i=0; i<m; i++) { // Rows loop
11   int start_idx = row_ptr[i];
12   int end_idx   = row_ptr[i+1];
13   int elems     = end_idx-start_idx;
14   vy            = Zero(d);
15   int limit     = (elems/LANES)*LANES;
16   int j         = 0;
17   for (; j<limit; j+=LANES) {// Row cols loop
18     // Load values
19     vvalues = Load(d, &values[start_idx+j]);
20     // Load column indices
21     vcol32  = Load(di32, &col_idx[start_idx+j]);
22     // Promote to int64
23     vcol64  = PromoteTo(di64, vcol32);
24     // Gather elements of x
25     vx      = GatherIndex(d, x, vcol64);
26     // Fused multiply-add
27     vy      = MulAdd(vvalues, vx, vy);
28   }
29   // Reduce accumulator
30   double sum = ReduceSum(d, vy);
31   // Scalar tail handling
32   for (; j<elems; j++)
33     sum += values[start_idx+j]*
34         x[col_idx[start_idx+j]];
35   // Store row result
36   y[i]=sum;
37 }
```

Fig. 4: Núcleo SpMV vectorial basado en CSR usando GHW.

```
1  // sELL-c-based vector SpMV using Highway
2  const ScalableTag<double> d;
3  Vec<decltype(d)> vy, vvalues, vx;
4  const int LANES = Lanes(d);
5  const Rebind<int32_t, decltype(d)> di32;
6  Vec<decltype(di32)> vcol32;
7  const Rebind<int64_t, decltype(d)> di64;
8  Vec<decltype(di64)> vcol64;
9
10 int off=0;
11 for (int i=0; i<s; ++i) { // matrix slices loop
12   vy = Zero(d);
13   for (int j=0; j<nzs[i]; ++j) {//slice cols loop
14     // Load values
15     vvalues  = Load(d, &values[off+j*LANES]);
16     // Load column indices
17     vcol32   = Load(di32, &col_idx[off+j*LANES]);
18     // Promote to int64
19     vcol64   = PromoteTo(di64, vcol32);
20     // Gather elements of x
21     vx       = GatherIndex(d, x, vcol64);
22     // Fused multiply-add
23     vy       = MulAdd(vvalues, vx, vy);
24   }
25   // Update offset
26   offset+=nzs[i]*LANES;
27   // Store slice result
28   Store(vy, d, &y[i*LANES]);
29 }
```

Fig. 5: Núcleo SpMV vectorial basado en sELL-c usando GHW.

Esto permite que diferentes franjas operen con distintos parámetros n_{zr}, lo que reduce la sobrecarga de relleno (*padding overhead*). Las entradas de los vectores `values` y `col_idx` para cada franja se almacenan en formato ELLPACK y, en nuestro caso, se organizan en orden por columnas. Asumiremos en adelante que el número de filas de la matriz m es un múltiplo entero de la dimensión de fila de la franja c, y denotaremos el número de franjas como $s = m/c$. Además, consideraremos que las dimensiones de la i-ésima franja con relleno vienen dadas por $c \times n^i_{zs}$, donde n^i_{zs}(=`nzs[i]`) es el número máximo de no nulos de cualquiera de las filas de esa franja. La Figura 3 ilustra las franjas con relleno para los vectores `values` y `col_idx` junto con una implementación práctica de SpMV basada en sELL-c.

B. SpMV *vectorial utilizando GHW*

Se presentan a continuación las implementaciones de SpMV para los formatos CSR y sELL-c utilizando GHW para asegurar la portabilidad entre hardware, longitudes de vector y tipos de datos.

CSR Vectorial. La rutina de la Figura 4 vectoriza la SpMV a través de la dimensión de columna de cada fila. Concretamente, el bucle vectorial (línea 17) procesa las columnas de la fila en bloques de `Lanes(d)` elementos (que coincide con la longitud del vector). Dentro de este bucle, debido al almacenamiento por filas, las entradas de los vectores `values` y `col_idx` se recuperan de la memoria mediante cargas vectoriales (*vector loads*) (líneas 19 y 21). Los índices se promocionan de enteros `int32` a `int64` (línea 23), ya que los vectores de 32 y 64 bits tienen diferentes disposiciones de carriles (*lanes*). A continuación, las entradas del vector `x` se recolectan (*gather*) de la memoria en el registro vectorial `vx` (línea 25); se multiplican con las entradas de la matriz en el registro vectorial `vvalues` y se acumulan en el registro vectorial `vy` (línea 27). Después del bucle, los contenidos de `vy` se reducen (línea 20) y, si la fila contiene un número de no nulos que no es un múltiplo entero del tamaño del lane, un código escalar de residuo completa el procesamiento de la fila (líneas 32–34). Esta discusión expone que la efectividad de la vectorización para CSR está ligada al número de no nulos por fila que, para problemas dispersos, suele ser bastante pequeño.

sELL-c Vectorial. La Figura 5 muestra nuestra implementación de SpMV basada en sELL-c utilizando GHW. La rutina procesa y vectoriza la carga de trabajo a través de las columnas de cada franja de filas (bucle en la línea 13), con c(=`Lanes(d)`) elementos por columna. Dentro de este bucle, debido al almacenamiento por columnas dentro de la franja de la matriz, las entradas de `values` y `col_idx` se recuperan mediante cargas vectoriales (líneas 15 y 17); los índices se promocionan de enteros `int32` a `int64` (línea 19); las entradas de `x` se recolectan de la memoria en `vx` (línea 21); se multiplican con las entradas de la matriz en `vvalues` y se acumulan en `vy` (línea 23). Después del bucle, los contenidos de `vy` se escriben de nuevo en memoria (línea 28).

C. Discusión de otros núcleos vectoriales

Además de la SpMV, la resolución iterativa de sistemas lineales dispersos requiere otros núcleos fundamentales, como AXPY (Sección II), el producto escalar y la aplicación del precondicionador [5]. Al igual que AXPY, el producto escalar es una operación BLAS y puede implementarse de forma directa

en GHW. Por el contrario, los precondicionadores abarcan un amplio espectro de complejidad [5], que va desde esquemas simples, como el escalado de Jacobi, hasta métodos sofisticados, como el multigrid algebraico (AMG) de ciclo en V, cada uno de los cuales plantea distintos desafíos de vectorización.

IV. Evaluación del rendimiento

Esta sección compara el rendimiento de las rutinas vectoriales que utilizan GHW con sus contrapartes específicas de hardware en x86-64 AVX2, Arm NEON, Arm SVE2 y RISC-V RVV 1.0, así como con la autovectorización del compilador. Los códigos desarrollados en este trabajo y utilizados en la evaluación están disponibles públicamente en `https://github.com/martineh/SpMV`.

Se reconoce que tanto AMD como Arm ofrecen bibliotecas de álgebra lineal especializadas que incluyen núcleos (*kernels*) optimizados para SpMV. No obstante, la dependencia de bibliotecas como AOCL (AMD) o ArmPL compromete la portabilidad, la cual constituye el enfoque principal de este trabajo. El estándar Sparse BLAS dista aún de una estandarización efectiva, y estas bibliotecas suelen exponer interfaces y formatos de matriz heterogéneos. En particular, ArmPL emplea un esquema de tipo *inspector-executor* que analiza el patrón de dispersión de la matriz y puede reorganizar internamente los datos en un formato opaco y propietario durante una fase de preprocesamiento. Comparar nuestras implementaciones de SpMV con núcleos que dependen de formatos optimizados *ad hoc* no permitiría, por tanto, una comparativa equitativa. Además, la sobrecarga asociada a dicho preprocesamiento debería evaluarse en el contexto de la ejecución completa de un resolvedor iterativo, mientras que nuestro estudio aisla y examina el rendimiento del núcleo SpMV de forma individual. Por estas razones, se han excluido las bibliotecas de los fabricantes de nuestra evaluación.

A. Configuración experimental

Los experimentos presentados en este trabajo se realizaron utilizando aritmética de doble precisión IEEE en los sistemas comerciales descritos en la Tabla II.

Aunque todas las plataformas cuentan con procesadores multinúcleo, este trabajo se centra exclusivamente en la vectorización. Para evitar que los efectos del equilibrado de la carga de trabajo oculten los resultados, todos los experimentos se restringieron a un solo núcleo. En el enfoque más sencillo, añadir una directiva `#pragma omp parallel for` al bucle más externo de los algoritmos de las Figuras 4 y 5, posiblemente combinada con una estrategia básica de equilibrado de carga, es suficiente. Dado que este bucle procesa las filas del vector y, no se producen condiciones de carrera. Un enfoque más sofisticado podría, en su lugar, distribuir la carga de trabajo de modo que cada hilo o núcleo procese aproximadamente el mismo número de elementos no nulos.

Los compiladores utilizados fueron `gcc` (Ubuntu 8.4.0-1ubuntu1 18.04) 8.4.0 en los dos primeros sistemas; `g++` (Ubuntu 13.3.0-6ubuntu2 24.04) 13.3.0 en el NVIDIA DGX Spark; y `gcc` 13.2.0 (Ubuntu 13.2.0-4ubuntu3-bb2) en el BananaPi F3.

Para la evaluación experimental de los núcleos SpMV, seleccionamos 159 matrices dispersas de la *SuiteSparse Matrix Collection* [15]. Las matrices objetivo tienen dimensiones que superan las 100,000 tanto en filas como en columnas y provienen de una variedad de aplicaciones científicas y de ingeniería, excluyendo problemas relacionados con grafos. Cada núcleo se ejecutó repetidamente durante al menos 5 segundos, y los resultados reportados representan valores promediados en el tiempo.

B. Evaluación y discusión de SpMV

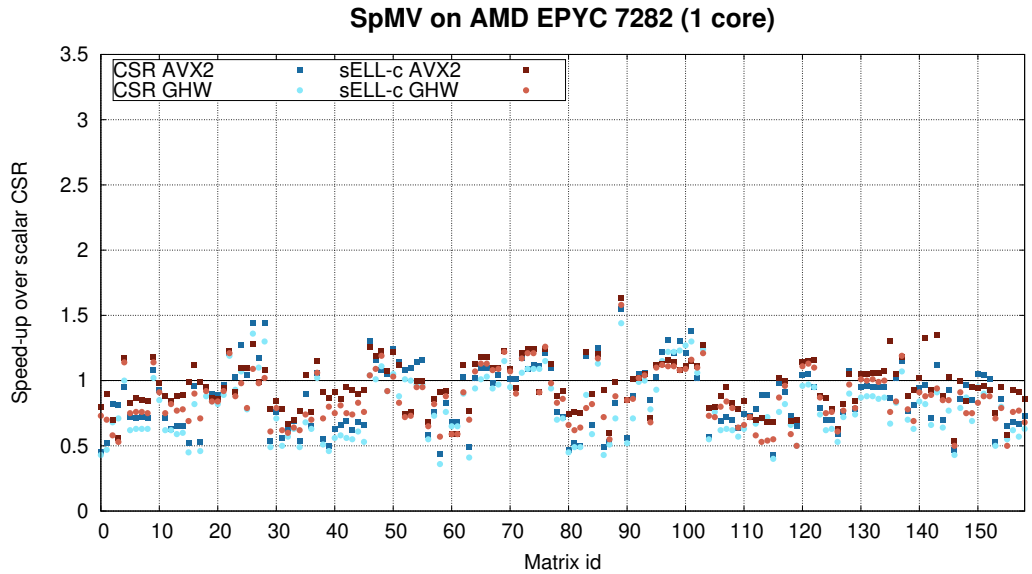
Las Figuras 6 y 7 muestran la aceleración (*speed-up*) lograda por las versiones de SpMV basadas en CSR y sELL-c, vectorizadas utilizando las intrínsecas "nativas" de cada plataforma (AVX2 en AMD EPYC 7282, Arm NEON en NVIDIA Carmel, etc.) o las intrínsecas de GHW. La Figura 8 resume estos resultados en forma de un único diagrama de caja por plataforma. Esta figura también incluye los resultados obtenidos con la autovectorización del compilador aplicada a las versiones escalares de SpMV basadas en CSR y sELL-c. En todos los casos, la aceleración se calcula con respecto al SpMV escalar basado en CSR mostrado en la Figura 2 (arriba), compilado con `-O3 -fno-vectorize` o `-O3 -fno-tree-vectorize`, dependiendo de la plataforma.

El impacto en el rendimiento fluctúa según la plataforma y el formato empleado. Por ello, a continuación se analizan las diferencias globales entre el uso de intrínsecas nativas y GHW a partir de los *speedups* promedio de la Figura 8:

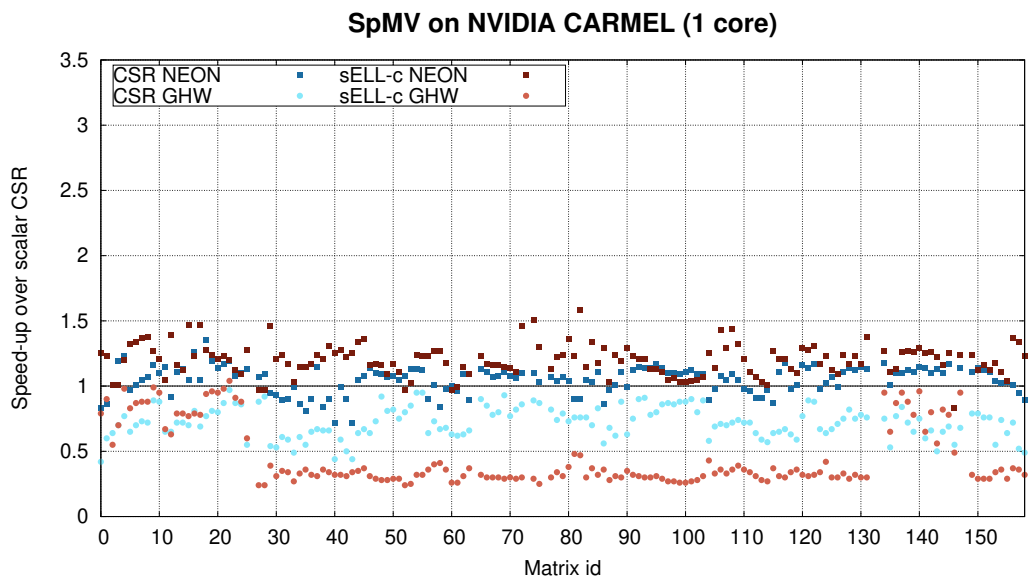
- AMD EPYC 7282: Los núcleos vectorizados mediante GHW presentan un rendimiento inferior al de sus homólogos nativos AVX2. Al inspeccionar el código ensamblador generado, se observa que la principal discrepancia radica en que GHW impone un sistema de tipos más estricto, promocionando los índices de columna de `int32` a `int64` de forma previa a la operación *gather*. Por el contrario, la versión nativa mantiene los índices como `int32`. Esta limitación es compleja de resolver en GHW, ya que su diseño prioriza la portabilidad entre arquitecturas (como ARM SVE) y evita depender de capacidades específicas de la ISA AVX2, tales como la ejecución de operaciones *gather* con vectores de índices de 32 bits. Asimismo, la planificación de instrucciones (*instruction scheduling*) en la versión GHW genera una cadena de dependencias más extensa: la promoción de índices, el *gather* y la FMA se serializan de manera más rigurosa, lo que reduce la capacidad de ocultar latencias en comparación con la implementación nativa.
- NVIDIA Carmel: En esta plataforma, GHW también muestra un rendimiento inferior a las implementaciones basadas en intrínsecas nativas

Tabla II: Características hardware de los sistemas evaluados. La longitud de vector (Vlen) se expresa en bits.

Sistema	CPU	#Cores	Caché L2 ‖Caché L3	ISA	Vlen
AMD server	AMD EPYC 7282	32	512 KB ‖ 64 MB por socket	AVX2	256
NVIDIA Jetson Xavier	NVIDIA Carmel	8	2 MB por 2 cores ‖ 4 MB compartida	NEON	128
NVIDIA DGX Spark	Arm Cortex-X925	10	2 MB por core ‖ 16 MB	SVE2	128
BananaPi F3	SpaceMiT K1	8	512 KB por 4 cores ‖ –	RVV 1.0	256



(a) AMD EPYC 7282.

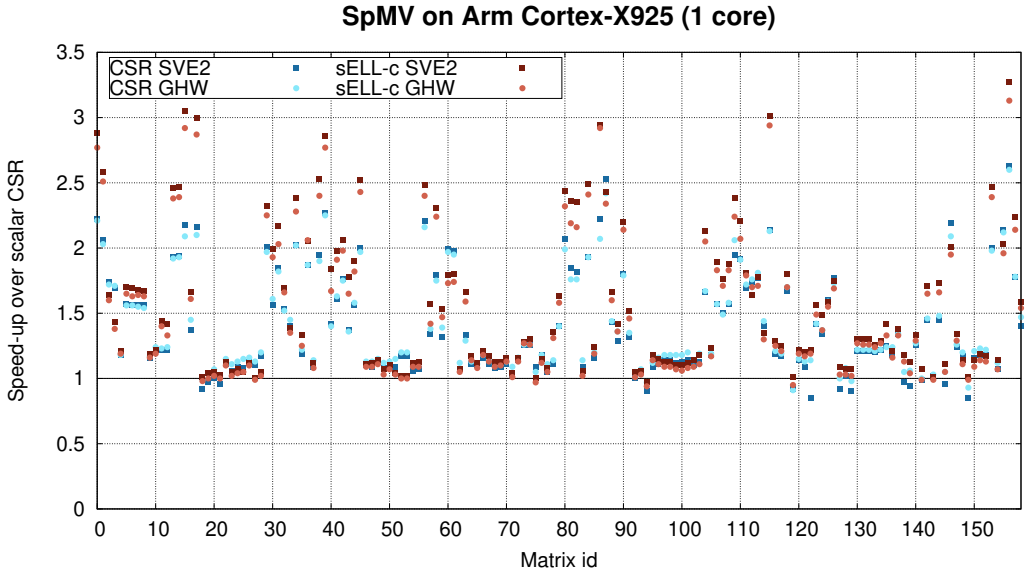


(b) NVIDIA Carmel.

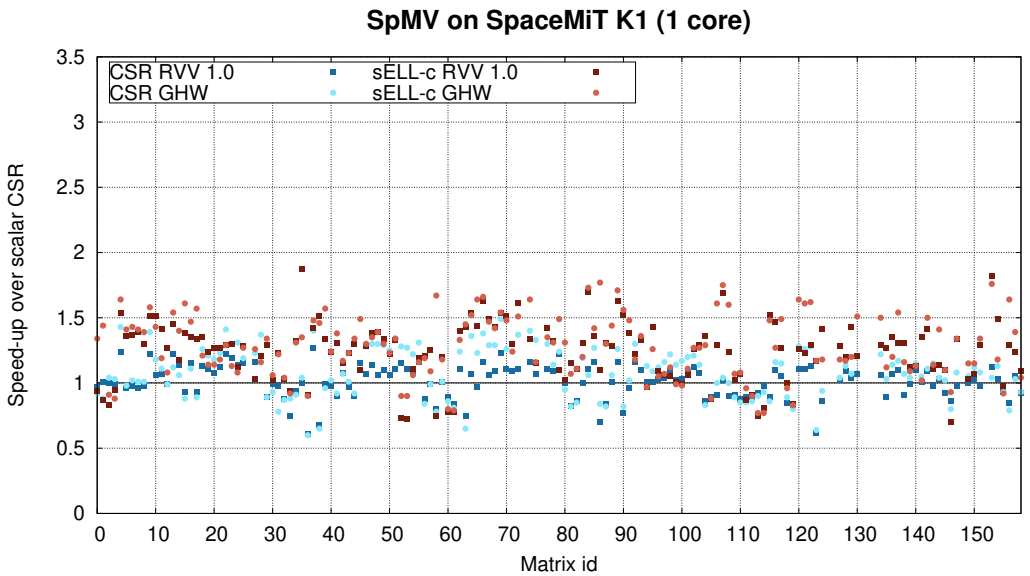
Fig. 6: Speed-up del SpMV vectorial basado en CSR y en sELL-*c* sobre el SpMV escalar basado en CSR en AMD EPYC 7282 (arriba) y NVIDIA Carmel (abajo).

de NEON. Esto se debe a que NEON no dispone de soporte para operaciones *gather*. En consecuencia, la intrínseca `GatherIndex` de GHW se traduce en un código ensamblador que carga cada índice de columna de forma individual para luego combinarlos en un registro vectorial. Este enfoque resulta altamente ineficiente y, además, la implementación generada por GHW provoca desbordamiento de registros (*register spilling*). Por el contrario, nuestra rutina nativa optimiza la carga manual aprovechando que un registro vectorial de 128 bits puede albergar dos valores FP64. Esta deficiencia podría mitigarse especializando la implementación de `GatherIndex` en GHW para Arm NEON (y potencialmente para tipos de datos específicos). Si se realiza con precisión, esta estrategia preservaría la portabilidad a costa de reducir la generalidad del marco de trabajo.

- Arm Cortex-X925: En este procesador, el escenario es más favorable para el enfoque agnóstico de GHW. Su rendimiento es equiparable o incluso ligeramente superior al de las rutinas nativas SVE2, dependiendo de si el formato de almace-



(a) Arm Cortex-X925.



(b) SpaceMiT K1.

Fig. 7: Speed-up de SpMV vectorial (CSR y sELL-c) frente a la ejecución escalar (CSR) en los procesadores Arm Cortex-X925 (arriba) y SpaceMiT K1 (abajo).

namiento empleado es CSR o sELL-c, respectivamente.

- SpaceMiT K1: Esta plataforma constituye el escenario óptimo para GHW, logrando superar a los códigos nativos RVV en ambos diseños de almacenamiento disperso.

C. *Integración y evaluación en bibliotecas de álgebra lineal dispersa*

Se han integrado los núcleos SpMV basados en CSR y sELL-c, desarrollados con intrínsecas de GHW, en las bibliotecas PETSc y Ginkgo mediante la sustitución directa de sus variantes escalares y el ajuste de los parámetros correspondientes. Este procedimiento se amplió a diversas operaciones de álgebra lineal básica, tales como AXPY, producto escalar, cálculo de normas y un precondicionador de Jacobi elemental. Tras ejecutar el resolvedor de Gradiente Conjugado (CG) precondicionado en estas nuevas instancias de las bibliotecas para distintos casos de prueba en las cuatro plataformas de evaluación, se observaron factores de aceleración consistentes con el coste computacional de cada núcleo y su respectiva ganancia SIMD. Por motivos de brevedad, estos resultados detallados se han omitido en el presente artículo.

V. Conclusiones

Las conclusiones de este estudio se estructuran en torno a cuatro cuestiones fundamentales:

¿Permite GHW el desarrollo de implementaciones de SpMV portables?
Sí. Las rutinas vectoriales desarrolladas para las variantes de SpMV (basadas en CSR y sELL-c) mediante GHW se compilaron y ejecutaron con éxito en las cuatro plataformas de evaluación sin necesi-

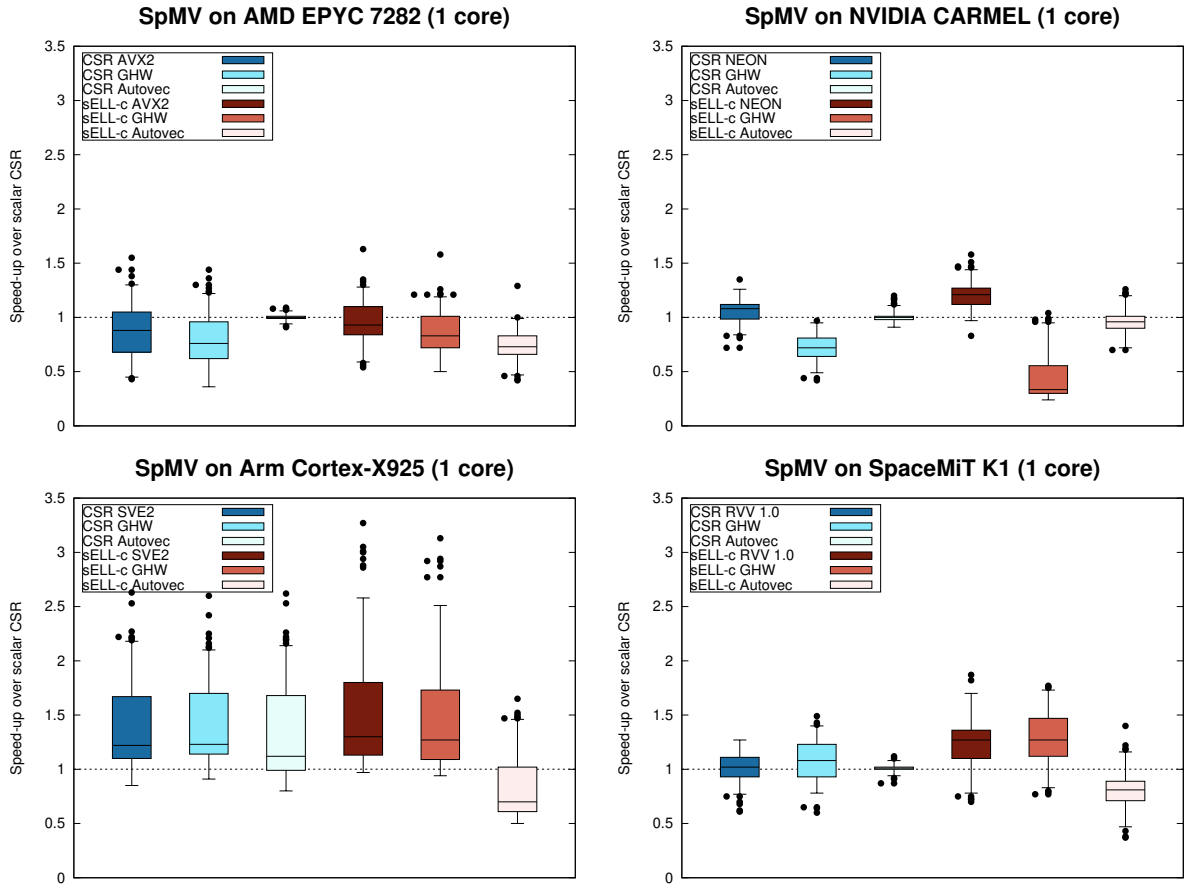


Fig. 8: Speed-up de SpMV vectorial (CSR y sELL-c) frente a la ejecución escalar (CSR) en las cuatro plataformas.

dad de modificaciones en el código fuente.

¿Puede GHW servir como base para bibliotecas portables de álgebra lineal dispersa?
Es altamente probable. Aunque no se ha implementado la funcionalidad completa de una biblioteca de producción, los núcleos desarrollados —SpMV y las primitivas para AXPY, producto escalar y cálculo de normas— constituyen una base sólida. Si bien ciertos núcleos avanzados, como la aplicación de precondicionadores complejos, requerirán un desarrollo adicional, no se han identificado obstáculos conceptuales que impidan la adopción de GHW para este propósito.

¿Garantiza GHW un alto rendimiento manteniendo la portabilidad?
Resultados heterogéneos. Se ha constatado una degradación del rendimiento en las arquitecturas AVX2 y Arm NEON, mientras que en Arm SVE2 y RVV 1.0 se alcanzó un rendimiento (*throughput*) competitivo o incluso superior a las versiones nativas. Las causas de la pérdida de eficiencia en las dos primeras ISAs han sido identificadas y, en el caso de NEON, podrían mitigarse mediante una especialización del código, aunque a costa de reducir la generalidad del marco de trabajo. En el ecosistema Arm, dado que NEON representa una tecnología heredada (*legacy*) frente al estándar actual SVE2, este soporte limitado no se considera crítico. Respecto a la extensión emergente SME, los beneficios de una ISA orientada a matrices para el álgebra lineal dispersa son aún inciertos, dada la naturaleza irregular y limitada por memoria (*memory-bound*) de estas cargas de trabajo.

¿Existen otras alternativas para el álgebra lineal dispersa portable?
Sí, aunque con compromisos. La autovectorización, donde el compilador traduce código escalar a instrucciones vectoriales, suele ofrecer un rendimiento subóptimo según nuestros resultados. Por otro lado, la propuesta `std::simd` de C++ busca una abstracción vectorial agnóstica similar; sin embargo, GHW proporciona un control explícito de bajo nivel que generalmente se traduce en una mayor eficiencia. Finalmente, infraestructuras como MLIR [16] ofrecen mecanismos para generar código vectorial portable, pero operan en un nivel distinto de la pila de software, centrándose en representaciones intermedias y transformaciones sistemáticas. Por tanto, MLIR y GHW abordan problemáticas complementarias pero diferenciadas.

Agradecimientos

Este trabajo ha sido financiado por los proyectos PID2023-146569NB-C2 del MCIN/AEI/10.13039/501100011033 y CIPROM/2022/20 de la Generalitat Valenciana. Asimismo, ha recibido financiación del proyecto DARE SGA1, a través de la Empresa Común Europea de Computación de Alto Rendimiento (EuroHPC JU) en virtud del Acuerdo de Subvención n.º 101202459,

y del proyecto PCI2024-161687-3, financiado por el MICIU/AEI/10.13039/501100011033 y el mecanismo "NextGenerationEU"/PRTR de la Unión Europea. La EuroHPC JU cuenta con el respaldo del programa de investigación e innovación Horizonte Europa de la Unión Europea y de España, Alemania, Chequia, Italia, Países Bajos, Bélgica, Finlandia, Grecia, Croacia, Portugal, Polonia, Suecia, Francia y Austria.

Referencias

[1] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 5 edition, 2011.

[2] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Steven Benson, Jed Brown, Peter Brune, Kris Buschelman, Emil M. Constantinescu, Lisandro Dalcin, Alp Dener, Victor Eijkhout, Jacob Faibussowitsch, William D. Gropp, Václav Hapla, Tobin Isaac, Pierre Jolivet, Dmitry Karpeev, Dinesh Kaushik, Matthew G. Knepley, Fande Kong, Scott Kruger, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Lawrence Mitchell, Todd Munson, Jose E. Roman, Karl Rupp, Patrick Sanan, Jason Sarich, Barry F. Smith, Stefano Zampini, Hong Zhang, Hong Zhang, and Junchao Zhang, "PETSc Web page," `https://petsc.org/`, 2026.

[3] Hartwig Anzt, Terry Cojean, Goran Flegar, Fritz Göbel, Thomas Grützmacher, Pratik Nayak, Tobias Ribizel, Yuhsiang Mike Tsai, and Enrique S. Quintana-Ortí, "Ginkgo: A Modern Linear Operator Algebra Framework for High Performance Computing," *ACM Transactions on Mathematical Software*, vol. 48, no. 1, pp. 2:1–2:33, February 2022.

[4] The Trilinos Project Team, *The Trilinos Project*, Linux Foundation / High Performance Software Foundation, 2026, Accessed: January 8, 2026.

[5] Y. Saad, *Iterative methods for sparse linear systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 3rd edition, 2003.

[6] Nathan Bell and Michael Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *Proc. Conf. on High Performance Computing Networking, Storage and Analysis*, 2009, SC '09, pp. 18:1–18:11.

[7] Moritz Kreutzer et al., "A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units," *SIAM J. Scientific Computing*, vol. 36, no. 5, pp. C408–C432, 2014.

[8] Haodong Bian et al., "CSR2: A new format for SIMD-accelerated SpMV," in *Proc. 20th IEEE/ACM Int. Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, 2020, pp. 350–359.

[9] Aydın Buluç et al., "Reduced-bandwidth multithreaded algorithms for sparse matrix-vector multiplication," in *Proc. IEEE Int. Parallel & Distributed Processing Symposium (IPDPS)*, 2011, pp. 721–733.

[10] Hoang-Vu Dang and Bertil Schmidt, "The sliced COO format for sparse matrix-vector multiplication on CUDA-enabled GPUs," *Procedia Computer Science*, vol. 9, pp. 57–66, 2012, Proc. Int. Conf. Computational Science, ICCS 2012.

[11] G. Flegar and E. S. Quintana-Ortí, "Balanced CSR sparse matrix-vector product on graphics processors," in *High-Performance Computing – ISC 2016 Workshops*, Lecture Notes in Computer Science, pp. 697–709. 2016.

[12] Max Grossman et al., "A survey of sparse matrix-vector multiplication performance on large matrices," *CoRR*, vol. abs/1608.00636, 2016.

[13] Yousef Saad, "Jagged diagonal storage (JDS) format," in *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, 2000.

[14] Samuel Williams et al., "Optimization of sparse matrix-vector multiplication on emerging multicore platforms," in *Proc. 2007 ACM/IEEE Conference on Supercomputing*, 2007, pp. 1–12.

[15] Timothy A. Davis and Yifan Hu, "The SuiteSparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, pp. 1–25, 2011.

[16] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko, "MLIR: Scaling compiler infrastructure for domain specific computation," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021, pp. 2–14.

Evaluación de la Vectorización Automática del Compilador frente a SIMD Manual para el Cálculo de Slope Entropy

Juan Seguí Moreno[1], Antonio Molina Picó[2]

Resumen— **Slope Entropy (SlopEn) es una medida de complejidad para series temporales que codifica las diferencias de amplitud entre muestras consecutivas mediante un alfabeto de cinco símbolos y calcula la entropía de Shannon sobre la distribución de frecuencias de los patrones resultantes. Su estructura algorítmica combina una fase aritmética vectorizable con una fase de búsqueda de patrones que presenta dependencias de datos. En este trabajo se comparan cuatro implementaciones del cálculo de SlopEn en C compiladas con GCC 12.2: una versión de referencia (`-O0`), una con optimizaciones estándar (`-O2`), una con vectorización automática agresiva (`-O3 -march=native -ffast-math`), y una con instrucciones SIMD AVX-512 escritas manualmente. Los experimentos sobre señales gaussianas sintéticas de longitud $N \in \{10^4, 10^5, 10^6, 10^7\}$ en un procesador Intel Core i7-10700K muestran aceleraciones de hasta $3{,}06\times$ con vectorización automática y de $4{,}03\times$ con SIMD manual. Se analiza la fase de búsqueda de patrones como cuello de botella estructural y se cuantifica su impacto mediante la ley de Amdahl.**

Palabras clave— **Slope Entropy, SIMD, AVX-512, vectorización automática, series temporales, paralelismo a nivel de instrucción.**

I. Introducción

El análisis de series temporales es una disciplina fundamental en múltiples campos científicos y tecnológicos, especialmente en biomedicina, ingeniería industrial y procesamiento de señales. Entre las herramientas disponibles para cuantificar la complejidad de una señal, las medidas de entropía han demostrado ser especialmente útiles al proporcionar una representación cuantitativa del desorden y la estructura dinámica interna de una secuencia de datos [1], [2].

En este contexto, Slope Entropy (SlopEn) [3], propuesta por Cuesta-Frau, ha demostrado una capacidad discriminativa superior a Permutation Entropy (PE) [4] y Sample Entropy (SampEn) [5] en múltiples conjuntos de datos públicos, incluyendo señales EEG, series de intervalos RR cardíacos y electromiogramas (EMG) [6], [7]. Su mecanismo central codifica la pendiente entre muestras consecutivas de una subsecuencia mediante un alfabeto de cinco símbolos $(-2, -1, 0, +1, +2)$ y calcula la entropía de Shannon sobre la distribución de frecuencias de los patrones resultantes.

[1]Departamento de Informática de Sistemas y Computadores (DISCA), Universitat Politècnica de València, Campus d'Alcoi. e-mail: `jsegui@disca.upv.es`.

[2]Departamento de Informática de Sistemas y Computadores (DISCA), Universitat Politècnica de València, Campus d'Alcoi. e-mail: `amolina@disca.upv.es`.

A medida que crece la longitud de las señales —situación habitual en monitorización continua o análisis de grandes volúmenes de datos biomédicos— el coste computacional de SlopEn se vuelve significativo. Esto motiva el estudio de su optimización mediante técnicas de paralelismo a nivel de instrucción (ILP). Los procesadores Intel modernos ofrecen instrucciones AVX-512 [8], que permiten operar simultáneamente sobre ocho valores `double` por instrucción, constituyendo una candidata natural para acelerar el cálculo.

El trabajo previo de los autores [9] comparó la implementación estándar de SlopEn frente a una versión manual con instrucciones AVX-512, identificando un umbral de rentabilidad en torno a las 20.800 muestras. El presente trabajo amplía ese análisis introduciendo la vectorización automática del compilador como nivel intermedio, respondiendo a la siguiente pregunta de investigación: *¿hasta qué punto puede GCC explotar automáticamente el potencial vectorial de SlopEn, y qué beneficio adicional aporta el SIMD manual?*

El resto del artículo se organiza del siguiente modo: la Sección II describe los fundamentos de SlopEn y de AVX-512; la Sección III detalla las cuatro implementaciones y el diseño experimental; la Sección IV presenta y discute los resultados; la Sección V recoge las conclusiones.

II. Fundamentos

A. Slope Entropy

Dada una serie temporal $\mathbf{x} = \{\mathbf{x_0}, \mathbf{x_1}, \ldots, \mathbf{x_{N-1}}\}$ de longitud N, SlopEn extrae todas las subsecuencias solapadas de longitud m. Para la subsecuencia que comienza en el índice j, se calculan $m-1$ diferencias consecutivas $d_i = x_i - x_{i-1}$ y cada una se clasifica en un símbolo del alfabeto $\{-2, -1, 0, +1, +2\}$ según los umbrales γ y δ $(\gamma > \delta > 0)$:

- $d > \gamma \;\Rightarrow\; +2$
- $\delta < d \leq \gamma \;\Rightarrow\; +1$
- $|d| \leq \delta \;\Rightarrow\; 0$
- $-\gamma \leq d < -\delta \;\Rightarrow\; -1$
- $d < -\gamma \;\Rightarrow\; -2$

La secuencia de símbolos resultante constituye un patrón de pendientes. El algoritmo mantiene una lista dinámica de todos los patrones distintos encontrados (Ψ^m) y sus frecuencias. La entropía se calcula como:

$$\mathrm{SlopEn} = -\sum_{\mathrm{k}=0}^{|\Psi^{\mathrm{m}}|-1} \mathrm{p_k} \log_2 \mathrm{p_k}, \quad \mathrm{p_k} > 0 \qquad (1)$$

donde $p_k = c_k/|\Psi^m|$ y c_k es el número de ocurrencias del patrón k. La normalización por el número de patrones *realmente observados* $|\Psi^m|$ —y no por el máximo teórico 5^{m-1}— permite explotar la información de los patrones prohibidos [3], [10]. Los parámetros por defecto son $\gamma = 1$, $\delta = 0{,}001$, $m = 3$. Cabe destacar que existen extensiones multiescala de las medidas de entropía clásicas [11] que podrían aplicarse también a SlopEn en trabajos futuros.

Desde el punto de vista computacional, el algoritmo presenta dos fases:

1. **Fase vectorizable** (complejidad $O(N)$): cálculo de las $N-1$ diferencias consecutivas y clasificación simbólica de cada una. No existen dependencias entre iteraciones.
2. **Fase secuencial** (complejidad $O(N \cdot P)$, donde $P \leq 5^{m-1}$): búsqueda del patrón actual en la lista Ψ^m y actualización de su contador. Involucra control de flujo dependiente de datos (salida anticipada al encontrar coincidencia).

Esta asimetría estructural es el eje central del análisis presentado en este trabajo.

B. Instrucciones AVX-512

AVX-512 [8] es un conjunto de instrucciones SIMD introducido por Intel que permite realizar operaciones sobre vectores de 512 bits, equivalentes a ocho valores `double` por instrucción. Entre sus características más relevantes para este trabajo destacan:

- `_mm512_sub_pd`: calcula 8 diferencias en paralelo.
- `_mm512_cmp_pd_mask`: comparación vectorizada con resultado en máscara de 8 bits, que permite clasificar simbólicamente 8 elementos sin ramificación explícita.
- `_mm512_loadu_pd` / `_mm512_storeu_pd`: carga y almacenamiento no alineado para acceso flexible a memoria.

Estas características hacen de AVX-512 una herramienta adecuada para la Fase 1 de SlopEn, mientras que la Fase 2 no es directamente vectorizable sin cambiar la estructura de datos subyacente.

III. Metodología

A. Implementaciones

Todas las implementaciones están escritas en C y compiladas con GCC 12.2. La corrección de todas las versiones se verifica contra el ejemplo del Apéndice A de [3], obteniendo SlopEn = 5,29 para la señal de prueba proporcionada. Se comparan cuatro configuraciones:

Versión de referencia (`-O0`). Implementación fiel al Algoritmo 1 de [3], compilada sin optimizaciones. Sirve como línea base para medir el coste algorítmico puro, sin interferencia del compilador.

Versión `-O2`. El mismo código compilado con `-O2`. Este nivel activa inlining de funciones, eliminación de código muerto y mejor asignación de registros, pero GCC no vectoriza automáticamente los bucles del algoritmo salvo que cumplan condiciones triviales de ausencia de *aliasing*.

Vectorización automática (`-O3 -march=native -ffast-math`). A este nivel, GCC intenta vectorizar [12], [13] cualquier bucle donde puede demostrar ausencia de *aliasing* y trip count suficiente. Se utilizó el flag `-fopt-info-vec-all` para obtener un informe detallado y la herramienta Compiler Explorer [14] para inspeccionar el código ensamblador generado de forma interactiva. Como se analiza en detalle en la Sección IV-B, GCC vectoriza con éxito el cálculo aislado de diferencias, pero *no* consigue vectorizar el bucle combinado de diferencias y clasificación simbólica debido al control de flujo introducido por las condiciones `if/else`.

SIMD manual AVX-512 (`-O2 -mavx512f -mavx512dq`). Las Fases 1 y 2 de clasificación se reimplementan usando intrínsecos AVX-512 [8], [15]. La Fase 2 de búsqueda de patrones permanece idéntica a la implementación secuencial. El fragmento central del bucle vectorizado se muestra en el Algoritmo 1.

```
/* Fase 1: diferencias con AVX-512 (8 doubles/ iter) */
__m512d vg = _mm512_set1_pd(gamma);
__m512d vng = _mm512_set1_pd(-gamma);
for (i = 0; i <= N-9; i += 8) {
    __m512d va = _mm512_loadu_pd(&x[i]);
    __m512d vb = _mm512_loadu_pd(&x[i+1]);
    __m512d vd = _mm512_sub_pd(vb, va);
    _mm512_storeu_pd(&d[i], vd);
}
/* Fase 2: clasificacion con mascaras (sin branches) */
for (i = 0; i <= N-9; i += 8) {
    __m512d vd = _mm512_loadu_pd(&d[i]);
    __mmask8 mp2 = _mm512_cmp_pd_mask(vd,vg, _CMP_GT_OQ);
    __mmask8 mn2 = _mm512_cmp_pd_mask(vd,vng, _CMP_LT_OQ);
    /* ... resto de mascaras para +1, 0, -1 ... */
}
```

Fig. 1: Núcleo SIMD AVX-512: diferencias vectorizadas (Fase 1) y clasificación simbólica sin ramificación (Fase 2).

B. Diseño experimental

Los experimentos se han realizado sobre un procesador Intel Core i7-10700K (3,8 GHz base, soporte AVX-512, 16 MB caché L3) con Linux 5.15 y GCC 12.2. La medición de tiempos se realiza con `clock_gettime(CLOCK_MONOTONIC)` con resolución de nanosegundos. Cada configuración se ejecuta 5 veces por tamaño de señal, reportando el tiempo medio y la desviación estándar.

Como señales de entrada se han utilizado series gaussianas sintéticas (media 0, varianza 1) generadas con semilla fija (42) para garantizar la reproducibilidad, siguiendo la práctica habitual en estudios de caracterización de entropía [3]. Los tamaños evaluados son $N \in \{10^4, 10^5, 10^6, 10^7\}$ muestras. Los parámetros de SlopEn son $m = 3$, $\gamma = 1{,}0$, $\delta = 0{,}001$ conforme a la configuración por defecto de [3].

IV. Resultados experimentales

A. Tiempos de ejecución y speedup

La Tabla I muestra los tiempos de ejecución medios y el speedup respecto a la versión de referencia (`-O0`) para todas las configuraciones y longitudes de señal evaluadas.

Tabla I: Tiempo de ejecución medio (ms) y speedup respecto a la versión de referencia (`-O0`). Hardware: Intel Core i7-10700K, AVX-512, GCC 12.2. Cada valor es la media de 5 repeticiones.

Config.	$N = 10^4$	$N = 10^5$	$N = 10^6$	$N = 10^7$
Tiempo de ejecución (ms)				
`-O0`	1,54	16,39	181,95	1989,30
`-O2`	0,84	8,26	92,44	1010,00
`-O3`	0,60	5,55	59,47	650,10
SIMD AVX-512	0,50	4,48	45,14	493,90
Speedup vs. referencia (`-O0`)				
`-O2`	1,84×	1,99×	1,97×	1,97×
`-O3`	2,58×	2,96×	3,06×	3,06×
SIMD AVX-512	3,07×	3,66×	4,03×	4,03×

De los resultados se extraen tres observaciones principales.

Efecto de `-O2`. La versión `-O2` ya obtiene un speedup de entre 1,84× y 1,99× respecto a la referencia. Esta mejora se atribuye principalmente al inlining de la función `classify_symbol` y a una mejor asignación de registros. El beneficio se estabiliza en torno a 2× porque la fase de búsqueda de patrones —que domina el tiempo total de ejecución— no se transforma con `-O2`.

Efecto de la vectorización automática. La vectorización automática (`-O3`) muestra una dependencia clara con el tamaño de la señal: para $N = 10^4$ el speedup es de 2,58×, limitado por el overhead de configuración de los bucles vectorizados; para $N = 10^7$ alcanza 3,06×. Este comportamiento confirma que la vectorización automática es más efectiva cuando la fracción vectorizable —el cálculo de diferencias— representa una proporción mayor del trabajo total.

Efecto del SIMD manual. El SIMD manual AVX-512 supera consistentemente a la vectorización automática en todos los tamaños, con un speedup máximo de 4,03× para $N \geq 10^6$. La ganancia adicional sobre `-O3` (entre un 19 % y un 24 % según el tamaño) se atribuye a: (1) la implementación manual evita suposiciones conservadoras de *aliasing* que impiden a GCC encadenar completamente los bucles vectorizados; (2) el enfoque basado en máscaras elimina predicciones de salto costosas que persisten en la versión auto-vectorizada.

La Figura 2 ilustra la evolución del speedup con el tamaño de la señal para las tres configuraciones optimizadas.

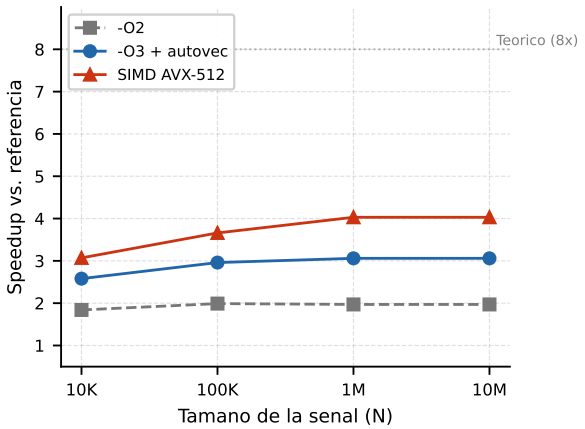


Fig. 2: Speedup respecto a la versión de referencia (`-O0`) en función del tamaño de la señal. La vectorización automática alcanza el 75 % del beneficio del SIMD manual sin modificar el código fuente.

B. Análisis del código ensamblador generado

Para entender en detalle por qué cada configuración ofrece un rendimiento distinto, se ha analizado el código ensamblador generado por GCC para el bucle interno del algoritmo, tomando como referencia una versión simplificada que combina el cálculo de diferencias y la clasificación simbólica. El análisis se ha realizado tanto localmente como con la herramienta Compiler Explorer [14], que facilita la visualización lado a lado de las distintas configuraciones.

`-O0`. El compilador genera código escalar literal, con acceso a memoria mediante operandos `rbp`-relativos para cada variable local. Cada iteración realiza dos cargas de memoria (`movsd xmm0/xmm1, QWORD PTR ...`), una resta escalar (`subsd xmm0, xmm1`) y una cadena de comparaciones (`comisd` + `jbe`) para clasificar el símbolo. No hay reutilización de registros entre iteraciones: los operandos se vuelven a cargar desde la pila en cada paso. Esto explica el coste base elevado.

`-O2`. El compilador mantiene en `xmm0` el valor de la muestra previa entre iteraciones, eliminando una carga de memoria por iteración. Las constantes de comparación (γ, δ) se cargan en registros `xmm3`–`xmm5` fuera del bucle. Sin embargo, la operación sigue siendo *escalar* (`subsd`, `comisd`), sobre registros XMM de 64 bits útiles. El *speedup* de ∼2× frente a `-O0` se explica por la reducción de accesos a memoria y la mejor planificación, no por vectorización.

`-O3 -march=native -ffast-math`. El informe `-fopt-info-vec-all` revela un comportamiento clave: GCC **no vectoriza** el bucle combinado de diferencias y clasificación. El mensaje exacto es:

```
missed: not vectorized: control flow in
loop.
```

La cadena de `if/else` para asignar el símbolo introduce control de flujo que el vectorizador de GCC no traduce eficientemente. Sin embargo, cuando el bucle de diferencias se aísla en una función independiente (sin la clasificación), GCC lo vectoriza correctamente generando `vsubpd ymm0, ymm1, [rcx+rax]` sobre registros AVX2 de 256 bits, procesando 4 doubles por iteración. Esto sugiere que la mejora observada de `-O3` sobre `-O2` (∼1,5× adicional) procede principalmente del *loop unrolling*, mejor *instruction scheduling* y reducciones de coste por operación escalar mediante prefijos VEX (`vmovsd`,

`vsubsd`) [16], no de una vectorización SIMD genuina del bucle completo.

SIMD manual AVX-512. El código intrínseco se traduce directamente a instrucciones AVX-512 sobre registros de 512 bits:

```
vmovupd zmm0, [rdi+rax*8]
vmovupd zmm1, [rdi+rax*8+8]
vsubpd zmm2, zmm1, zmm0
vmovupd [rsi+rax*8], zmm2
```

Cada iteración procesa 8 doubles en paralelo. La clasificación se implementa con instrucciones `vcmppd` que generan máscaras de 8 bits, evitando el control de flujo problemático que impedía la vectorización automática. Esta es la diferencia estructural fundamental respecto a `-O3`: el SIMD manual sustituye *branches* por operaciones predicadas mediante máscaras.

La Tabla II resume las características del código generado.

Tabla II: Características del código ensamblador generado por cada configuración para el bucle interno del algoritmo.

Config.	Registros	Anchura	Branches
`-O0`	XMM	escalar	sí
`-O2`	XMM	escalar	sí
`-O3`	XMM (VEX)	escalar (no SIMD)	sí
SIMD manual	ZMM	8 doubles	no (máscaras)

Este análisis revela un hallazgo no trivial: la diferencia de rendimiento entre `-O3` y el SIMD manual no se debe a que ambos vectoricen y el segundo lo haga mejor, sino a que **el SIMD manual vectoriza y `-O3` no** en presencia del control de flujo de la clasificación simbólica. Esto refuerza la conclusión de que la estructura del código fuente —y no solo los flags del compilador— es determinante para extraer rendimiento del hardware vectorial.

C. *Análisis del cuello de botella: la búsqueda de patrones*

Más allá de la clasificación simbólica, el bucle de búsqueda de patrones también aparece en el informe de vectorización como *no vectorizable* por razones distintas: (1) condiciones de salida dependientes de datos (*break* al encontrar coincidencia); (2) *aliasing* potencial en los accesos a la estructura de patrones; (3) la actualización del contador requiere operaciones *scatter/gather* que GCC considera no rentables.

Aplicando la ley de Amdahl [17], si la fracción vectorizable del algoritmo representa F del tiempo total, el speedup máximo alcanzable con AVX-512 de ancho 8 está acotado por:

$$S_{\text{máx}} = \frac{1}{(1-F) + F/8} \tag{2}$$

A partir de los resultados experimentales se estima $F \approx 0{,}35$ para $m = 3$, lo que produce $S_{\text{máx}} \approx 4{,}1\times$, coherente con el speedup máximo observado de $4{,}03\times$. Para valores mayores de m (e.g., $m = 5$, espacio de patrones $5^4 = 625$), la fase de búsqueda dominaría aún más y F disminuiría, reduciendo el beneficio de la vectorización. Por el contrario, si se sustituyera la lista dinámica de patrones por una tabla *hash* con acceso $O(1)$, F podría aproximarse a 1 y el speedup teórico convergiría hacia $8\times$, constituyendo la dirección de trabajo futuro más prometedora.

Estos resultados son consistentes con los del trabajo previo [9], que identificó el umbral de rentabilidad de AVX-512 manual en torno a 20.800 muestras. El presente trabajo ofrece una explicación más precisa de ese umbral mediante el análisis de la fracción vectorizable, y confirma que `-O3 -march=native` permite aproximarse al 75 % del beneficio del SIMD manual sin coste de desarrollo adicional.

V. Conclusiones

Se ha presentado una comparativa sistemática de cuatro estrategias de compilación y vectorización para el cálculo de Slope Entropy sobre hardware Intel AVX-512. Las principales conclusiones son:

1. La vectorización automática del compilador (`-O3 -march=native -ffast-math`) consigue speedups de $2{,}58\times$ a $3{,}06\times$ según la longitud de la señal, con menor beneficio para señales cortas ($N = 10^4$) debido al overhead de configuración de los bucles vectorizados.
2. El SIMD manual con AVX-512 alcanza speedups de $3{,}07\times$ a $4{,}03\times$, superando consistentemente a la vectorización automática gracias a una mejor gestión del *aliasing* y a la eliminación de predicciones de salto en la clasificación simbólica.
3. La fase de búsqueda de patrones constituye el cuello de botella estructural del algoritmo: no es vectorizable con la estructura de datos actual y limita el speedup global, con una fracción vectorizable efectiva estimada en $F \approx 0{,}35$ para $m = 3$.
4. Para uso práctico, `-O3 -march=native` representa la mejor relación coste-beneficio, obteniendo ≈75 % del beneficio del SIMD manual sin ningún cambio en el código fuente.

Como trabajo futuro se explorará la sustitución de la lista dinámica de patrones por una tabla *hash* con acceso $O(1)$, lo que aumentaría significativamente F y podría aproximar el speedup al teórico de $8\times$ con AVX-512. También se evaluará la extensión a paralelismo multihilo con OpenMP, la combinación con paralelismo distribuido a través de MPI [18] para señales de gran tamaño, y la portabilidad a arquitecturas ARM con instrucciones SVE (Scalable Vector Extension).

Referencias

[1] Steven M. Pincus, "Approximate entropy as a measure of system complexity," *Proceedings of the National Academy of Sciences*, vol. 88, no. 6, pp. 2297–2301, 1991.

[2] George Manis et al., "Review of Recent (2015–2024) Popular Entropy Definitions Applied to Physiological Signals," *Entropy*, vol. 27, no. 9, pp. 983, 2025.

[3] David Cuesta-Frau, "Slope Entropy: A New Time Series Complexity Estimator Based on Both Symbolic Patterns and Amplitude Information," *Entropy*, vol. 21, no. 12, pp. 1167, 2019.

[4] Christoph Bandt and Bernd Pompe, "Permutation entropy: A natural complexity measure for time series," *Physical Review Letters*, vol. 88, no. 17, pp. 174102, 2002.
[5] Joshua S. Richman and J. Randall Moorman, "Physiological time-series analysis using approximate entropy and sample entropy," *American Journal of Physiology – Heart and Circulatory Physiology*, vol. 278, no. 6, pp. H2039–H2049, 2000.
[6] David Cuesta-Frau and Antonio Molina-Picó, "Fever Time Series Analysis Using Slope Entropy. Application to Early Unobtrusive Differential Diagnosis," *Entropy*, vol. 22, no. 9, pp. 1034, 2020.
[7] J. Seguí Moreno and A. Molina Picó, "A Comprehensive Literature Review on Slope Entropy Algorithm: Bridging Past Insights with Future Directions," in *Intelligent Computing. SAI 2024. Lecture Notes in Networks and Systems, vol 1018*. 2024, Springer.
[8] Intel Corporation, "Intel Intrinsics Guide," `https://www.intel.com/content/www/us/en/docs/intrinsics-guide/`, 2023, Accedido: abril 2026.
[9] Nombre] [Apellido, "Aceleración del cálculo de Slope Entropy mediante instrucciones AVX-512," in *Actas de las XXXV Jornadas de Paralelismo, SARTECO 2025*, 2025, pp. [pp–pp].
[10] David Cuesta-Frau and Antonio Molina-Picó, "Slope Entropy Normalisation by Means of Analytical and Heuristic Reference Values," *Entropy*, vol. 24, no. 12, pp. 1717, 2022.
[11] Madalena Costa, Ary L. Goldberger, and C.-K. Peng, "Multiscale entropy analysis of complex physiologic time series," *Physical Review Letters*, vol. 89, no. 6, pp. 068102, 2002.
[12] GCC Project, "Auto-vectorization in GCC," `https://gcc.gnu.org/projects/tree-ssa/vectorization.html`, 2023, Accedido: abril 2026.
[13] David Callahan, Jack Dongarra, and David Levine, "Vectorizing compilers: A test suite and results," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, 1988, pp. 98–105.
[14] Matt Godbolt, "Compiler Explorer," `https://compiler-explorer.com/`, 2024, Herramienta interactiva para visualizar el código ensamblador generado por compiladores. Accedido: abril 2026.
[15] Daniel Kusswurm, *Modern Parallel Programming with C++ and Assembly Language: x86 SIMD Development using AVX, AVX2, and AVX-512*, Apress, New York, NY, 2022.
[16] Agner Fog, "Optimizing Software in C++: An Optimization Guide for Windows, Linux and Mac Platforms," 2023, Technical University of Denmark. Accedido: abril 2026.
[17] Gene M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the AFIPS Spring Joint Computer Conference*, 1967, vol. 30, pp. 483–485.
[18] Dong Zhong, Qinglei Cao, George Bosilca, and Jack Dongarra, "Using Advanced Vector Extensions AVX-512 for MPI Reductions," in *Proceedings of the 27th European MPI Users' Group Meeting (EuroMPI)*, 2020.

Breaking Barriers in Neural Networks and Transformers Compilation

Shreya Alladi[1], Alberto Ros[1] y Alexandra Jimborean[1]

Resumen— **This paper addresses the challenges of compiling neural networks and Transformers for resource-constrained edge devices, particularly within the RISC-V ecosystem, where performance often depends on manual optimizations. To improve scalability and portability, we introduce ONNX-MLIR-LLVM (OML), a novel compilation flow that leverages MLIR and LLVM IR to enable automatic optimizations and generate stand-alone RISC-V binaries.**

Through comprehensive analysis, we identify key limitations in existing auto-vectorization, especially for reduction operations and inefficient data layouts, and resolve them using an MLIR-based reduction pass and a compile-time transpose pass. These transformations (OML-vect) enhance the MLIR affine super-vectorizer, reducing the need for manual tuning.

Evaluations on RISC-V CPU across multiple Transformer models show that OML-vect achieves on average performance gains of 59 %, compared to manually optimized baselines, providing a more efficient and portable solution for edge deployment.

I. Introduction

AR tificial Intelligence (AI) has grown rapidly due to major advances in algorithms, the availability of training data, and software frameworks. In particular, Transformer architectures [42] have reshaped the field. Through attention mechanisms, Transformers underpin state-of-the-art models such as BERT, ViT, and GPT [38]. This progress has also increased the demand for specialized hardware accelerators, compiler optimizations, and AI frameworks that can meet the rising computational cost of AI workloads.

For example, PyTorch [1] introduces fused-kernel optimizations for layers such as self-attention and multi-head attention, achieving up to 2× inference speedup over standard implementations [34]. Combined with its large ecosystem and widespread adoption, these advances have made PyTorch a leading framework for deep learning research and deployment [13]. PyTorch relies on hand-optimized C++ kernels leveraging SIMD vectorization, tiling, unroll-and-jam, and multithreading for performance. However, Python-level dispatch can still introduce overhead from frequent library calls [41]. Moreover, PyTorch is mainly optimized for Intel, ARM, and AMD platforms, with limited support for emerging ISAs such as RISC-V, creating challenges for adoption in open hardware ecosystems.

Although efforts exist to port PyTorch to RISC-V ISA, significant challenges remain [7], including missing low-level dependencies such as cpuinfo [9] and the need to manually reimplement optimized kernels, leading to a notable performance gap between x86 and RISC-V [24]. Bridging this gap requires a static compilation flow with automated analysis and optimization for efficient Transformer inference.

One such effort is the Intermediate Representation Execution Environment (IREE) compiler [16]. IREE consists of IREE Turbine (formerly Shark Turbine) and the IREE runtime. However, the current runtime does not support generating executables for 64-bit RISC-V targets [15]. Additionally, IREE does not directly expose LLVM IR, limiting independent binary generation via the upstream LLVM toolchain outside the full runtime stack. Instead, it exposes higher-level dialects such as Torch and Linalg. Its generated MLIR is also not fully compatible with upstream MLIR due to reliance on a custom *Utils* dialect [14], making direct lowering to LLVM IR and standalone executable generation non-trivial.

Runtime deployment on edge platforms is also costly, with substantial memory overhead from dependencies such as Python. TinyIREE [17] was introduced as a lightweight alternative for embedded systems, small-footprint edge devices, and bare-metal environments, but it still cannot generate 64-bit RISC-V code.

ONNX-MLIR, by contrast, exposes the LLVM dialect and integrates more naturally with MLIR and LLVM, but it also has limitations. By default, it does not support a RISC-V backend and therefore cannot generate executables for RISC-V CPUs [31].

Additionally, ONNX-MLIR does not use MLIR's Affine super-vectorizer; instead, it replaces key kernels such as matrix multiplication with manually vectorized implementations. Although this can improve performance, it reduces flexibility and scalability. A key reason is a limitation in MLIR's reduction pass [27], which fails to detect reduction patterns in matrix multiplication when complex operations such as 'math.fma' (from the *Math* dialect) are present. Current MLIR infrastructure lacks robust support for decomposing such complex operations and recovering reduction structure. This is critical because the Affine super-vectorizer depends on identified reductions to safely and effectively vectorize along reduction dimensions [28]. Furthermore, matrix multiplication often involves non-contiguous memory accesses, which MLIR's vectorization pass typically does not auto-vectorize, further limiting automated vectorization opportunities.

To address these gaps, we propose ONNX-MLIR-LLVM (OML) and OML-vect, a RISC-V-oriented compilation flow built on ONNX-MLIR. We introduce practical enhancements that substantially improve MLIR auto-vectorization, enabling *automatic*

[1]Dpto. de Inform´atica, Universidad de Murcia

optimization of neural-network workloads at MLIR.

Overall, this work addresses key state-of-the-art limitations in compiling Transformer models for RISC-V execution. Through automatic code transformations, it also promotes automatic vectorization and reduces reliance on manually optimized kernels.

In this paper, we make the following contributions:

- We propose a compilation flow ONNX-MLIR-LLVM (OML) that (1) exposes MLIR and LLVM IR to enable automatic optimizations and (2) generates stand-alone RISC-V binaries.
- We analyze the reasons why the auto-vectorizer cannot handle (apparently) vectorization-friendly operators, such as matmul, and identify two of its main challenges: (1)undected reduction operations and (2) the data layout is not suitable for vectorization (Section III).
- Identifying reduction operations: We exhibit the limitations of the mainstream MLIR pass to identify reduction operations and how this prevents auto-vectorization. Next, we propose a more versatile, stand-alone MLIR reduction pass to identify multiple reductions (Section IV-C).
- Data layout transformations: We analyze the data layouts that prevent the auto-vectorization due to the performance penalties of the vectorized code. In response, we write a stand-alone pass to transpose the vectorization-unfriendly matrices at compile time, thus exposing contiguous memory locations (Section IV-A).
- Finally, we demonstrate how our automatic transformations (OML-vect) effectively enables the vectorization through the mainstream MLIR super-vectorizer (Section VI-B).
- We demonstrate the benefits of our approach on RISC-V ISA on a set of eight neural networks and Transformer models and show that automatic vectorization enabled by our passes outperforms the manually vectorized libraries. We achieve on-par performance compared to ONNX-MLIR manually vectorized libraries (92 % compared to ONNX-MLIR without manual optimizations). On RISC-V we surpass the performance of the ONNX Runtime for 4 out of 8 models, emphasizing the need for automatic optimizations (Section VI-B).

II. Background

This section provides a description of auto-vectorization in MLIR as well as an overview of ONNX and ONNX-MLIR.

A. Auto-Vectorization in MLIR

In MLIR, auto-vectorization is facilitated through passes such as the affine super-vectorizer, accompanied by helper passes such as *the reduction pass*. These passes analyze loop nests, identify opportunities for vectorization, and transform them into efficient vector operations, maximizing hardware performance and reducing execution time.

Input: Matrix A of size $m \times k$, Matrix B of size $k \times n$
Output: Matrix C of size $m \times n$ where $C = AB$
for $i = 0$ to m:
  for $j = 0$ to n:
    $C[i][j] = 0$
    for $l = 0$ to k:
      $C[i][j] += A[i][l] \cdot B[l][j]$

Fig. 1: Standard Matrix Multiplication (`MatMul`)

Input: Matrix A of size $m \times k$, Matrix B of size $k \times n$
Output: Matrix C of size $m \times n$ where $C = AB$
VL = Vector Length
for $i = 0$ to m:
  for $j = 0$ to n with step size VL:
    $C[i][j : j + \text{VL} - 1] = 0$
    for $l = 0$ to k:
      $A_brd = vector_\text{broadcast}(A[i][l], \text{VL})$
      $C[i][j : j + \text{VL} - 1] \mathrel{+}= A_brd \circ B[l][j : j + \text{VL} - 1]$

Fig. 2: Vectorized ONNX-MLIR `MatMul`

A.1 Reduction Identification

Reductions are essential operations used to combine values across loop iterations, such as summing elements or finding the maximum value in an array.

To this end, MLIR's reduction pass detects generic reductions within loop nests by examining a list of iteration-carried arguments (*iterCarriedArgs*) and the position of the potential reduction argument.

Identified reduction operations are used to generate reductionMaps, consumed by the Affine loop vectorizer to vectorize along the reduction dimension.

B. ONNX-MLIR

ONNX-MLIR is an open-source compiler that generates optimized inference code from ONNX models. Built on the MLIR infrastructure in LLVM, it uses a structured lowering pipeline: the ONNX graph is first converted to the ONNX dialect and then lowered to the Krnl dialect for loop-level optimizations.

At the Krnl stage, operations like MatMul and GEMM are optimized using manual optimized kernels (vector-dialect) with techniques such as tiling, unroll and jam, and vectorization. The code is then lowered to the Affine and LLVM dialects and compiled into a shared library.

B.1 MatMul

A basic `MatMul` implementation is shown in Figura 1. However, because matrix B is traversed column-wise, MLIR cannot effectively vectorize this access pattern. To address this issue, ONNX-MLIR adopts an alternative vectorization algorithm.

The ONNX-MLIR `MatMul` algorithm follows a different strategy, outer-product, as illustrated in Figure 2. In contrast to standard `MatMul`, where matrix B is accessed column-wise (see Figura 1), this approach accesses B row-wise. As a result, memory accesses to B become contiguous and more amenable to vectorization. The generated products are accumulated into the corresponding output row of matrix C.

Tabla I: Ability of MLIR to identify common reduction operations in Transformers

Kernel	Dialect	Reduction op.	Identified	Candidates
MatMul (float)	math	fma	No	Yes
MatMul (float)	arith	addf	Yes	Yes
MatMul (int)	arith	addi	Yes	Yes
SoftMax	arith	maxnumf	No	Yes

III. Motivation

The Transformer architecture, particularly the attention mechanism, has become a foundational component in generative AI and various other domains. The attention mechanism computes attention weights to quantify the relative importance of input elements, which are then used to modulate the contribution of each input token to the output.

The Transformer architecture, especially the attention mechanism, is a key component in generative AI. Attention computes weights that capture the importance of input tokens and use them to scale their contributions to the output. MatMul operations dominate execution time in both attention layers. In BERT, around 97 % of attention-layer execution time is spent on MatMuls, and similar models typically show 95 % MatMul dominance. Additional reduction operations, such as `maximum`, are also common, in `SoftMax` [42]. In MLIR, enabling auto-vectorization requires identifying reductions, so the loop can be vectorized along the reduction dimension [28].

A. Challenges with Reduction Passes in MLIR

The MLIR *match-reduction-test* pass [27, 29] identifies simple reduction patterns, such as *addf* in the *arith* dialect. However, it fails to detect more complex reductions from the *math* dialect, including fused multiply-add (FMA), maximum, and minimum operations. Tabla I summarizes common reduction operations in Transformers and shows which are supported by MLIR's pass. In contrast, all of these are candidates to be identified by the OML-reduction.

Our goal: Identify reduction operations across multiple dialects and build a reduction map [25].

B. Data Layout vs Auto-Vectorization Cost Model

Vectorization is extensively studied, but auto vectorization still lacks capability to match manually vectorized libraries. One of the key challenge for auto vectorization is non-contiguous memory accesses and data layout transformation. Vectorization is beneficial when the memory accesses are contiguous, while non-contiguous memory accesses can lead to performance degradation. Figura 1 illustrates a basic pseudocode for MatMul, clearly showing that matrix A has contiguous memory accesses, in contrast to matrix B. MLIR's *vectorizeLoops* [28] does not vectorize MatMul due to its data access patterns. Forcing vectorization against the indication given by the cost model leads to performance degradation as shown in Figura 3. Prior works [35, 19] have explored runtime data layout optimization, they come with runtime overheads (see Subsección VII-B

Our goal: Convert to a more vectorization-friendly data layout at compile-time, ensuring that

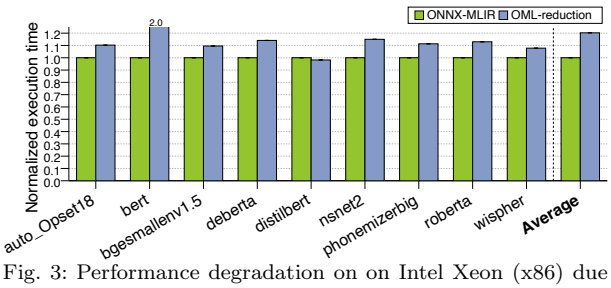


Fig. 3: Performance degradation on on Intel Xeon (x86) due to forceful vectorization when data access is non-consecutive

memory accesses target contiguous data.

C. ONNX-MLIR Vectorization

Section II-B.1 illustrates the current approach in ONNX-MLIR for vectorizing MatMul operations. The decision to employ manual vectorization libraries arises from MLIR's reduction pass inability of identifying a broad range of reduction operations, as well as the inability of the affine vectorizer to vectorize MatMul operations due to their data access patterns. ONNX-MLIR's manual optimizations ensure consecutive memory accesses for both input matrix A and B, it significantly increases the number of loads, stores (see Section II-B.1). Consequently, increasing the number of loads and stores can lead to memory bottlenecks, performance degradations due to excessive data transfers. Optimizing memory accesses is crucial to improve computational efficiency.

Our goal: Accumulate the partial results and write the final sum to the output matrix C only once to reduce memory operations as much as possible.

D. Putting It All Together

The process of improving auto-vectorization can be broken down into four key steps:

1. Apply data layout changes (e.g., by statically transposing matrices) to yield memory accesses to contiguous data.
2. Update the MatMul algorithm to accumulate the partial results and write the final sum to the output matrix only once.
3. Detect reductions.
4. Apply vectorization.

Although data layout changes can be performed either before or after detecting reductions, it is generally more efficient to carry out this step at the highest abstraction level.

IV. OML-Vect Compilation Flow

We propose two multi-stage compiler pipelines named ONNX-MLIR-LLVM (OML) and ONNX-MLIR-LLVM-vect (OML-vect), shown in Figura 4 with blue and purple boxes, respectively. Unmodified sections of the pipeline are shown in gray. Furthermore, OML-vect leverages MLIR to implement compiler passes highlighted in pink, such as a reduction pass and a data-layout transformation pass, to aid auto-vectorization and improve performance.

The pipeline starts by exporting a pre-trained model to ONNX, followed by graph traversal and data layout transformations by our proposed compile time

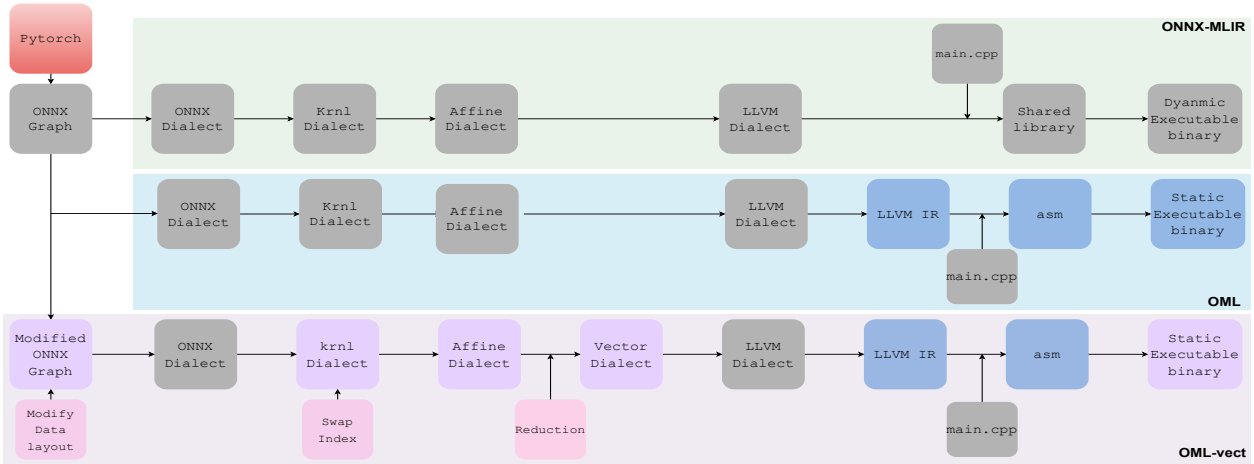
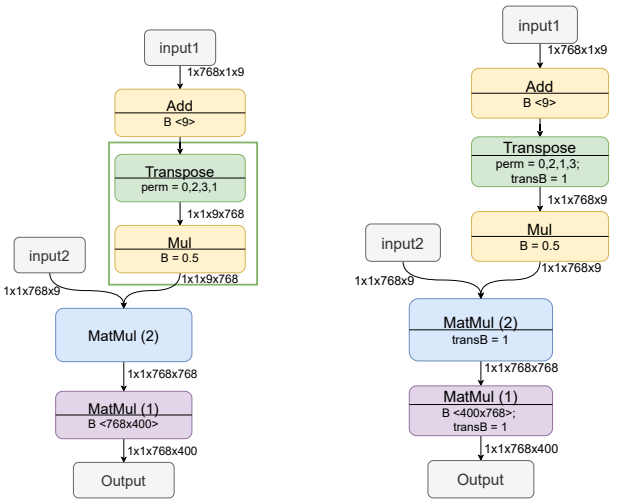


Fig. 4: Overview of ONNX-MLIR (prior work), OML, and OML-vect (our proposals) compilation flow



(a) Before (b) After

Fig. 5: ONNX graph before and after data layout optimization

data-layout transformation pass to enable efficient auto-vectorization (section IV-A). The model is then lowered to the ONNX, krnl, and affine dialects using ONNX-MLIR (see section IV-B). At the affine level, our proposed reduction pass is applied following the affine loop vectorization pass (section IV-C). Finally, the code is lowered to LLVM IR, producing assembly and standalone binaries (section IV-D).

A. Data Layout Transformations

We traverse the ONNX graph bottom-up to identify operations that cannot be auto-vectorized due to vectorization-unfriendly data access patterns. One such example is MatMul (see Section III-B). At complie-time, to convert the data layout into one suitable for vectorization, we transpose the second matrix in the multiplication.

Matmuls in Transformers can be classified into two types based on the second matrix's role, such as whether it is available at compile time or not: Weighted MatMuls and Generic MatMuls.

A.1 Weighted MatMuls

For MatMuls with embedded weights available at compile time (see Figura 5a MatMul (1)), we transpose the second input, the weight matrix at compile time and annotate the node with metadata (see Figura 5b MatMul (1)). Since this matrix is used only within this kernel, the original matrix is not retained reducing the memory footprint.

Input: Matrix A of size $m \times k$, Matrix B of size $k \times n$ and $B_Transposed = Tranpose(B)$
Output: Matrix C of size $m \times n$ where $C = AB$

for $i = 0$ to m:
for $j = 0$ to n:
$C[i][j] = 0$
for $l = 0$ to k:
$C[i][j] += A[i][l] \cdot B_Transposed[j][l]$

Fig. 6: OML-vect Matrix Multiplication (`MatMul`)

A.2 Generic Matmuls

For non-weighted MatMuls, where both inputs are only known at runtime (Figura 5a MatMul (2)), a different optimization strategy is needed. In Transformer attention layers, these MatMuls often receive the second input through a transpose operation [42], which can be exploited for optimization. We traverse the ONNX graph to locate the second input of the MatMul and stop at the first non-scalar operation, typically a transpose (see Figura 5a, green box). Instead of inserting new operations, we modify the permutation of the existing transpose and annotate the MatMul node (see Figura 5b MatMul (2)) to reflect the transposed input.

Unlike runtime transpose operations (e.g., memref.transpose), which add additional runtime instructions, overhead and extra memory usage [19, 36, 22], our approach performs this transformation at compile time eliminating the runtime overhead, data copying and paving the road to auto-vectorization.

B. ONNX to Affine Dialect

Once the data layout has been changed by compile-time transposing the input matrix, the modified ONNX graph is passed to ONNX-MLIR. Initially, the ONNX graph is lowered to the ONNX dialect and then to the Krnl dialect following ONNX-MLIR flow. The transpose-metadata flag dictates how the matrix multiplication is performed. If set, OML-vect swaps the indices of the B matrix during MatMul as shown in Figura 6; if the flag is not set, the standard

MatMul algorithm is employed as shown in Figura 1. Finally, the Krnl dialect is lowered to the affine dialect. Affine dialect servers as a structured IR where transformations such as vectorization are applied to further optimize the generated code.

C. Reduction Pass

The affine dialect represents an abstraction level higher than LLVM IR, making it well-suited for vectorization. It provides auto-vectorization capabilities through the affine super-vectorizer, which identifies innermost loops, detects reductions, builds reduction maps, and applies vectorization. However, complex operations using *iter_ args* that are not directly mapped to AtomicRMWKind [26] cannot be handled by default. To address this, we introduce the OML-reduction pass, which decomposes such operations into supported primitives. For example, math.fma can be split into mulf and addf, where addf acts as the reduction combiner. We construct a reduction map that captures the reduction pattern. This enables the affine super-vectorizer to optimize previously undetected reductions, with the resulting vectorized code lowered to the LLVM dialect.

We construct a *reductionMap* that encapsulates the reduction behavior. Finally, the affine super vectorizer is invoked to vectorized the code and then lowered to the LLVM dialect.

D. Generating Executables for RISC-V

The LLVM dialect represents LLVM IR within MLIR, but ONNX-MLIR does not directly expose LLVM IR (see Figure 4), limiting executable generation via the LLVM toolchain [20]. OML and OML-vect address this by translating the LLVM dialect into LLVM IR, enabling standalone binary generation. The generated LLVM IR serves as input to the LLVM compiler, allowing further optimizations and assembly generation through the LLVM's OPT and LLC, followed by native assembling and linking to produce executables. Exposing LLVM IR is especially critical for RISC-V, as it enables the creation of statically linked binaries using the LLVM toolchain. Our OML pipeline explicitly exposes LLVM IR and leverages LLVM to generate RISC-V binaries[1].

E. Discussion

Although Transformer models support dynamic input shapes, their weights remain static and known at compile time [21]. For weighted MatMuls, we optimize only the weight layout, as they are compile-time constants (see Section IV-A.1). For generic MatMuls in self-attention [39] and cross-attention [10], where a transpose precedes the second input, we optimize by adjusting the transpose attributes (see Section IV-A.2). This eliminates the need for runtime transposes, though runtime-aware handling may be required for the first input with dynamic shapes. Our work focuses on compile-time data layout optimizations, leaving such runtime considerations out of scope.

[1]We compiled ONNX-MLIR runtime libraries for RISC-V, manually porting components (OInstrument, OMRandomNormal, OMTensor) and resolving known issues [31]

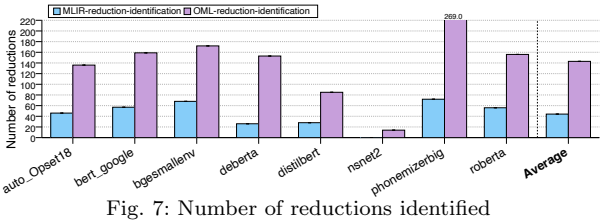


Fig. 7: Number of reductions identified

V. Methodology

We evaluate the effectiveness of OML and OML-vect using seven Transformer models and one recurrent neural network, nsNet2 [6, 23]. The Transformer models include auto_Opset18, bert_google [4], bgesmallenv [5], deberta [11], distilbert [12], phonemizerbig [33], and roberta [37]. Training and testing datasets are described in their respective references.

The models are evaluated on RISC-V platform: a Xilinx U55C FPGA emulating[2] an Atrevido 423 RISC-V 64-bit core with a 512-bit vector unit [2]. Since we use FPGA emulation, running full Transformers is time-consuming. Thus, we decided to evaluate only the attention layers.

The evaluation consists of two stages:

1. **Reduction Pass Evaluation:** Comparison between reductions identified by OML reduction and MLIR's *test-match-reduction-pass*.
2. **Performance Analysis (RISC-V):** Evaluation of attention layers under five configurations: *(a)* ONNX-MLIR-no-custom-opts (O3, no manual optimizations); *(b)* OML (default MLIR reduction pass with MLIR opt O3 and LLVM opt O3, along with auto-vectorization at O3, including SLP, loop vectorization, and VPlan [3]); *(c)* ONNX-MLIR-default (our baseline, with manual optimizations enabled (see section II-B); *(d)* OML-vect (our reduction and data layout optimizations with MLIR auto-vectorization); *(e)* ORT (ONNX Runtime [30] with hand-optimized libraries);

VI. Evaluation

This section evaluates the impact of OML and OML-vect pipeline on RISC-V CPUs.

A. Impact of the Reduction Pass

We first analyze the number of reductions identified by MLIR's `match reduction test` pass and our proposed pass OML-reduction. Figura 7 shows the total reductions detected across workloads. Our approach achieves significant improvements due to:

- It detects reductions across multiple dialects (e.g., *arith*, *math*), while `match reduction test` is limited to *arith*.
- Within *arith*, it captures additional reductions not handled by MLIR (e.g., *arith.maxnumf*),

[2]The FPGA emulates a production core(Atrevido), being 100 % cycle accurate [40]

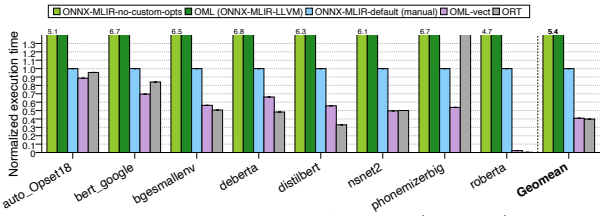


Fig. 8: Performance on Atrevido (RISC-V)

commonly arising from SoftMax in self-attention layers [42].

- It identifies complex reduction patterns in *math.fma*, which MLIR misses, improving matrix multiplication vectorization.

In summary, OML-reduction introduces a systematic decomposition strategy to detect and vectorize complex ternary reductions (e.g., FMA), addressing a gap in MLIR. This enables up to 2.5× higher reduction coverage in neural networks

B. Performance Analysis

We evaluate several approaches (see Section V) against ONNX-MLIR-default (manual) as the baseline. Figura 8 presents results on RISC-V, focusing on Transformer models where MatMul is the primary bottleneck. Out-of-the-box MLIR auto-vectorization fails to vectorize most MatMuls due to missing reduction detection, resulting in negligible gains for ONNX-MLIR-no-custom-opts (light green bar in Figura 8). While OML-reduction identifies more reductions than the default flow, it does not improve performance when used alone, since MLIR and LLVM cost models penalize non-contiguous layouts and therefore inhibit vectorization. As a result, OML also shows no significant gains (light blue bar).

ORT includes a C/C++ library with manually vectorized code optimized for RISC-V as shown in Figura 8, grey bar). There is less RISC-V-specific tuning in ORT, which allows OML-vect to outperform it on RISC-V in 4 out of 8 models (see Figura 8). ORT relies on manually designed, model-specific optimizations, particularly for popular Transformer architectures such as BERT [32]. However, for less commonly optimized models like phonemizerbig, the lack of targeted tuning leads to a noticeable performance gap between ORT and OML-vect.

On average, OML-vect (combining OML-reduction and data layout optimizations) achieves 59 % performance improvements over the baseline RISC-V (see Figura 8). These results highlight the importance of combining reduction detection with data layout optimization to fully exploit MLIR and LLVM auto-vectorization.

In contrast, ONNX-MLIR-manual uses a more efficient hand-optimized MatMul implementation (see Section II-B.1), including unroll-and-jam, and tiling. OML-vect instead relies on the standard MatMul formulation (Figura 1) and does not apply tiling. We observe that tiling with packing is only beneficial for very large matrices (e.g., $> 3000\times1000$), while for smaller matrices it introduces overhead on RISC-V platform. For instance, in attention layers of models such as BERT and RoBERTa, where matrices are typically 768×768, tiling introduces overhead.

Transposing the matrix enables contiguous memory access, making auto-vectorization feasible and beneficial. ONNX-MLIR instead uses outer-product to improve locality for matrix A and ensure contiguous access for matrix B, but this introduces partial results and increases loads and stores to the output matrix. RISC-V runtime traces show that OML-vect reduces vector loads by 18.83 % and stores by 99 % compared to ONNX-MLIR-manual, improving memory efficiency through compile-time data layout transformations. Without manual optimizations, OML-vect achieves performance comparable to ONNX-MLIR-default (manual) for most attention layers on RISC-V. Since attention dominates execution time (90 %–97 %), these trends also hold for full Transformer models on RISC-V platforms.

VII. Related Work

A. Porting PyTorch to RISC-V

Prior work [8] tried to port PyTorch to RISC V. However, several PyTorch dependencies, such as the Chromium Breakpad library, the SLEEF vectorized Math Library, and the cpuinfo, were not compatible with the RISC-V ISA. The authors extended support for these libraries for RISC-V, but this required substantial effort and still resulted in a significant performance gap compared to x86.

B. Run-Time Data Layout Optimization

Data layout optimization is essential for efficient vectorization. Prior work [18, 35] explores data replication-based layouts, which can negatively affect cache behavior, increase memory consumption, and introduce runtime overhead. Replication enlarges the working set, leading to more cache evictions and reduced locality, while also increasing memory footprint and bandwidth usage, which limits scalability. Such approaches are beneficial only when their performance gains outweigh these costs.

In contrast, our work performs data layout optimization at compile time, producing vectorization-friendly layouts eliminating runtime overhead and ensures contiguous memory accesses.

VIII. Conclusion

Our auto-vectorization approach, OML-vect, achieves performance improvements of 59 % on RISC-V over the ONNX-MLIR-manual (our baseline), which relies on manually optimizations. Although ORT outperforms our method in specific cases (e.g., roberta), OML-vect is fully automatic, making it more adaptable across diverse workloads and architectures. The OML-reduction pass works in conjunction with compile-time data layout transformations to maximize performance. By enabling vectorization-friendly layouts at compile time, our approach eliminates runtime transpose overhead and removes dependence on externally optimized or manually ported libraries.

Acknowledgment

This work has been partially funded by the EU's Horizon 2021 research and innovation program (CONVOLVE, g.a. no. 101070374 under HORIZON-CL4-2021-DIGITAL-EMERGING-01), the CDTI within the framework of the "Science and Innovation Missions" program linked to the Strategic Project for Microelectronics and Semiconductors, PERTE CHIP, (PRO-VISION-PRO, g.a. MIG-20231020), the MCIN/AEI/10.13039/501100011033/, the "ERDF A way of making Europe", EU (DAMAS, grant PID2022-136315OB-I00), and the Ramón y Cajal Research Contract (RYC2018-025200-I). The authors thank Semidynamics for their support, useful comments, and discussions along the way.

Referencias

[1] Jason Ansel et al. "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation". En: ASPLOS '24. La Jolla, CA, USA: Association for Computing Machinery, 2025, págs. 929-947. ISBN: 9798400703850. DOI: 10.1145/3620665.3640366. URL: https://doi.org/10.1145/3620665.3640366.

[2] *Atrevido 423 with V8 vector unit (online)*. 2025. URL: https://riscv.org/news/2023/07/semidynamics-announces-fully-customisable-4-way-atrevido-423-risc-v-core-for-big-data-applications/.

[3] *Auto-Vectorization in LLVM (online)*. 2010. URL: https://llvm.org/docs/Vectorizers.html#the-loop-vectorizer.

[4] *BertForSequenceClassification (online)*. 2016. URL: https://huggingface.co/docs/transformers/v4.41.3/en/model_doc/bert#transformers.BertForSequenceClassification.

[5] *bge-small-en Model by BAAI*. 2025. URL: https://huggingface.co/BAAI/bge-small-en.

[6] Sebastian Braun e Ivan Tashev. "Data Augmentation and Loss Normalization for Deep Noise Suppression". En: *Speech and Computer*. Ed. por Alexey Karpov y Rodmonga Potapova. Cham: Springer International Publishing, 2020, págs. 79-86. ISBN: 978-3-030-60276-5.

[7] Iacopo Colonnelli, Robert Birke y Marco Aldinucci. "Experimenting with PyTorch on RISC-V". En: RISC-V Summit Europe 2023. Barcelona, Spain, 2023. URL: https://hdl.handle.net/2318/1925291.

[8] Iacopo Colonnelli, Robert Birke y Marco Aldinucci. "Experimenting with PyTorch on RISC-V". En: *RISC-V Summit Europe*. University of Torino, Computer Science Dept. Corso Svizzera 185, 10149, Torino, Italy, jun. de 2023.

[9] *CPU Info PyTorch Dependency Library (online)*. 2017. URL: https://github.com/pytorch/cpuinfo.

[10] *Cross-Attention Mechanism in Transformers*. en-US. Section: NLP. URL: https://www.geeksforgeeks.org/nlp/cross-attention-mechanism-in-transformers/ (visitado 27-11-2025).

[11] *Hugging Face Transformers: DeBERTa Model Documentation*. 2025. URL: https://huggingface.co/docs/transformers/model_doc/deberta.

[12] *Hugging Face Transformers: DistilBERT Model Documentation*. 2025. URL: https://huggingface.co/docs/transformers/en/model_doc/distilbert.

[13] *PyTorch: An Ever-growing AI Framework Among AI Practitioners (online)*. 2023. URL: https://www.huawei.com/sg/news/sg/2023/pytorch-an-ever-growing-ai-framework-among-ai-practitioners.

[14] *IREE internal dialects*. 2023. URL: https://iree.dev/reference/mlir-dialects/#iree-internal-dialects.

[15] *IREE-RISCV cross compile completely infeasible (online)*. 2025. URL: https://github.com/iree-org/iree/issues/19057.

[16] *IREE: Intermediate Representation Execution Environment*. 2019. URL: https://github.com/openxla/iree.

[17] Hsin-I Cindy Liu et al. "TinyIREE: An ML Execution Environment for Embedded Systems From Compilation to Deployment". En: *IEEE Micro* 42.5 (2022), págs. 9-16. DOI: 10.1109/MM.2022.3178068.

[18] Jun Liu et al. "A compiler framework for extracting superword level parallelism". En: *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '12. Beijing, China: Association for Computing Machinery, 2012, págs. 347-358. ISBN: 9781450312059. DOI: 10.1145/2254064.2254106. URL: https://doi.org/10.1145/2254064.2254106.

[19] Jun Liu et al. "A compiler framework for extracting superword level parallelism". En: *SIGPLAN Not.* 47.6 (jun. de 2012), págs. 347-358. ISSN: 0362-1340. DOI: 10.1145/2345156.2254106. URL: https://doi.org/10.1145/2345156.2254106.

[20] *Compiling LLVM IR to Binary (online)*. 2019. URL: https://borretti.me/article/compiling-llvm-ir-binary.

[21] *Mastering LLM Techniques: Inference Optimization*. en-US. Nov. de 2023. URL: https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/ (visitado 27-11-2025).

[22] *memref.transpose (online)*. 2010. URL: https://mlir.llvm.org/docs/Dialects/MemRef/#memreftranspose-memreftransposeop.

[23] Riccardo Miccini et al. "Dynamic nsNET2: Efficient Deep Noise Suppression with Early Exiting". en. En: *2023 IEEE 33rd Internatio-*

nal Workshop on Machine Learning for Signal Processing (MLSP). Rome, Italy: IEEE, sep. de 2023, págs. 1-6. ISBN: 9798350324112. DOI: 10.1109/MLSP55844.2023.10285925. URL: https://ieeexplore.ieee.org/document/10285925/ (visitado 24-01-2025).

[24] Gianluca Mittone et al. "Experimenting with Emerging RISC-V Systems for Decentralised Machine Learning". En: *Proceedings of the 20th ACM International Conference on Computing Frontiers*. CF '23. ACM, mayo de 2023. DOI: 10.1145/3587135.3592211. URL: http://dx.doi.org/10.1145/3587135.3592211.

[25] MLIR. *mlir::affine Namespace Reference*. LLVM Project. n.d. URL: https://mlir.llvm.org/doxygen/namespacemlir_1_1affine.html#ac58a9af2e4bb4c811ad5329293609224.

[26] *AtomicRMWKind*. 2023. URL: https://mlir.llvm.org/docs/Dialects/ArithOps/#atomicrmwkind.

[27] *MLIR Test: test-match-reduction.mlir*. 2025. URL: https://github.com/llvm/llvm-project/blob/main/mlir/test/Analysis/test-match-reduction.mlir.

[28] *super-vectorization in affine dialect*. 2022. URL: https://mlir.llvm.org/doxygen/SuperVectorize_8cpp.html#a7b753fbff5d2aa2f6353b3ab53c4dcc6.

[29] *Create a generic reduction detection utility (online)*. URL: https://reviews.llvm.org/D110303.

[30] *ONNX Runtime (online)*. 2018. URL: https://github.com/microsoft/onnxruntime.

[31] *ONNX-MLIR RISC-V Support Issue (online)*. 2025. URL: https://github.com/onnx/onnx-mlir/issues/2824.

[32] *ONNX Runtime optimizations: BERT contrib ops (online)*. 2021. URL: https://github.com/ucb-bar/onnxruntime-riscv/tree/2021-12-23/onnxruntime/contrib_ops/cpu/bert.

[33] *phonemizer-big Model by RUPhon*. 2025. URL: https://huggingface.co/RUPhon/phonemizer-big/tree/main.

[34] *A BetterTransformer for Fast Transformer Inference (online)*. 2022. URL: https://pytorch.org/blog/a-better-transformer-for-fast-transformer-encoder-inference/.

[35] Gang Ren, Peng Wu y David Padua. "Optimizing data permutations for SIMD devices". En: *Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '06. Ottawa, Ontario, Canada: Association for Computing Machinery, 2006, págs. 118-131. ISBN: 1595933204. DOI: 10.1145/1133981.1133996. URL: https://doi.org/10.1145/1133981.1133996.

[36] Gang Ren, Peng Wu y David Padua. "Optimizing data permutations for SIMD devices". En: *SIGPLAN Not.* 41.6 (jun. de 2006), págs. 118-131. ISSN: 0362-1340. DOI: 10.1145/1133255.1133996. URL: https://doi.org/10.1145/1133255.1133996.

[37] *RoBERTa FacebookAI roberta-base (online)*. 2016. URL: https://huggingface.co/docs/transformers/en/model_doc/roberta.

[38] Bo-Kai Ruan, Hong-Han Shuai y Wen-Huang Cheng. *Vision Transformers: State of the Art and Research Challenges*. 2022. arXiv: 2207.03041 [cs.CV]. URL: https://arxiv.org/abs/2207.03041.

[39] *Self - Attention in NLP*. en-US. Section: NLP. URL: https://www.geeksforgeeks.org/nlp/self-attention-in-nlp/ (visitado 27-11-2025).

[40] *Semidynamics-Evaluate$_S$emidynamics$_R$ISC−V$_T$echnology.(online)*. URL: https://semidynamics.com/en/resources/evaluation.

[41] Jialiang Tan et al. "Toward efficient interactions between Python and native libraries". En: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2021. Athens, Greece: Association for Computing Machinery, 2021, págs. 1117-1128. ISBN: 9781450385626. DOI: 10.1145/3468264.3468541. URL: https://doi.org/10.1145/3468264.3468541.

[42] Ashish Vaswani et al. "Attention is all you need". En: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, págs. 6000-6010. ISBN: 9781510860964.

Arquitecturas heterogéneas y modelos de programación (3/4)

ChipletGPU-Sim: Desarrollando un Simulador para Arquitecturas GPU Multi-Chiplet

Germán Vicente[1], Antonio García-Guirado[2], Alberto Ros[1], José L. Abellán[1] y Manuel E. Acacio[1]

Resumen— **En la actualidad, las GPUs se han consolidado como el acelerador más utilizado para el procesamiento paralelo de aplicaciones modernas. Su diseño monolítico ha permitido incrementar la capacidad de cómputo mediante el aumento del número de núcleos de procesamiento y de los recursos de memoria. Sin embargo, las limitaciones propias de los procesos de fabricación de chips impulsan la búsqueda de nuevos enfoques arquitectónicos que permitan sostener su evolución.**

Las GPU multi-chiplet (MCM-GPU) presentan una arquitectura basada en la integración de múltiples módulos GPU (GPM) bajo un mismo encapsulado capaz de permitir el incremento de recursos computacionales más allá de los límites de un único chip. La fragmentación del diseño monolítico tradicional en múltiples módulos trae consigo nuevos retos que abordar para mitigar los efectos de la división física. Para caracterizar y proponer nuevas soluciones a estos desafíos, la academia necesita de simuladores arquitecturales capaces de modelar fielmente el comportamiento de las MCM-GPU.

Para abordar esta necesidad, presentamos ChipletGPU-Sim, un simulador que extiende el *framework* de simulación de GPUs monolíticas Accel-Sim para la simulación de sistemas MCM-GPU. Este entorno de simulación, actualmente en desarrollo, busca, en última instancia, proporcionar la infraestructura necesaria para el modelado fiable de configuraciones MCM-GPU para toda la comunidad investigadora. Además de la presentación del simulador y las decisiones de diseño más relevantes tomadas, realizamos una primera evaluación de su comportamiento con la que analizar su estado actual.

Palabras clave— **GPU, chiplet, MCM-GPU, simulador, evaluación.**

I. Introducción

LAS unidades de procesamiento gráfico (GPU) se han convertido en un recurso indispensable para la computación de alto rendimiento (HPC). Estos equipos, concebidos originalmente para el tratamiento de gráficos, hoy se encuentran detrás del cómputo de una amplia selección de cargas de trabajo. Desde grandes modelos de lenguaje (LLM) hasta simulaciones médicas, químicas o meteorológicas, las GPUs han evolucionado hasta convertirse en un acelerador para cargas de trabajo altamente paralelas. Desde sus orígenes, las GPUs han aumentado su capacidad computacional, a través del incremento del número de núcleos de procesamiento (SM), sobre un diseño monolítico en el que todos los componentes están integrados en un mismo chip. Pese a que este enfoque ha permitido el avance de las GPUs hasta ahora, las limitaciones de los procesos litográficos actuales, con un tamaño máximo de retículo de 858 mm^2 [1], impiden seguir incrementando el número de SMs dentro de un único chip y fuerzan la investigación en nuevas arquitecturas capaces de seguir mejorando la capacidad de cómputo de las GPUs.

Una estrategia inicial para escalar el rendimiento sin modificaciones sustanciales en la arquitectura consiste en la interconexión de múltiples GPUs en un sistema multi-GPU (mGPU), como el sistema DGX-1 de NVIDIA [2]. Este enfoque desplaza el foco desde la optimización del rendimiento de una única GPU hacia la mejora global mediante la coordinación y cooperación de múltiples GPUs interconectadas. Sin embargo, este cambio de paradigma no es trivial y trae consigo problemas derivados de la interconexión física entre GPUs y una mayor complejidad al programador. Conforme nos alejamos de las unidades de cómputo de las GPUs, las conexiones presentan un menor ancho de banda, dando lugar a un mayor tráfico entre las múltiples GPU y un mayor gasto energético. Asimismo, los sistemas mGPU introducen un aumento en la complejidad del modelo de programación con el que debe lidiar el programador en comparación con las GPUs monolíticas, requiriendo de la gestión de un conjunto de tareas críticas para maximizar la utilización de las mGPUs, como son la división y el balanceo de la carga de trabajo, así como la compartición de datos.

Poniendo en valor las dificultades que presentan las mGPU, surgen las GPUs multi-chiplet (MCM-GPU), un diseño basado en la réplica e interconexión de múltiples módulos GPU (GPM) bajo un único empaquetado. Esta propuesta permite seguir explorando el rendimiento individual de las GPUs a la vez que ofrece al programador un sistema lógicamente unificado sobre una arquitectura físicamente distribuida. Esta configuración de múltiples chiplets está presente en las últimas generaciones GPU de NVIDIA [3], [4] y AMD [5], mostrando cómo las MCM-GPU se están integrando en la industria. En la academia, constituyen un campo de interés que presenta nuevos retos que abordar para seguir mejorando las capacidades de las GPUs.

Para que los investigadores sean capaces de identificar cuellos de botella, proponer soluciones y comparar sus propuestas con el estado del arte, es imprescindible disponer de herramientas de simulación que modelen fielmente las MCM-GPU. Si bien la comunidad encuentra a su disposición diferentes simuladores de arquitecturas GPU, carece de un simulador de

[1]Dpto. de Ingeniería y Tecnología de Computadores, Universidad de Murcia, e-mail: {germanvt,jlabellan,meacacio}@um.es,aros@ditec.um.es
[2]Imagination Technologies, e-mail: Antonio.Guirado@imgtec.com

MCM-GPU abierto y contrastado. Esta ausencia dificulta la comparación de propuestas y la replicación de resultados, sentando una primera barrera para los investigadores que quieran adentrarse en el estudio de estos sistemas.

En este trabajo presentamos ChipletGPU-Sim, un simulador destinado a la comunidad científica para impulsar el avance de las MCM-GPU. Partiendo de la última versión disponible del *framework* Accel-Sim [6], nos servimos de su modelo de simulación de arquitecturas GPU monolíticas y lo extendemos mediante la división física en chiplets e inclusión de las estructuras propias de las MCM-GPUs sin modificar el *front-end* de simulación. Durante nuestro proceso de desarrollo, fundamentamos todas las decisiones de diseño conforme a las características de los productos presentes en la industria y el estado del arte de las MCM-GPUs.

Las contribuciones de este trabajo son:

- ChipletGPU-Sim, un simulador para arquitecturas MCM-GPU, desarrollado a partir de Accel-Sim [6], cuyas decisiones de diseño se apoyan en un análisis de las características de los sistemas MCM-GPU modelados en el estado del arte.
- Análisis del rendimiento de sistemas MCM-GPU respecto a un sistema monolítico ideal y caracterización inicial de un sistema MCM-GPU con la configuración propia del estado del arte.

II. Estado del Arte

A. Arquitectura MCM-GPU

La arquitectura MCM-GPU propuesta por Arunkumar *et al.* [7] presenta una agrupación de módulos GPU (GPM) homogéneos interconectados en un mismo empaquetado. De esta forma, cada GPM dispone de una red intra-chiplet que le permite conectar sus recursos locales y se enlaza a través de la red inter-chiplet con el resto de chiplets en el sistema para el acceso a las estructuras de memoria remotas. Cada uno de los módulos dispone de múltiples SMs acompañados por sus cachés privadas de primer nivel (L1D$); el sistema de *crossbars* (xbars) denominado GPM-Xbars, encargado del correcto enrutamiento de las peticiones dentro y fuera del chiplet; y un conjunto de cachés de último nivel (LLC) cada una de ellas conectada con una partición de memoria.

En la Figura 1 se muestra la estructura de una MCM-GPU compuesta por 4 GPMs junto a las conexiones existentes en la red intra-chiplet e inter-chiplet. Dentro de cada GPM observamos un nuevo nivel de caché denominado L1.5 (L1.5$) destinado a la reducción del número de accesos a las LLC y particiones de memoria a través de la red inter-chiplet que será motivado y descrito en la Sección II-A.2.

A.1 Accesos a memoria no uniformes

La división física de las MCM-GPUs trae consigo un modelo de accesos no uniformes a memoria (NUMA) a pesar de que el sistema expone un espacio de memoria lógicamente compartido al programador.

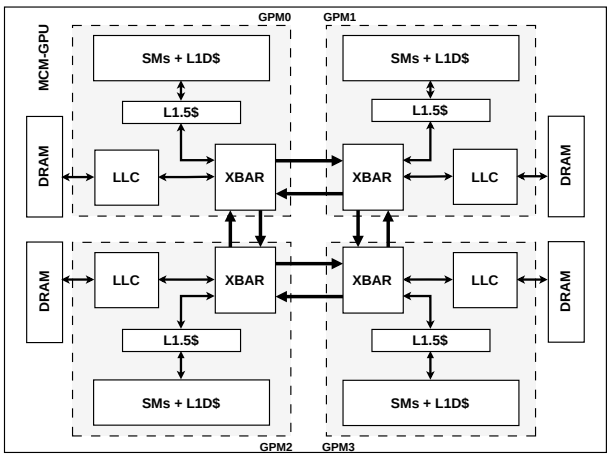


Fig. 1: Arquitectura de una MCM-GPU provista de una caché L1.5 destinada a la mitigación de los accesos remotos

Este modelo de accesos no uniformes también se extiende a la LLC (NUCA), ya que cada banco de la LLC solo mantiene datos pertenecientes a la partición de memoria a la que se encuentra directamente conectada, comportándose como una *memory-side* caché con el objetivo de simplificar el mantenimiento de la coherencia en el sistema. Esta división física de la LLC y la memoria traen consigo la diferenciación entre accesos locales y remotos. Los accesos locales son aquellos destinados a las LLC y particiones de memoria directamente accesibles por cada GPM; mientras que los accesos remotos son aquellos que deben salir a la red inter-chiplet para su resolución en otro GPM.

En un sistema como el propuesto, es crucial la correcta gestión de accesos locales y remotos, ya que la gestión ineficiente de los accesos remotos entre los GPM provoca una degradación de prestaciones. Más concretamente, esta gestión ineficiente de los accesos remotos se traduce en un aumento de la latencia de las respuestas, una alta contención en la red inter-chiplet y un mayor consumo energético. Algunas de las estrategias utilizadas para mitigar la aparición de los accesos remotos se apoyan en el estudio de la selección de políticas de mapeo de páginas y de distribución de los bloques de hilos (CTA), que permiten maximizar el uso de datos locales de cada GPM. La política de mapeo de páginas *first-touch* (FT) permite asociar la gestión de una página al primer GPM que accede a ella, maximizando el número de accesos locales para aquellas cargas de trabajo con patrones de acceso regular. Por otro lado, la asignación de los CTAs a través de una política distribuida permite la asignación de CTAs consecutivos para un mismo chiplet, minimizando así también los accesos remotos para aquellas cargas de trabajo con patrones de acceso en los que hilos subsecuentes acceden a datos consecutivos.

A.2 Aumento de la jerarquía de caché: Caché L1.5

Con el fin de aumentar la reutilización de los datos remotos obtenidos a través de la red inter-chiplet y con ello reducir el tráfico entre chiplets, Arunkumar

et al. [7] proponen la incorporación de un nuevo nivel de caché situado entre la caché L1D y la LLC: el nivel L1.5. La caché L1.5 constituye un nivel compartido de caché entre todos los SMs de un mismo chiplet en el que almacenar datos provenientes del resto de chiplets del sistema. De esta forma, la caché L1.5 recibe todas las peticiones emitidas por la caché L1D y solo atiende aquellas cuya resolución debe realizarse en otro GPM, permitiendo la omisión de la caché L1.5 para las peticiones locales al GPM.

En el estado del arte, este nuevo nivel L1.5 presenta una política de escritura *write-through* (WT) que encaja perfectamente con el mantenimiento exclusivo en caché de datos remotos, el uso de la política FT para el mapeo de páginas y la política distribuida para la asignación de los CTA. En esta configuración, los accesos remotos quedan limitados a una fracción menor frente a los accesos locales, que concentran el mayor volumen de accesos a memoria. La selección de una política WT capaz de propagar automáticamente los cambios a la LLC, permite la simplificación de la gestión de la coherencia permitiendo su mantenimiento a través de la propagación hasta las cachés L1.5 de las operaciones de *flush* cuando estas son recibidas por las cachés L1D.

B. Simuladores GPU

Los simuladores GPU constituyen una herramienta fundamental para los investigadores enfocados en la caracterización y mejora de estos sistemas. Gracias a la colaboración de la academia y la industria, hoy en día encontramos simuladores que permiten cubrir diferentes tipos de arquitecturas GPU:

- **Accel-Sim** [6]**.** Este es el *framework* de referencia para la simulación de GPUs de la familia NVIDIA. Accel-Sim ofrece un entorno de simulación para modelar y validar arquitecturas actuales y futuras de una forma sencilla y precisa. El motor de simulación GPGPU-Sim ofrece un modelo de rendimiento de gran fiabilidad que se vio sometido a una mejora significativa con el lanzamiento de su versión 4.0.
- **Modelos GPU de gem5.** El simulador gem5 cuenta con una amplia trayectoria y una gran popularidad en la arquitectura de computadores, siendo uno de los simuladores de referencia para el modelado de CPUs. Su colaboración con AMD permitió incorporar una extensión GPU [8] con la que modelar una *accelerated processing unit* (APU) de la familia GCN3, que presenta una CPU y una GPU integradas en un mismo chip. La colaboración de la comunidad ha permitido el avance de la arquitectura APU originalmente propuesta hasta el soporte de VEGA [9] y la incorporación de nuevos modelos de GPUs discretas [10] de gran capacidad computacional, como la MI300X [11].
- **MGPUSim** [12]**.** Este simulador permite modelar sistemas multi-GPU de AMD y está validado frente a sistemas reales de la familia GCN3 [13]. Una de las características principales de este simulador es el soporte multihilo que permite minimizar el tiempo de simulación necesario, uno de los grandes problemas de los simuladores hardware.

Tal y como hemos expuesto, los simuladores GPU disponibles en la actualidad permiten el modelado de GPUs monolíticas y sistemas multi-GPU. Sin embargo, la comunidad no tiene a su disposición un simulador que permita la simulación de sistemas MCM-GPU. Si bien existe un simulador de reciente lanzamiento basado en el modelo APU de gem5 [14], su sistema propuesto difiere del adoptado por los trabajos de referencia de las MCM-GPUs. Es por esto que la mayoría de los trabajos de investigación dirigidos a sistemas MCM-GPU han optado por el desarrollo de sus propias extensiones privadas [15], [16], [17], [18], [19], [20], [21], [22] para dar soporte a esta nueva organización en chiplets. Cabe destacar que muy recientemente se ha publicado a través de un repositorio abierto el *artifact* utilizado por Park *et al.* [16].

III. ChipletGPU-Sim

Ante la falta de un simulador libre que, al inicio de nuestra investigación, nos permitiera caracterizar el comportamiento de las arquitecturas MCM-GPU, decidimos comenzar el desarrollo de nuestro propio simulador. Para ello, decidimos partir de un *framework* de simulación de gran reputación como es Accel-Sim. De este modo, ideamos las modificaciones necesarias a incorporar para ser capaces de modelar una MCM-GPU a partir del modelo monolítico original. ChipletGPU-Sim constituye un simulador de arquitecturas MCM-GPU cuyo propósito es ofrecer a la comunidad científica un entorno de simulación capaz de realizar un modelado fiel de las MCM-GPU en el que evaluar propuestas del estado del arte y nuevas soluciones destinadas a la mejora de esta nueva arquitectura.

A. Extensión para el modelado de MCM-GPU

La migración de un sistema monolítico a un sistema MCM-GPU requiere de un alto grado de conocimiento tanto de las arquitecturas GPU en sí, como de su modelado por la herramienta de simulación. En lo que respecta al modelo de ejecución, donde incluimos todo aquello relacionado con los SMs, no se ha requerido de la modificación de grandes aspectos para la transición al nuevo modelo. Sin embargo, existen dos aspectos que han ocupado la mayor parte de nuestro trabajo para modelar correctamente el diseño MCM-GPU: la división de una única red global en dos nuevas redes para la gestión del tráfico intra-chiplet e inter-chiplet; y la ampliación de la jerarquía de cachés. En la Figura 2 encontramos la arquitectura del sistema MCM-GPU modelado en ChipletGPU-Sim, resaltando en color gris las diferencias en las redes intra-chiplet e inter-chiplet respecto a la arquitectura presente en la Figura 1. En nuestro modelo, hemos extraído del bloque original GPM-Xbar una conexión directa entre la caché L1.5 y las LLC locales al chiplet y dedicado la nueva estructura

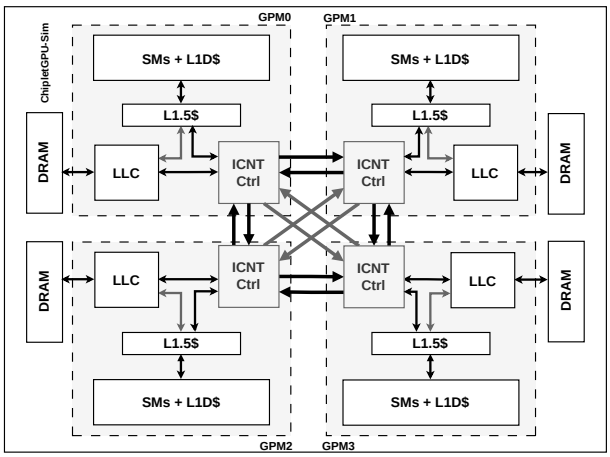


Fig. 2: Arquitectura del sistema MCM-GPU simulado en ChipletGPU-Sim

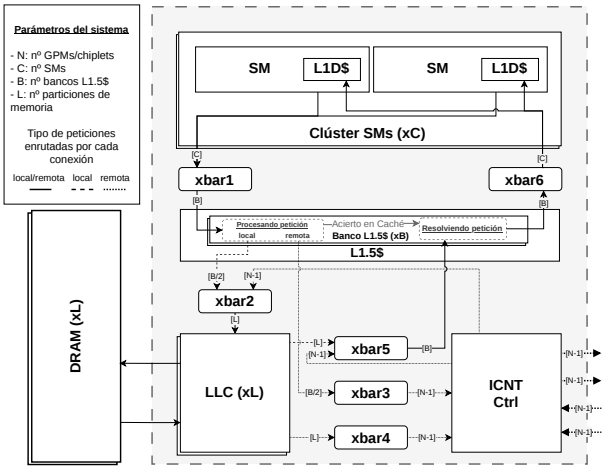


Fig. 3: Esquema de los componentes principales de cada uno de los módulos de GPU y los *crossbars* incorporados para la gestión de la conexión entre componentes, detallando los tipos de peticiones que transcurren por cada conexión

ICNT Ctrl únicamente a la gestión de la red inter-chiplet. Respecto a la red inter-chiplet, hemos decidido para este primer modelo ampliar la topología en anillo (*ring*) original a una topología totalmente conectada (*all-to-all*).

B. Diseño de las redes de interconexión

En el modelo monolítico disponible en Accel-Sim [6] encontramos dos tipos de recursos totalmente conectados entre sí a través de la red de interconexión *on-chip*: las agrupaciones o clústeres de SMs y las particiones de memoria. Esta conexión *all-to-all* entre SMs y particiones de memoria permite la simplificación de la red global *on-chip*. Sin embargo, la arquitectura MCM-GPU divide los recursos del sistema entre los diferentes chiplets, generando la necesidad de modelar dos redes independientes para el tráfico intra-chiplet y el tráfico inter-chiplet, tal y como se presenta en la Sección II-A.1.

B.1 Diseño de la red inter-chiplet

La red inter-chiplet está diseñada para transportar los accesos remotos generados por los chiplets. Esta red presenta una topología totalmente conectada constituida por múltiples canales unidireccionales entre cada pareja de chiplets en el sistema. Con el fin de ofrecer un ancho de banda acorde al utilizado en estudios previos y simplificar la gestión del tráfico dentro del simulador, por cada pareja de chiplets, cada chiplet cuenta con un canal exclusivo para el envío de peticiones y otro para el envío de respuestas con el chiplet con el que conforma la pareja. La gestión del tráfico inter-chiplet se lleva a cabo por la nueva estructura que incluimos dentro de los chiplets denominada *ICNT Ctrl*. Esta estructura se encarga de proporcionar, por cada ciclo, el envío o recepción de un mensaje por cada uno de los enlaces a los que se encuentra conectado el chiplet.

B.2 Diseño de la red intra-chiplet

La red intra-chiplet permite el enrutamiento de los mensajes que fluyen dentro de un chiplet, pudiendo haber sido estos generados por el propio chiplet o peticiones remotas de otros chiplets que han llegado para su procesamiento. En la Figura 3 presentamos, de forma detallada, los elementos contenidos en un chiplet, así como los *crossbars* (xbars) y enlaces que permiten la correcta conexión entre ellos. Tal y como se aprecia en la Figura 3, se han incorporado 6 xbars para modelar las conexiones existentes entre los elementos que componen los chiplets. Estos 6 xbars atienden diferentes necesidades del sistema, pudiendo diferenciar a gran escala entre aquellos encargados de conectar los clústeres de SMs y la caché L1.5; y los encargados de conectar entre sí la caché L1.5, las particiones de memoria y el *ICNT Ctrl*.

El xbar1 y el xbar6 son los encargados de establecer la conexión entre los clústeres de SMs y la caché L1.5. Modelamos un enlace para el envío de datos y un enlace para la recepción de respuestas por parte de cada clúster de SM a la caché L1.5. En el caso de una caché L1.5 monolítica que únicamente fuera utilizada por una instancia de caché L1D podríamos conectar directamente ambas cachés entre sí. Sin embargo, dado que existen múltiples instancias de caché L1D y la caché L1.5 está banqueada (Sección III-C), necesitamos incluir *crossbars* para la gestión del tráfico a los diferentes bancos de la caché L1.5.

Si nos movemos más allá de la caché L1.5, nos encontramos con 4 *crossbars*, dedicados a dirigir peticiones a las particiones de memoria locales (xbar2), el envío de peticiones remotas (xbar3), el envío de respuestas remotas (xbar4); y enrutar las respuestas a las peticiones propias de la caché L1.5 del GPM (xbar5). El xbar2 es el encargado de dirigir las peticiones locales generadas en el propio chiplet a sus particiones de memoria, así como también las peticiones remotas que han llegado al chiplet por medio de la red inter-chiplet. El xbar3 constituye el punto de conexión a través del que las peticiones remotas generadas en la caché L1.5 llegan al *ICNT Ctrl* para salir a la red inter-chiplet. De forma análoga al funcionamiento del xbar3, el xbar4 conecta las particiones de memoria del chiplet con el *ICNT Ctrl* para la salida de las respuestas a las peticiones remotas

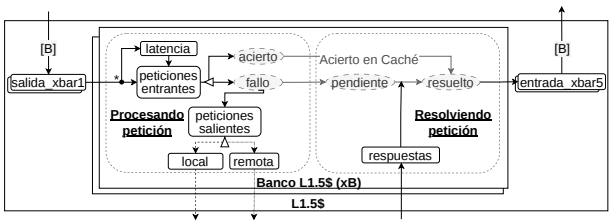


Fig. 4: Estructura interna de los bancos de la caché L1.5

que han sido procesadas en el chiplet. Finalmente, el xbar5 dirige las respuestas a peticiones locales desde las particiones de memoria locales; y las respuestas de peticiones remotas llegadas al *ICNT Ctrl* al banco correspondiente de la caché L1.5.

C. Diseño de la caché L1.5

Nuestro modelo MCM-GPU amplía la jerarquía de cachés presente en el modelo monolítico con la introducción del nivel L1.5 [7], obteniendo un total de tres niveles de caché: L1D, L1.5 y LLC. Respecto al modelo monolítico, mantenemos intacto el primer nivel de caché e introducimos el nuevo nivel intermedio previo al acceso al último nivel de caché (LLC), referido en el modelo monolítico como L2. Cada instancia de la caché L1D es privada a su SM mientras que la caché L1.5 está compartida por todos los SMs de un mismo chiplet y la LLC se encuentra lógicamente compartida por todo el sistema, pese a estar físicamente distribuida.

En nuestro sistema modelado, dado que la caché L1.5 es compartida entre todos los clústeres de SMs de un mismo chiplet, la hemos particionado en múltiples bancos independientes. La división de un recurso compartido como este evita que el procesamiento de las peticiones suponga un cuello de botella para el sistema e idealmente nos permite atender, en cada ciclo, tantas peticiones como bancos existan. La distribución de las peticiones dentro de la caché L1.5 se realiza por medio de una función *hash* basada en la operación XOR que nos proporciona los bits para la selección del banco y del conjunto. La lógica interna de los bancos de la caché L1.5 se presenta en la Figura 4, donde mostramos las colas de peticiones y respuestas implicadas en la resolución de los accesos a memoria, junto con una simplificación de los estados por los que transiciona una petición de memoria.

C.1 Integración de la caché L1.5 en ChipletGPU-Sim

Actualmente, el modelo MCM-GPU de ChipletGPU-Sim pretende establecer el *baseline* sobre el que integrar y evaluar las propuestas existentes del estado del arte. El estado actual del sistema motiva pequeñas modificaciones sobre el diseño original propuesto para la caché L1.5. En esta sección exponemos las decisiones de diseño tomadas para integrar este nuevo nivel intermedio de caché en nuestro simulador: almacenamiento de datos locales y remotos; y la política *write-back* (WB).

En nuestro modelo actual, empleamos la implementación de Accel-Sim [6] de la función hash IPOLY [23], basada en operaciones XOR polinómicas sobre polinomios irreducibles, para la asignación estática de direcciones a los distintos bancos de memoria del sistema. La ausencia de una correlación entre las direcciones a las que acceden los chiplets y las direcciones de las cuales son responsables sus particiones de memoria locales provoca que, en un sistema compuesto por N chiplets y para patrones de acceso regulares, los accesos a memoria se distribuyan entre todos los GPMs resultando en un $1/N\,\%$ de accesos locales y un $(N-1)/N\,\%$ de accesos remotos. La relación $1:(N-1)$ de accesos locales frente a remotos plantea un escenario de interés en el que evaluar si destinar parte del almacenamiento al mantenimiento de datos locales supone una ventaja frente al acceso a las particiones de memoria en todas las ocasiones sacrificando ese espacio de almacenamiento que podría destinarse a datos remotos.

Respecto a la política de escritura utilizada, en nuestro actual sistema encontramos un alto porcentaje de accesos remotos por lo que aunque la política WT permite extender el esquema de coherencia propio de las cachés L1D, inundaríamos la red inter-chiplet con la propagación de las escrituras a sus correspondientes particiones remotas. Es por esto que optamos por una política WB que nos permita retrasar la actualización de los bloques escritos hasta los puntos de sincronización por medio del uso de la operación de *flush*. Para conseguir llevar a cabo nuestro planteamiento hemos implementado las operaciones de *flush* en la finalización de *kernels* para la caché L1.5 y la LLC, ya que se trata de una funcionalidad ausente en Accel-Sim [6].

IV. Metodología

Sistema simulado. Para la configuración de nuestro sistema MCM-GPU base de 4 chiplets (4-GPM) partimos de las especificaciones del modelo V100 [24] de NVIDIA y ajustamos los nuevos parámetros definidos para la configuración del sistema MCM-GPU en base a los sistemas evaluados en trabajos previos [7], [22], [16]. En la Tabla I se muestran los parámetros seleccionados para la configuración 4-GPM, entre los que destacamos la elección de los bancos que conforman la caché L1.5 y los canales de los que se compone la red inter-chiplet. Respecto al número de bancos de la caché L1.5, se ha escogido el mismo número que clústeres de SMs presentes en cada chiplet para permitir que, en caso de que los accesos se distribuyan de forma ideal entre los bancos disponibles, se pueda atender una petición de cada clúster por ciclo. Respecto a la red inter-chiplet, la presencia de los cuatro canales dedicados entre cada pareja de chiplets (Sección III-B.1) nos permite satisfacer los 768GB/s modelados en el estado del arte gracias a un total de 24 canales unidireccionales con un tamaño máximo de petición de 40B (32B/datos).

Otras configuraciones. El objetivo de nuestra evaluación reside en la caracterización del comportamiento de nuestra configuración actual de 4 chiplets

Tabla I: Configuración 4-GPM

Parámetro	Valor
Nº Chiplets	4
Nº SMs	256 (32 clústeres/GPM, 2 SMs/clúster)
Freq. GPU (GHz)	1
Tamaño Línea de Caché	128B
Caché L1D	128KB/SM, 4 bancos, 32KB, 4 conjuntos, 256 vías, WT
Caché L1.5	2MB/GPM, 32 bancos, 64 KB, 32 conjuntos, 16 vías, WB
Caché LLC	8MB/sistema, 64 bancos, 128 KB, 64 conjuntos, 16 vías, WB
Tamaño Caché L1.5+LLC	16 MB
Red inter-chiplet	Topología todos a todos, 32 ciclos/salto, 40B/ancho, 24 canales unidirecc., 768GB/s (1GHz)
Memoria DRAM	64 GB, 16 part./GPM, HBM2, 3.48TB/s (850MHz)

Tabla II: Cargas de trabajo evaluadas

Suite	Benchmark	Input
Rodinia	b+tree (BT)	command.txt
	backprop (BP)	65536
	BFS	graph65536.txt
	NN	filelist_32
	lavaMD (LMD)	10x10x10
	hotspot (HS)	temp_512, power_512
Pannotia	pagerank (P)	coAuthorsDBLP.graph
Polybench	2D conv. (2DC)	4096×4096
	3D conv. (3DC)	256×256×256
	GEMM	512×512×512

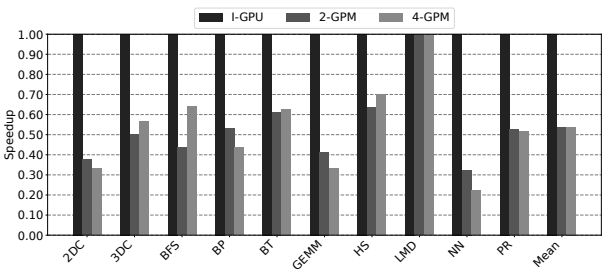


Fig. 5: Comparación del speedup de los sistemas MCM-GPU frente al modelo monolítico ideal (I-GPU)

respecto a dos sistemas con las mismas capacidades de procesamiento y de memoria pero con distinta organización física: una MCM-GPU de 2 chiplets y una configuración monolítica ideal.

- **2-GPM:** esta configuración presenta las mismas características que la configuración 4-GPM pero valiéndose únicamente de 2 chiplets. Para alcanzar la misma capacidad computacional y de memoria que 4-GPM, ampliamos hasta 64 los clústeres de SMs en cada chiplet; y la caché L1.5 queda compuesta por 64 bancos para obtener un total de 4MB por GPM. Sin embargo, con el objetivo de proporcionar el mismo ancho de banda que para la configuración 4-GPM, hemos incorporado a nuestro simulador un nuevo parámetro que junto con la modificación del fichero de configuración de la red, nos permite incrementar el número de mensajes enviados entre cada par de GPMs. De esta forma, pasamos de los 4 canales unidireccionales (128GB/s), a los que quedaría reducida la red sin la presencia de esta modificación, a 24 canales unidireccionales tal y como se presenta en 4-GPM pero únicamente entre dos GPMs a través de un factor multiplicador de 6.
- **I-GPU:** esta configuración definida como ideal debido a la incapacidad de fabricarla con las limitaciones actuales, nos permite disponer de un sistema monolítico de las mismas características que los sistemas MCM-GPU contemplados con el que sentar el rendimiento ideal que se buscaría alcanzar en los sistemas MCM-GPU. Este sistema ideal utiliza la configuración del modelo V100, usado para la configuración del resto de sistemas evaluados, aumentando el número de SMs totales hasta 256, conforme a las configuraciones MCM-GPU utilizadas; y modificando el almacenamiento destinado a las cachés. Puesto que la GPU monolítica cuenta con un nivel menos de caché y para que la memoria destinada a las cachés sea equivalente en las tres configuraciones, hemos repartido uniformemente el tamaño asociado a las cachés L1.5 entre las múltiples cachés L1D, aumentando el número de conjuntos de 4 a 5 en cada caché incrementando su tamaño hasta los 160KB.

Aplicaciones. Las cargas de trabajo seleccionadas para realizar nuestra evaluación pertenecen a tres de las suites de benchmarks más utilizadas en el estudio de las MCM-GPUs como son Rodinia [25] (en su versión 3.1), Pannotia [26] y Polybench [27]. En la Tabla II se recogen los 10 benchmarks utilizados, junto a sus *inputs*. Los *inputs* utilizados para las suites de Rodinia y Pannotia han sido extraídos directamente de la evaluación realizada por Baiqing Zhong *et al.* [22].

V. Resultados Experimentales

Para el análisis de los resultados obtenidos utilizaremos las estadísticas propias que nos ofrece Accel-Sim [6] y nuevas métricas definidas durante nuestra extensión del simulador. Dentro de estas nuevas métricas, encontramos métricas transversales a los sistemas monolíticos y MCM-GPU, así como estadísticas propias para la caracterización de los sistemas MCM-GPU.

A. Evaluación del rendimiento

Como podemos observar en la Figura 5, a través del *speedup* normalizado respecto a la configuración I-GPU, ninguno de los sistemas MCM-GPU es capaz de alcanzar el rendimiento de la configuración monolítica. Estos resultados son esperables, ya que quedan múltiples optimizaciones propuestas en el estado del arte por incluir en nuestro simulador que permitirán reducir la brecha de rendimiento respecto a la arquitectura monolítica. Si comparamos el *speedup* obtenido por las configuraciones MCM-GPU a partir de la columna *Mean*, puede parecer que ambas

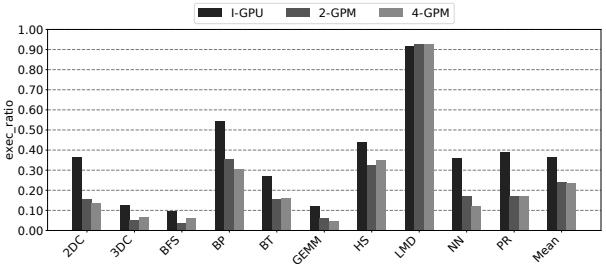


Fig. 6: Relación de los ciclos de ejecución frente a los ciclos totales requeridos (*exec_ratio*) durante la ejecución de cada benchmark sobre los tres sistemas evaluados

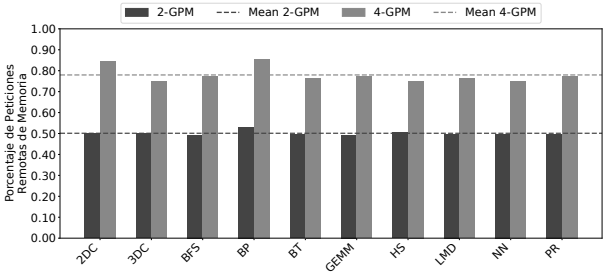


Fig. 7: Porcentaje de las peticiones remotas de memoria observadas para las configuraciones MCM-GPU para cada uno de los benchmarks estudiados

configuraciones se comportan de manera similar, sin embargo, si nos fijamos en los resultados para cada uno de los diferentes benchmarks, podemos observar rápidamente diferencias en su comportamiento. Llevaremos a cabo un análisis en profundidad del comportamiento de ambas configuraciones MCM-GPU en el apartado V-B.

Accel-Sim [6] nos ofrece una métrica denominada *gpu_occupancy* con la que el usuario puede observar la ocupación del sistema durante la simulación. La ocupación del sistema se define como: la acumulación durante todos los ciclos y para todos los SMs disponibles, del ratio de *warps* 'activos' frente a los totales que podrían ejecutarse en cada SM. Sin embargo, dado que un *warp* es considerado 'activo' en el momento en que es asignado a un SM, esta métrica no puede ser utilizada para controlar los ciclos que un SM realmente utiliza sus unidades funcionales. Por ello, hemos incorporado dos nuevas métricas: *exec_cycles* y *exec_ratio*. A través de *exec_cycles*, somos capaces de contar el número de ciclos en los que al menos una unidad funcional del SM se encuentra activa; y mediante *exec_ratio* relacionamos los ciclos proporcionados por *exec_cycles* respecto a los ciclos totales empleados por los SMs para procesar los *warps*. En la Figura 6 mostramos, para cada una de las configuraciones propuestas, el *exec_ratio* para cada uno de los benchmarks evaluados, mostrando cómo los valores aportados a través de esta métrica concuerdan con lo observado en la Figura 5. Esta gráfica nos permite observar el tiempo realmente dedicado a la ejecución y cómo las arquitecturas MCM-GPU presentan de forma general un *exec_ratio* bastante alejado del obtenido por la configuración I-GPU a consecuencia de la alta latencia derivada de la resolución de los accesos remotos.

B. Evaluación de las configuraciones MCM-GPU

Si miramos el conjunto completo de benchmarks a través de la columna *Mean* en la Figura 5 observamos que ambas configuraciones MCM-GPU presentan un comportamiento muy similar. Sin embargo, aunque ambas configuraciones puedan parecer completamente análogas ya que presentan las mismas capacidades computacionales, tamaño dedicado a las estructuras de memoria y ancho de banda en la red inter-chiplet; la división física en 2 o en 4 módulos GPU trae consigo importantes efectos para un sistema con accesos

Tabla III: Benchmarks con menor diferencia entre configuraciones para el porcentaje de peticiones remotas de memoria observadas

Benchmark	**2-GPM**	**4-GPM**	$\lvert\Delta\rvert$
HS	0.5053	0.7513	0.2460
3DC	0.4994	0.7493	0.2499
NN	0.4977	0.7503	0.2526
LMD	0.4974	0.7615	0.2641
BT	0.4961	0.7660	0.2699
PR	0.4984	0.7749	0.2765
GEMM	0.4922	0.7709	0.2786
BFS	0.4928	0.7723	0.2795
BP	0.5314	0.8557	0.3242
2DC	0.4999	0.8456	0.3458

no uniformes a memoria. La existencia de 2 GPMs implica que cada GPM disponga de 32 particiones de memoria locales, frente a las 16 contempladas para cada GPM en la configuración 4-GPM. Esta diferencia en el número de particiones locales a cada GPM provoca que, para un conjunto de accesos equitativo entre las diferentes particiones de memoria del sistema, la configuración 4-GPM presente un 75 % de accesos remotos frente a un 50 % en la configuración 2-GPM. En la Figura 7 podemos observar para cada uno de los benchmarks evaluados el número de peticiones remotas generadas para cada una de las configuraciones MCM-GPU. En media, los valores teóricos expuestos para la relación de peticiones remotas en cada sistema se ajustan a los valores reales medidos del 50.11 % y 77.98 % para 2-GPM y 4-GPM respectivamente.

Con el objetivo de explicar por qué algunos benchmarks obtienen un mayor *speedup* para la configuración 4-GPM frente a 2-GPM, ponemos el foco en los diferentes valores de peticiones remotas obtenidos para ambas configuraciones. En la Tabla III recogemos todos los benchmarks evaluados, ordenados de menor a mayor conforme a la diferencia que existe entre el porcentaje de peticiones remotas observadas para ambas configuraciones. Los dos primeros benchmarks (HS, 3DC) de la tabla muestran un mejor comportamiento en 4-GPM, mostrando que ante la reducción de la diferencia entre el porcentaje de accesos remotos entre ambas configuraciones 4-GPM es capaz de aprovechar mejor los recursos que 2-GPM. Sin embargo, esto no es así para todos los benchmarks en la tabla y no podemos hacer una generalización únicamente a partir de este dato, ya que el resto de benchmarks para los que 4-GPM obtiene un mayor *speedup* (BT, BFS) se encuentran en posi-

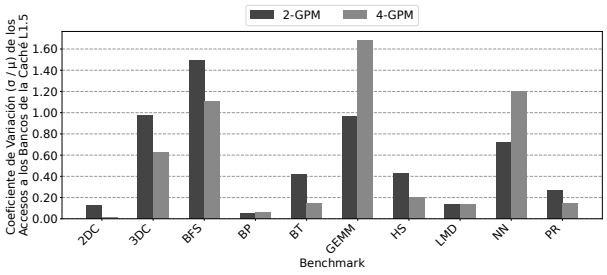


Fig. 8: Comparación del coeficiente de variación de los accesos a los bancos de la caché L1.5 entre configuraciones MCM-GPU

ciones más bajas de la tabla.

En la Figura 8 podemos observar, para cada uno de los benchmarks evaluados, el coeficiente de variación de los accesos a los distintos bancos de la caché L1.5 para las dos configuraciones MCM-GPU. A partir de estos valores, vemos como efectivamente, para los 4 benchmarks que obtienen un mejor *speedup* para 4-GPM, se está haciendo una mejor repartición de los accesos entre los bancos disponibles en la caché L1.5 frente a la configuración 2-GPM ya que obtenemos valores más cercanos a 0. Para el resto de benchmarks, los resultados obtenidos son consistentes cuando los analizamos en relación con el porcentaje de peticiones remotas evaluadas. Esto nos permite identificar que cuando una de estas dos variables se ve favorecida de forma significativa en una de las configuraciones, dicha configuración presenta un mayor *speedup*; lo que nos demuestra que tanto el acceso a datos remotos como la correcta explotación de los recursos locales, en este caso los bancos de la caché L1.5, son dos aspectos clave que debemos seguir mejorando en futuras versiones del simulador.

VI. Conclusiones

Los sistemas GPU se han implantado en la actualidad como la solución más extendida para la aceleración de grandes cargas computacionales mediante su procesamiento en paralelo. Con el objetivo de superar las barreras físicas que limitan su expansión, las arquitecturas MCM-GPU son una de las soluciones más utilizadas en la actualidad, existiendo aún un gran abanico de mejoras por explorar. En este trabajo, presentamos un simulador en construcción destinado al modelado de MCM-GPUs para su caracterización y la evaluación de nuevas propuestas. La evaluación realizada sobre la versión actual del simulador muestra que el desarrollo realizado sigue el camino esperado y nos revela los próximos puntos a mejorar para obtener una mayor fiabilidad respecto a los sistemas reales que encontramos en la industria y los evaluados en el estado del arte.

Agradecimientos

Trabajo realizado en el contexto del proyecto PID2022-136315OB-I00 financiado por MICIU/AEI/10.13039/501100011033/ y "FEDER Una manera de hacer Europa", UE. También por MICIU/AEI/10.13039/501100011033 y la "Unión Europea NextGenerationEU/PRTR", bajo la subvención RYC2021-031966-I. Este trabajo también ha recibido financiación del Consejo Europeo de Investigación (ERC) en el marco del programa de investigación e innovación Horizonte 2020 de la Unión Europea (ECHO, acuerdo de subvención n.º 819134). Germán Vicente ha sido financiado mediante la ayuda de Formación de Profesorado Universitario FPU24/02687 del Ministerio de Ciencia, Innovación y Universidades.

Referencias

[1] Hans Meiling, Edwin Boon, Nico Buzing, Kevin Cummings, Olav Frijns, Judy Galloway, Mieke Goethals, Noreen Harned, Bas Hultermans, Roel De Jonge, et al., "Performance of the full field euv systems," in *Emerging Lithographic Technologies XII*. SPIE, 2008, vol. 6921, pp. 171–183.

[2] NVIDIA Corporation, "NVIDIA DGX-1 V100 System Architecture," `https://images.nvidia.com/content/pdf/dgx1-v100-system-architecture-whitepaper.pdf`, 2018.

[3] NVIDIA Corporation, "NVIDIA Blackwell Architecture Technical Brief," `https://resources.nvidia.com/en-us-blackwell-architecture/blackwell-architecture-technical-brief`, 2024.

[4] Kyle Aubrey, "Inside the nvidia vera rubin platform: Six new chips, one ai supercomputer," January 2026.

[5] Advanced Micro Devices, Inc., "AMD CDNA™ 2 Architecture White Paper," `https://www.amd.com/content/dam/amd/en/documents/instinct-business-docs/white-papers/amd-cdna2-white-paper.pdf`, 2022.

[6] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers, "Accel-sim: An extensible simulation framework for validated gpu modeling," in *Proceedings of the 47th Annual ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2020, pp. 473–486.

[7] Akhil Arunkumar, Evgeny Bolotin, Benjamin Cho, Ugljesa Milic, Eiman Ebrahimi, Oreste Villa, Aamer Jaleel, Carole-Jean Wu, and David Nellans, "Mcm-gpu: Multi-chip-module gpus for continued performance scalability," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 320–332, 2017.

[8] Anthony Gutierrez, Bradford M. Beckmann, Alexandru Dutu, Joseph Gross, Michael LeBeane, John Kalamatianos, Onur Kayiran, Matthew Poremba, Brandon Potter, Sooraj Puthoor, Matthew D. Sinclair, Mark Wyse, Jieming Yin, Xianwei Zhang, Akshay Jain, and Timothy Rogers, "Lost in Abstraction: Pitfalls of Analyzing GPUs at the Intermediate Language Level," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 608–619.

[9] "Amd vega gpu model documentation," 2026.

[10] "Full system amd gpu model: Gpufs documentation," 2026.

[11] Advanced Micro Devices (AMD), "Introducing amd cdna™ 3 architecture," Tech. Rep., Advanced Micro Devices (AMD), 2023.

[12] Yifan Sun, Trinayan Baruah, Saiful A. Mojumder, Shi Dong, Xiang Gong, Shane Treadway, Yuhui Bao, Spencer Hance, Carter McCardwell, Vincent Zhao, Harrison Barclay, Amir Kavyan Ziabari, Zhongliang Chen, Rafael Ubal, José L. Abellán, John Kim, Ajay Joshi, and David Kaeli, "Mgpusim: enabling multi-gpu performance modeling and optimization," in *Proceedings of the 46th International Symposium on Computer Architecture*, New York, NY, USA, 2019, ISCA '19, p. 197–209, Association for Computing Machinery.

[13] Advanced Micro Devices (AMD), "Amd gcn3 instruction set architecture," Tech. Rep., Advanced Micro Devices, Inc., Aug. 2016.

[14] Preyesh Dalmia, Rajesh Shashi Kumar, and Matthew D. Sinclair, "Cpelide: Efficient multi-chiplet gpu implicit synchronization," in *Proceedings of the 2024 57th IEEE/ACM International Symposium on Microarchitecture*. 2024, MICRO '24, p. 700–717, IEEE Press.

[15] Mario Ibáñez Bolado, Borja Pérez Pavón, and Jose Luis Bosque Orero, "Rex: A remote execution model for continuos scalability in multi-chiplet-module gpus," *Future Generation Computer Systems*, p. 108268, 2025.

[16] Junhyeok Park, Sungbin Jang, Osang Kwon, Yongho Lee, and Seokin Hong, "Leveraging chiplet-locality for efficient memory mapping in multi-chip module gpus," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, 2025, pp. 1040–1057.

[17] Hossein SeyyedAghaei, Mahmood Naderan-Tahan, Magnus Jahre, and Lieven Eeckhout, "Memory-centric mcm-gpu architecture," *IEEE Computer Architecture Letters*, vol. 24, no. 1, pp. 101–104, 2025.

[18] Xiangrong Xu, Liang Wang, Limin Xiao, Lei Liu, Zihao Zhou, Yuanqiu Lv, Li Ruan, Xilong Xie, Meng Han, and Xiaojian Liao, "Exploiting intra-chip locality for multi-chip gpus via two-level shared l1 cache," *Journal of Systems Architecture*, vol. 167, pp. 103500, 2025.

[19] Shiqing Zhang, Mahmood Naderan-Tahan, Magnus Jahre, and Lieven Eeckhout, "Balancing performance against cost and sustainability in multi-chip-module gpus," *IEEE Computer Architecture Letters*, vol. 22, no. 2, pp. 145–148, 2023.

[20] Shiqing Zhang, Mahmood Naderan-Tahan, Magnus Jahre, and Lieven Eeckhout, "Sac: Sharing-aware caching in multi-chip gpus," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[21] Xu Zhang, Guangda Zhang, Lu Wang, Shiqing Zhang, and Xia Zhao, "Adcoalescer: An adaptive coalescer to reduce the inter-module traffic in mcm-gpus," in *Proceedings of the 53rd International Conference on Parallel Processing*, 2024, pp. 1001–1011.

[22] Baiqing Zhong, Zhirong Ye, Xiaojie Li, Peilin Wang, Haiqiu Huang, Zhaolin Li, Zhiyi Yu, and Mingyu Wang, "Lrm-gpu: Alleviating synchronization overhead for multi-chiplet gpu architecture," in *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2026, pp. 1–14.

[23] B. Ramakrishna Rau, "Pseudo-randomly interleaved memory," *SIGARCH Comput. Archit. News*, vol. 19, no. 3, pp. 74–83, Apr. 1991.

[24] NVIDIA Corporation, "Nvidia tesla V100 GPU architecture," August 2017.

[25] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the International Symposium on Workload Characterization (IISWC)*. 2009, pp. 44–54, IEEE.

[26] Shuai Che, Bradford M. Beckmann, Steven K. Reinhardt, and Kevin Skadron, "Pannotia: Understanding irregular gpgpu graph applications," in *2013 IEEE International Symposium on Workload Characterization (IISWC)*. 2013, pp. 185–195, IEEE.

[27] John Cavazos and Scott Grauer-Gray, "Polybench/gpu 1.0," `https://web.cse.ohiostate.edu/~pouchet.2/software/polybench/`, 2015, Online; accessed 2022-04-16.

Estrategias para implementar Propagación de Momentos en Redes Neuronales Bayesianas

Samuel Pérez Pedrajas[1], Javier Resano[1] and Darío Suárez Gracia[1]

Resumen— **En aplicaciones de inteligencia artificial con implicaciones severas, como puede ser la conducción autónoma o la medicina, se deben aportar estimaciones de incertidumbre fiables en las decisiones. Aunque las Redes Neuronales Bayesianas (BNNs) cuantifican esta incertidumbre, presentan un sobrecoste computacional significativo, pudiendo requerir hasta 514 veces más energía que una red neuronal tradicional (NN) al ejecutarse en GPU. Este coste se debe a su naturaleza probabilística, que exige un muestreo extensivo de distribuciones y múltiples ejecuciones para calcular los estimadores de Monte Carlo de la distribución de salida. Para abordar este cuello de botella, este trabajo propone tres estrategias de inferencia para BNNs destinadas a mejorar la eficiencia energética y la sostenibilidad, cada una adaptada a arquitecturas de modelos específicas. Basándose en el algoritmo de Propagación de Momentos (MP), estas optimizaciones reducen significativamente la necesidad de muestreos exhaustivos y múltiples ejecuciones, preservando al mismo tiempo la fiabilidad de la incertidumbre. Las evaluaciones experimentales demuestran que estas metodologías pueden reducir el sobrecoste de las BNNs solamente x3 respecto a las NN.**

Palabras clave— **Aprendizaje Automático, Optimización de rendimiento, Redes Neuronales Bayesianas.**

I. Introducción

LAS redes neuronales (NNs) se han convertido en una herramienta estándar para tareas de clasificación. Sin embargo, a menudo están mal calibradas y son propensas al sobreajuste con los datos de entrenamiento [1]. Estas limitaciones son especialmente críticas en contextos que requieran fiabilidad. El aprendizaje automático probabilístico permite cuantificar la incertidumbre predictiva y derivar intervalos de confianza, mejorando la explicabilidad, fiabilidad y transparencia de los modelos [2], [3].

Los enfoques de aprendizaje automático probabilístico para NNs incluyen el Monte Carlo (MC) Dropout [4], los Deep Ensembles [5] y las Redes Neuronales Bayesianas (BNNs). El Dropout y los métodos de ensembles aproximan la incertidumbre predictiva introduciendo aleatoriedad durante la inferencia o agregando múltiples modelos entrenados de forma independiente. Las BNNs formulan las NNs mediante un enfoque probabilístico, lo que permite descomponer la incertidumbre predictiva en la incertidumbre proveniente de la naturaleza de los datos y aquella derivada de la falta de conocimiento del modelo [6].

La necesidad de una cuantificación de incertidumbre se ilustra mejor en dominios críticos para la seguridad, donde una predicción excesivamente segura pero incorrecta puede tener consecuencias catastróficas. Como se muestra en la Fig. 1, las BNNs proporcionan un mecanismo para identificar salidas ambiguas mediante la incertidumbre. Esto puede permitir la implementación de sistemas tipo *human-in-the-loop*, donde los casos con una alta incertidumbre se remiten a un experto humano.

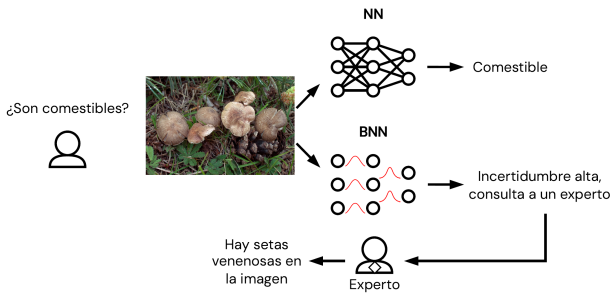


Fig. 1: Dos especies micológicas mezcladas, *Tricholoma terreum* y *Lepiota brunneoincarnata*. Mientras que la primera es una especie comestible muy apreciada, la segunda puede ser mortal. Este ejemplo ilustra la importancia de la incertidumbre proporcionada por las BNN en escenarios tipo *human-in-the-loop*.

La adopción de las BNN sigue viéndose obstaculizada por los sobrecostes de computación y memoria, presentando un coste energético de inferencia hasta 514 veces mayor. La inferencia estándar en BNN requiere múltiples ejecuciones de la red, en las cuales se realiza un muestreo de las distribuciones de los pesos, lo que es considerablemente más complejo que leer un valor fijo. Además, la inherente baja localidad temporal del proceso tienen un impacto negativo en jerarquías de memoria convencionales. A medida que aumenta la profundidad de la red, los pesos de las capas posteriores expulsan a los de las capas anteriores de las cachés de nivel inferior. Esto significa que cada pasada hacia adelante resulta en fallos de caché repetidos y transferencias desde la memoria principal, lo que degrada severamente el rendimiento.

El algoritmo de Propagación de Momentos (MP) es una alternativa prometedora para la inferencia de BNN. Consiste en operar directamente con distribuciones, en lugar de tener que muestrearlas varias veces para obtener datos. Sin embargo, las librerías más importantes para aprendizaje profundo probabilístico, como TensorFlow Probability [7], BayesianTorch [8] o Pyro [9], no incluyen soporte para MP.

La adopción generalizada de las BNN requiere una reducción de los costes de inferencia para garantizar la sostenibilidad energética. Alcanzar estos objetivos exige el desarrollo de bibliotecas optimizadas. Las contribuciones específicas de este trabajo son las siguientes:

[1]Departamento de Informática e Ingeniería de Sistemas (DIIS), Instituto Universitario de Investigación en Ingeniería de Aragón (I3A), Universidad de Zaragoza, HiPEAC European NoE. {samuel.perez,jresano,dario}@unizar.es

- Un estudio comparativo de aproximaciones analíticas de la función de activación Rectified Linear Unit (ReLU) para MP.
- La propuesta de funciones de activación y de pooling simplificadas para MP, junto con una evaluación de tres estrategias de ejecución distintas, cada una adaptada a un caso de uso específico, y su impacto en las métricas predictivas.
- El desarrollo de una biblioteca para ejecutar BNNs utilizando MP diseñada para integrarse con BayesianTorch[1].
- Un análisis de rendimiento y evaluación de las implementaciones de BNN en una GPU NVIDIA.

II. Redes Neuronales Bayesianas

Los pesos de las BNN en lugar de ser valores escalares fijos, son distribuciones de probabilidad, típicamente parametrizadas por medias y varianzas. Según el Teorema de Bayes, el entrenamiento de una BNN consiste en aprender un conjunto de pesos que aproxime la distribución a posteriori $p(w|D)$ dado el conjunto de datos de entrenamiento $D = \{x, y\}$. La inferencia variacional se utiliza comúnmente para este propósito, apoyándose en un conjunto de distribuciones gaussianas para aproximar la distribución a posteriori. Como resultado, el modelo entrenado consiste en un conjunto de medias y varianzas que sirven como los pesos de la red [10]. Una vez que el modelo está entrenado, se utiliza muestreo MC para aproximar la integral de la Ecuación 1, que de otro modo requeriría evaluar todas las configuraciones posibles de los pesos.

$$\begin{aligned} p(y^*|x^*, D) &= \int p(y|x^*, w)p(w|D)dw \\ &\approx \frac{1}{T}\sum_{t=0}^{T} p(y_t|x^*, w_t) \end{aligned} \tag{1}$$

En la práctica, se extrae un conjunto de pesos w_t de las distribuciones a posteriori aprendidas durante una ejecución. Esto significa que cada inferencia consta de T ejecuciones, cada una utilizando un conjunto diferente de pesos muestreados.

A. Propagación de Momentos

La Propagación de Momentos consiste en operar directamente sobre distribuciones de probabilidad en lugar de hacerlo sobre muestras [11]. La incertidumbre se propaga a través de la red mediante el seguimiento de los momentos de las activaciones, específicamente sus medias y varianzas. De acuerdo con el Teorema Central del Límite, la salida de una operación de multiplicación y acumulación (MAC), al ser la suma de múltiples variables aleatorias independientes, puede aproximarse mediante una distribución gaussiana. Como resultado, la distribución se preserva a través de las capas. Para las capas compuestas por operaciones lineales, estos momentos pueden calcularse analíticamente. La Ecuación 2 define la operación MAC utilizada para calcular los momentos de la activación resultante.

$$\begin{aligned} \mu_o &= \sum_{i=1}^{N} \mu_{xi}\mu_{wi} \\ \sigma_o^2 &= \sum_{i=1}^{N} \sigma_{xi}^2\sigma_{wi}^2 + \sigma_{xi}^2\mu_{wi}^2 + \sigma_{wi}^2\mu_{xi}^2 \end{aligned} \tag{2}$$

Este algoritmo aumenta el número de operaciones aritméticas y aproximadamente duplica el tamaño de los búferes de activación intermedios, ya que se deben propagar tanto las medias como las varianzas. Otro desafío que surje con esta aproximación es el manejo de las capas no lineales, como ReLU, MaxPool o Softmax, para las cuales la propagación analítica de momentos no es directa.

III. Trabajo Relacionado

Debido al elevado coste computacional de las BNNs el trabajo previo se ha dedicado en su mayoría a intentar reducir la complejidad computacional de las BNNs. El muestreo MC ha sido el enfoque más adoptado para la inferencia en BNNs. Trabajos previos han desarrollado aceleradores discretos en FPGA que utilizan distribuciones de probabilidad de Bernoulli en lugar de distribuciones gaussianas para mitigar el coste del muestreo [12].

En el contexto del despliegue de BNNs en sistemas empotrados y de intentar reducir el coste del muestreo, se introdujo una extensión de la ISA de RISC-V para acelerar las operaciones de muestreo y reducir la latencia de inferencia [13]. Esta técnica reduce el coste efectivo de los pasos de muestreo individuales, pero sigue dependiendo de múltiples ejecuciones, lo que continúa siendo un cuello de botella fundamental para aplicaciones en tiempo real o con restricciones de recursos.

Siguiendo en las aproximaciones basadas en muestreo MC, otros trabajos también proponen estrategias de cuantificación para BNNs. La biblioteca BayesianTorch incorpora técnicas cuantificación clásicas [14]. Asimismo, otros trabajos han implementado esquemas de cuantificación logarítmica para reducir la complejidad computacional preservando las estimaciones de incertidumbre mediante el entorno Pyro [15].

Por otro lado, los enfoques basados en MP tienen como objetivo eliminar el muestreo y las múltiples pasadas hacia adelante mediante la propagación analítica de la incertidumbre. Trabajos previos aproximan de forma analítica las funciones de activación no lineales, introduciendo variables Bernoulli para simplificar el cálculo de los momentos [16]. Los autores mostraron estos métodos en modelos de clasificación como LeNet y excluyen explícitamente las capas de pooling debido a la dificultad de manejar operaciones no lineales más allá de ReLU.

Otros trabajos proponen aproximaciones analíticas de los momentos para la función de activación Leaky ReLU [17]. Sin embargo, la validación experimental se limita a modelos de regresión pequeños. Los mismos autores introdujeron posteriormente el uso de

[1] https://anonymous.4open.science/r/empbnn-D650/

la unscented transform, una técnica utilizada en los Filtros de Kalman, para aproximar la propagación de la incertidumbre a través de activaciones ReLU utilizando un número pequeño de muestras deterministas. Sugiriendo que esta técnica podría generalizarse a otros tipos de capas, pero los resultados están nuevamente restringidos a modelos de regresión a pequeña escala [18].

Wu *et al.* proponen aproximar los momentos de una distribución gaussiana al pasar por activaciones no lineales como ReLU utilizando una función analítica compleja, demostrando resultados prometedores en modelos de regresión pequeños [11].

Por ultimo, otro trabajo reciente presenta un método de propagación analítica para activaciones ReLU y proporciona una comparación entre la inferencia MC y MP. Demuestran que MP escala bien tanto en modelos pequeños como grandes [19]. Esta aproximación tiene dos limitaciones. Primero, implica operaciones computacionalmente intensivas, como divisiones y raíces cuadradas, así como la PDF y la CDF de la gaussiana estándar, que podrían implementarse mediante dos lookup tables para reducir su coste computacional. Segundo, no puede utilizarse como remplazo directo sin reentrenamiento de los modelos.

Las estrategias propuestas en nuestro trabajo son complementarias a la mayoría de trabajos previos, ya que buscamos generalizar el algoritmo MP para cualquier BNN.

IV. Estrategias para Propagación de Momentos

La Fig. 2 muestra una comparación simplificada de las diferentes configuraciones para inferencia de BNNs junto nuestras tres propuestas. Para cada configuración, la figura detalla la composición de capas correspondiente y proporciona un modelo representativo que incorpora todos los tipos de capas estudiados.

En particular, las configuraciones ilustradas se organizan de la siguiente manera. Primero, se muestran 2 configuraciones existentes:

- Base MC: La primera fila muestra un algoritmo estándar de inferencia por muestreo MC para BNNs.
- Trabajo previo de MP: La segunda fila representa el trabajo previo que utiliza una aproximación de ReLU sin muestreo, la cual carece de una solución para las capas de pooling no lineales.

Y después, las 3 propuestas de este trabajo:

- Mixto: La tercera fila ilustra la propuesta inicial para la inferencia por MP, en la cual las capas no lineales se ejecutan en modo de muestreo MC.
- Simple: La cuarta fila detalla la transición a aproximaciones simplificadas para las capas no lineales, eliminando el requisito de muestreo en estas etapas.
- Split: La fila final presenta la estrategia Split, donde las ultimas capas se mantienen como capas de muestreo para mitigar degradación de las métricas de incertidumbre.

A. Estrategia Mixta

Para soportar múltiples capas, tanto lineales como no lineales, se propone una configuración de inferencia mixta en el que las operaciones lineales utilizan MP, mientras que las operaciones no lineales se gestionan mediante inferencia por muestreo MC (Mixto en la Fig. 2). Cuando aparece una función no lineal, el proceso de inferencia cambia a un modo de muestreo, aplicando la operación T veces a las activaciones muestreadas. Los momentos de salida se estiman a continuación a partir de las muestras. Para limitar el uso de memoria, se utilizan algoritmos en línea para la estimación de la media y la varianza, eliminando la necesidad de almacenar muestras intermedias.

Cambiar a la inferencia por muestreo requiere el cálculo de la desviación típica a partir de la varianza propagada y el proceso de muestreo exige conocer la familia de la distribución. Las activaciones resultantes de las operaciones MAC que involucran un gran número de operandos pueden aproximarse razonablemente como gaussianas considerando el Teorema Central del Límite (TCL). Si bien las activaciones pueden tratarse como gaussianas, se asume nuevamente que las salidas resultantes son gaussianas cuando se reanuda la inferencia MP para las siguientes capas. Dado que la aplicación de una función no lineal a una gaussiana no produce, en general, una distribución gaussiana, esto provoca una acumulación de error de aproximación.

Este trabajo también aprovecha el TCL para simplificar el proceso de muestreo. Siguiendo los enfoques de [12] y [13], la distribución de muestreo se sustituye por una alternativa más sencilla que preserva los momentos. Específicamente, se utiliza una distribución uniforme en lugar de una gaussiana, ya que puede muestrearse con un coste significativamente menor. Las transformaciones correspondientes utilizadas se definen en la Ecuación 3.

$$\begin{aligned} b &= \sigma\sqrt{12} \\ a &= \mu - b/2 \\ \sigma\ N(0,1) + \mu &\rightarrow a\ U(0,1) + b \end{aligned} \tag{3}$$

Como evaluación preliminar del impacto de estas aproximaciones y optimizaciones, se he realizado un estudio sencillo para cuantificar el error introducido en una función ReLU. La Fig. 3 compara los momentos de salida resultantes utilizando las aproximaciones propuestas frente a los resultados de diez mil muestras gaussianas transformadas por una ReLU. Los resultados indican que reemplazar la distribución no introduce un error adicional más allá del que ya se incurre al cambiar de MP a una inferencia basada en muestreo.

La Sección VI contiene los resultados experimentales obtenidos utilizando esta configuración. Esta primera estrategia ya supone una mejora significativa en el rendimiento con respecto al muestreo MC.

B. Estrategia Simple

Ya que la función no lineal ReLU es muy común, este trabajo introduce una alternativa más sencilla

Base MC	MC Conv	MC ReLU	MC Add	MC Max Pool	MC Linear	MC Softmax
Previo	MP Conv	Approx ReLU	MP Add	Undef	MP Linear	MC Softmax
Mixto	MP Conv	MC ReLU	MP Add	MC Max Pool	MP Linear	MC Softmax
Simple	MP Conv	Simple ReLU	MP Add	S. Max Pool	MP Linear	MC Softmax
Split	MP Conv	Simple ReLU	MP Add	S. Max Pool	MC Linear	MC Softmax

Fig. 2: Configuraciones de BNNs analizadas en este trabajo, las capas coloreadas indican que requieren muestreo MC.

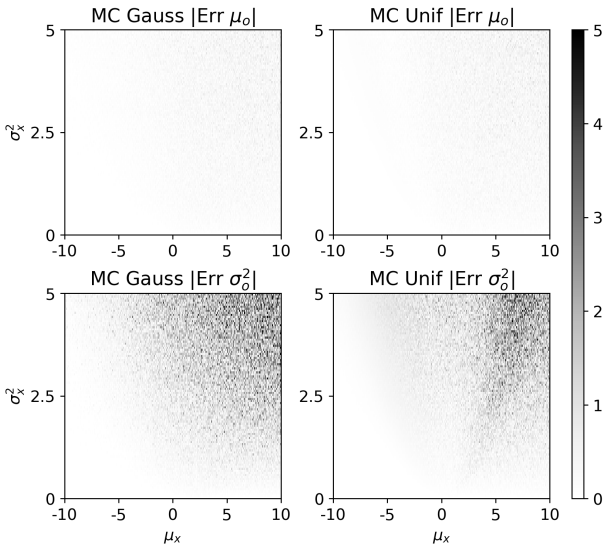


Fig. 3: Análisis de error de la media y varianza. Se evalúan los resultados para 100 muestras MC, utilizando muestreo Gaussiano (N) y Uniforme (U).

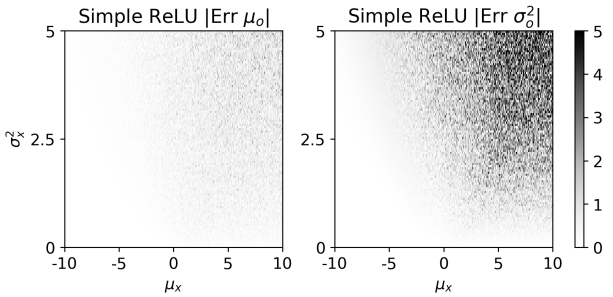


Fig. 4: Análisis de error de la media y varianza para la aproximación ReLU Simple.

para la inferencia MP con el fin de reducir el impacto computacional del muestreo en la aproximación híbrida. Este enfoque se basa únicamente en la media de la distribución, la Ecuación 4 presenta dicha aproximación. Dado que este trabajo se centra en la inferencia, las propiedades de entrenamiento del método propuesto quedan fuera de su alcance.

$$\begin{cases} \mu_o = \mu_x,\ \sigma_o^2 = \sigma_x^2 & \mu_x > 0 \\ \mu_o = 0,\ \sigma_o^2 = 0 & \mu_x \leq 0 \end{cases} \tag{4}$$

La aproximación propuesta introduce errores en los casos en que la media de la distribución gaussiana está cerca de cero con una varianza elevada. La Fig. 4 muestra una evaluación de este error. Este tipo de error limita el uso de esta aproximación en modelos con activaciones internas en estas regiones, ya que puede afectar significativamente a los resultados.

Esta metodología se extiende a las capas de max pooling, donde la media y la varianza de la distribución con la media más alta se propagan de manera similar (Simple en la Fig. 2). La eficacia de esta configuración se demuestra en la sección VI.

C. Estrategia Split

Los resultados experimentales indican que las métricas de incertidumbre se degradan más en arquitecturas pequeñas al utilizar MP. Investigaciones previas sugieren que las últimas capas de la red son las más importantes para calibrar la incertidumbre [20]. Adoptando este enfoque, proponemos una estrategia donde las capas iniciales utilizan MP mientras que la capa final se mantiene como una capa de muestreo (Split en la Fig. 2). Esta metodología mitiga la degradación observada en las métricas de incertidumbre con un impacto mínimo en el rendimiento, ya que la capa final representa una pequeña parte del cómputo total del modelo.

V. Metodología

El equipo en el que se han realizado los experimentos usa una CPU AMD Ryzen 9 5950X y una GPU NVIDIA GeForce RTX 4090 ejecutando Ubuntu 24.04 LTS.

Para las mediciones relacionadas con la GPU, la velocidad de los ventiladores se bloquearon al 40 % como mínimo y las frecuencias de los núcleos y memoria se fijaron en 2550 y 10251 MHz respectivamente, siendo estas las frecuencias pico observadas durante una ejecución inicial sin restricciones. Los valores de consumo se han obtenido con la herramienta `nvidia-smi`, incorporando periodos de inactividad entre experimentos para asegurar estabilidad térmica.

Para la evaluación, este trabajo considera 4 arquitecturas de BNN diferentes utilizadas en estudios relevantes y recientes sobre BNN. Todos los modelos fueron entrenados utilizando el marco de trabajo MOPED dentro de BayesianTorch empleando precisión de punto flotante [8].

- Modelo de Clasificación de Píxeles Hiperespectrales (HYPER): Modelo pequeño diseñado para la clasificación de píxeles hiperespectrales en aplicaciones de detección satelital. El modelo tiene una arquitectura simple de tres capas lineales totalmente conectadas y se evalúa con el conjunto de datos Salinas Valley [21].
- LeNet-5 Bayesiana (LENET): Una BNN convolucional con la misma arquitectura del modelo

LeNet-5 [22]. Entrenada para la clasificación del conjunto de imágenes CIFAR-10 [23].
- B2N2: El modelo de evaluación introducido por Awano *et al.* para su acelerador de BNN [12]. El modelo sigue una arquitectura similar a la familia VGG [24].
- TinyResNet Bayesiana (RESNET): Una BNN residual basada en la arquitectura ResNet definida en el benchmark MLPerf™ Tiny [25].

Los trabajos relacionados emplean habitualmente hasta 100 muestras para lograr aproximaciones fiables de la distribución posterior, aunque ciertas aplicaciones toleran un número menor [6], [8], [12], [15], por lo que durante los experimentos también se han utilizado 100 muestras con precisión punto flotante y un tamaño de batch de 32.

La evaluación de las optimizaciones para las BNN debe medir tanto el rendimiento predictivo como la calidad de las estimaciones de incertidumbre. Dado un conjunto de predicciones $a_{tk} = p(y_k|x, w_t)$ sobre K clases y T muestras MC, la Ecuación 5 define la incertidumbre predictiva.

$$H(y|x, D) = -\sum_{k=1}^{K} \left[\left(\sum_{t=1}^{T} \frac{a_{tk}}{T} \right) \log \left(\sum_{t=1}^{T} \frac{a_{tk}}{T} \right) \right] \quad (5)$$

VI. Resultados experimentales

La Fig. 5 proporciona un conjunto de visualizaciones para un análisis detallado sobre el impacto de las optimizaciones propuestas en las métricas de evaluación. Estas incluyen diagramas de calibración, estimaciones de densidad de la incertidumbre predictiva, tanto para clasificaciones correctas como incorrectas, y probabilidades predictivas filtradas por un umbral de incertidumbre variable.

El diagrama de calibración muestra la relación entre la confianza predicha y la probabilidad empírica, una línea diagonal representaría una calibración perfecta. Para medir como de bien la incertidumbre diferencia entre las predicciones correctas e incorrectas, se utiliza una estimación de densidad de incertidumbre. En este caso, el escenario ideal es que muestre dos distribuciones claramente diferenciadas. Las distribuciones se obtienen aplicando un estimador de densidad gaussiano.

Finalmente, para evaluar el impacto del filtrado de predicciones basado en un umbral de incertidumbre, se muestra la probabilidad condicional p(Cierto | Seguro), donde seguro significa que la incertidumbre de una predicción está por debajo del umbral, frente a p(Seguro), que muestra la proporción del conjunto de datos con incertidumbre por debajo de dicho umbral. Esto se explica mejor con un ejemplo. En el caso de la Subfig. 5l, si se decide utilizar 0,2 como umbral de incertidumbre, descartando todos los valores de predicción por encima de ese umbral, la figura proporciona dos piezas clave de información para evaluar esa decisión: primero, informa que la precisión media para las predicciones que cumplen con el umbral sería del 96 %; segundo, indica que con ese umbral, solo se procesaría el 56 % de las muestras.

En modelo HYPER muestra una degradación en la calidad de la incertidumbre al cambiar al paradigma MP. La estrategia Split mitiga esta degradación de las métricas de incertidumbre. Los tres gráficos, Figs. 5a, 5b, 5c, muestran diferencias notables entre los enfoques MC, Mixto y Simple. En comparación, la estrategia Split muestra un comportamiento muy similar a MC.

La estrategia Mixta funciona bien en el modelo LENET, mostrando un impacto reducido en las métricas. Sin embargo, este modelo presenta la mayor degradación al utilizar la optimización Simple. La precisión disminuye del 62.7 % al 56.0 %. Este comportamiento es coherente con la mayor incertidumbre predictiva del modelo, lo que provoca que una mayor fracción de las activaciones ocurran en la zona de alto error de la ReLU Simple propuesta. Se puede observar en la Fig. 4 cómo los valores cercanos a cero con alta varianza presentan errores mayores. Las Figs. 5d y 5f muestran las respectivas degradaciones en la calibración y en la precisión general, pero la incertidumbre sigue comportándose correctamente. La Fig. 5e muestra que las distribuciones de incertidumbre para las predicciones correctas e incorrectas siguen siendo distinguibles.

Para modelos más grandes como B2N2 y RESNET, los métodos Mixto, Simple y Split muestran métricas casi idénticas con reducciones considerables en el tiempo y la energía de la inferencia. Las dos últimas filas de la Fig 5 ilustran estas afirmaciones.

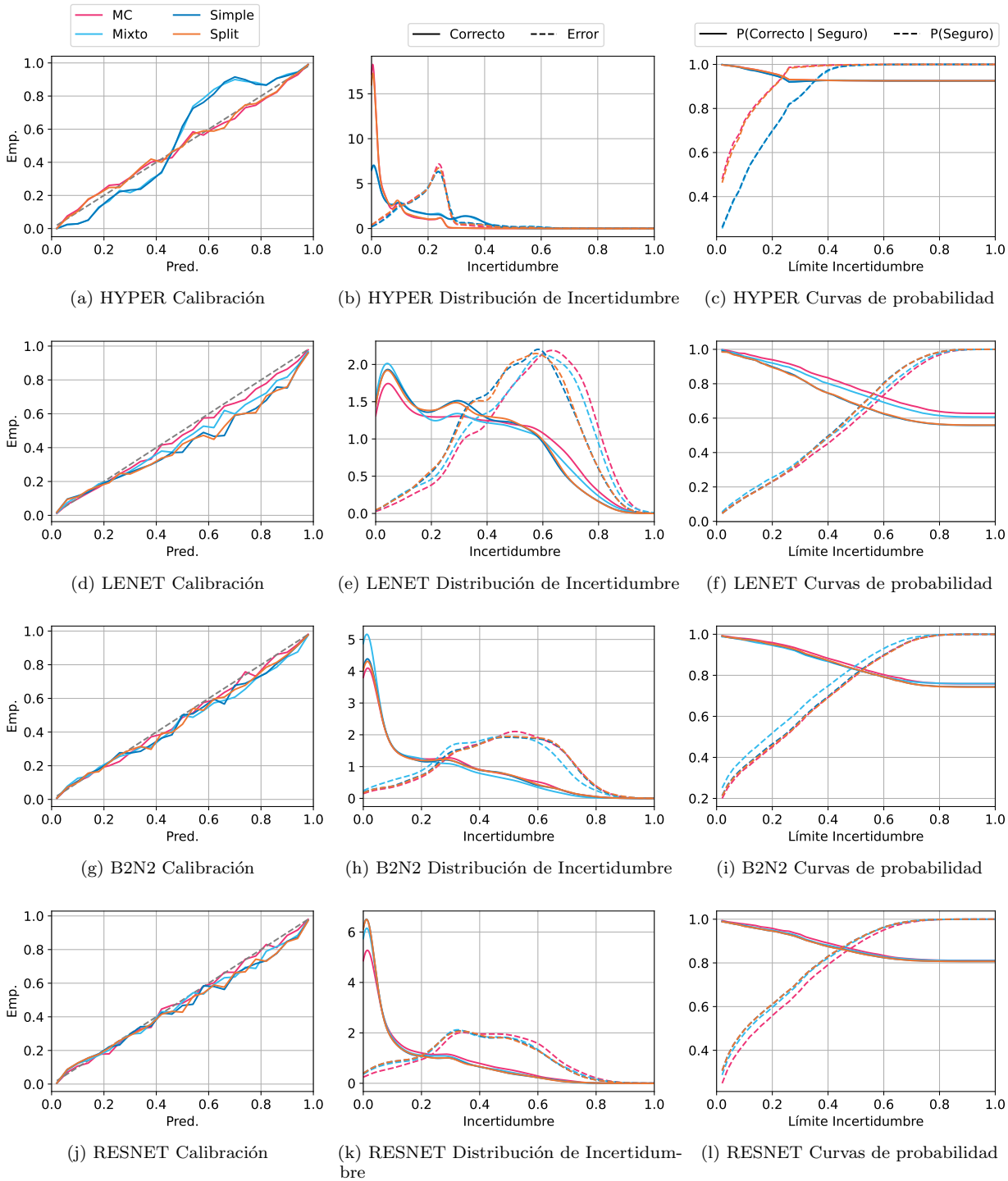
A. Evaluación de rendimiento

La implementación presentada en este trabajo incorpora varias optimizaciones, incluyendo la reutilización de muestras de MC a través de los elementos de un batch. Dado que las inferencias individuales son estadísticamente independientes, las muestras pueden compartirse sin alterar el proceso estocástico. Adicionalmente, se evalúan dos estrategias alternativas para la inferencia Mixta: un algoritmo en línea que minimiza el consumo de memoria mediante el cálculo en tiempo real de la media y la varianza, pero ejecutando mas kernels en la GPU, y una estrategia con búfer que reduce el número de kernels a ejecutar, aumentando el rendimiento pero también el uso de memoria.

Para el modelo LENET utilizando 100 muestras MC, el enfoque con búfer logra una aceleración de x2,95 a costa de un aumento de x13,78 del uso pico de memoria. De manera similar, el modelo RESNET muestra una aceleración de x7,43 con un sobrecoste de memoria de x13,67.

En cuanto a la evaluación de rendimiento, la Fig 6 muestra la latencia de inferencia en todas las configuraciones de los modelos. Esta figura incluye NN deterministas para resaltar la reducción en la diferencia de rendimiento lograda por los métodos propuestos. Nótese que los ejes verticales se presentan en una escala logarítmica para mostrar grandes diferencias de magnitud.

Para mayor claridad, las ganancias de rendimiento



(a) HYPER Calibración (b) HYPER Distribución de Incertidumbre (c) HYPER Curvas de probabilidad

(d) LENET Calibración (e) LENET Distribución de Incertidumbre (f) LENET Curvas de probabilidad

(g) B2N2 Calibración (h) B2N2 Distribución de Incertidumbre (i) B2N2 Curvas de probabilidad

(j) RESNET Calibración (k) RESNET Distribución de Incertidumbre (l) RESNET Curvas de probabilidad

Fig. 5: Análisis de incertidumbre de los modelos de prueba. La primera columna muestra gráficos de calibración. La segunda columna muestra la estimación de densidad de la incertidumbre para las predicciones correctas e incorrectas. La tercera columna muestra las probabilidades predictivas dado diferentes umbrales de incertidumbre.

para las configuraciones más efectivas para la inferencia, definidas como aquellas que maximizan el rendimiento sin comprometer las métricas del modelo, se detallan en la Tabla I. Las estrategias de optimización producen diferentes ganancias dependiendo de la arquitectura del modelo. El mejor caso de ganancia de rendimiento ocurre con el modelo RESNET, logrando una aceleración de x98.31. Las menores ganancias se observan en el modelo LENET, donde la aceleración se mantiene en x7.73. Esto ocurre porque el modelo LENET experimenta una degradación significativa de la precisión bajo la estrategia Simple y, por esa razón, se requiere el uso de la variante Mixta para mantener el rendimiento del modelo.

VII. Conclusiones

Para acelerar la inferencia en redes neuronales bayesianas (BNN) con independencia de su arquitectura, se presentan tres estrategias distintas basadas en la propagación de momentos (MP): una estrategia mixta donde las capas no lineales se ejecutan mediante muestreo Monte Carlo (MC); un método de aproximación

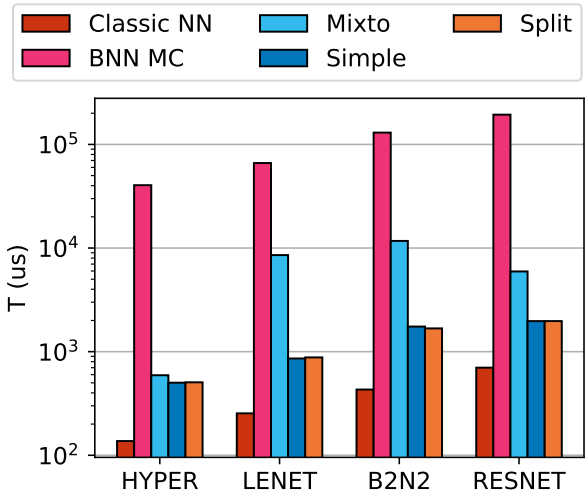


Fig. 6: Tiempo de inferencia de todos los modelos y configuraciones. El eje Y está en escala logarítmica.

Tabla I: Aceleraciones obtenidas con las configuraciones optimas para preservar incertidumbre.

Modelo	**Configuración**	**Speedup**
HYPER	Split	79,91
LENET	Mixto	7,73
B2N2	Simple	74,47
RESNET	Simple	98,31

simple para operaciones no lineales, específicamente ReLU y Max Pooling, que maximiza la eficiencia energética y el rendimiento; y la estrategia Split, que conserva una capa final de muestreo para preservar las métricas de incertidumbre en arquitecturas más pequeñas. Cada estrategia se adapta mejor a un escenario diferente: Split para modelos pequeños, mixta para modelos con alta incertidumbre y Simple para maximizar la eficiencia en modelos grandes y con baja incertidumbre.

Los resultados experimentales demuestran ganancias significativas en la eficiencia energética en relación con el algoritmo tradicional de muestreo MC implementado en la librería BayesianTorch. Esta metodología reduce el sobrecoste energético de las BNN sobre modelos deterministas de factores de x514 a x3. Además, utilizar la estrategia de ejecución óptima para la arquitectura del modelo garantiza que estas mejoras de rendimiento se logren sin ninguna degradación significativa en la precisión o en la calibración de la incertidumbre.

Este trabajo no cubre técnicas de cuantización para BNNs, en el futuro se explorarán para conseguir aumentar mas la eficiencia de este tipo de redes.

Agradecimientos

Esta investigación ha sido financiada por el proyecto PID2022-136454NB-C22, AEI/10.13039/501100011033, "ERDF A way of making Europe". Samuel Pérez Pedrajas cuenta con el apoyo de una beca de doctorado del Gobierno de Aragón.

Referencias

[1] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger, "On calibration of modern neural networks," 2017.

[2] Zoubin Ghahramani, "Probabilistic machine learning and artificial intelligence," *Nature*, vol. 521, no. 7553, pp. 452–459, May 2015.

[3] Umang Bhatt, Yunfeng Zhang, Javier Antorán, Q. Vera Liao, Prasanna Sattigeri, Riccardo Fogliato, Gabrielle Gauthier Melançon, Ranganath Krishnan, Jason Stanley, Omesh Tickoo, Lama Nachman, Rumi Chunara, Adrian Weller, and Alice Xiang, "Uncertainty as a form of transparency: Measuring, communicating, and using uncertainty," *CoRR*, vol. abs/2011.07586, 2020, `https://arxiv.org/abs/2011.07586`.

[4] Yarin Gal and Zoubin Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," 2016.

[5] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," 2017.

[6] Alex Kendall and Yarin Gal, "What uncertainties do we need in bayesian deep learning for computer vision?," 2017.

[7] Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matthew D. Hoffman, and Rif A. Saurous, "Tensorflow distributions," *CoRR*, vol. abs/1711.10604, 2017.

[8] Ranganath Krishnan, Mahesh Subedar, and Omesh Tickoo, "Specifying weight priors in bayesian deep neural networks with empirical bayes," vol. 34, pp. 4477–4484, Apr. 2020.

[9] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman, "Pyro: Deep universal probabilistic programming," *J. Mach. Learn. Res.*, vol. 20, pp. 28:1–28:6, 2019.

[10] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra, "Weight uncertainty in neural networks," 2015.

[11] Anqi Wu, Sebastian Nowozin, Edward Meeds, Richard E. Turner, José Miguel Hernández-Lobato, and Alexander L. Gaunt, "Deterministic variational inference for robust bayesian neural networks," 2019.

[12] Hiromitsu Awano and Masanori Hashimoto, "B2n2: Resource efficient bayesian neural network accelerator using bernoulli sampler on fpga," *Integration*, vol. 89, pp. 1–8, 2023.

[13] Samuel Perez Pedrajas, Javier Resano, and Darío Suárez Gracia, "Bnnrv: Hardware and software optimizations for weight sampling in bayesian neural networks on edge risc-v cores," *IEEE Transactions on Circuits and Systems for Artificial Intelligence*, vol. PP, pp. 1–12, 01 2025.

[14] Jun-Liang Lin, Ranganath Krishnan, Keyur Ruganathbhai Ranipa, Mahesh Subedar, Vrushabh Sanghavi, Meena Arunachalam, Omesh Tickoo, Ravishankar Iyer, and Mahmut Taylan Kandemir, "Quantization for bayesian deep learning: Low-precision characterization and robustness," in *2023 IEEE International Symposium on Workload Characterization (IISWC)*, 2023, pp. 180–192.

[15] Hendrik Borras, Yong Wu, Bernhard Klein, and Holger Fröning, "Uncertainty-preserving qbnns: Multi-level quantization of svi-based bayesian neural networks for image classification," 2025.

[16] Manuel Haußmann, Fred A. Hamprecht, and Melih Kandemir, "Sampling-free variational inference of bayesian neural networks by variance backpropagation," in *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, Ryan P. Adams and Vibhav Gogate, Eds. 22–25 Jul 2020, vol. 115 of *Proceedings of Machine Learning Research*, pp. 563–573, PMLR.

[17] David J. Schodt, Ryan Brown, Michael Merritt, Samuel Park, Delsin Menolascino, and Mark A. Peot, "A framework for variational inference of lightweight bayesian neural networks with heteroscedastic uncertainties," 2024.

[18] David J. Schodt, "Few-sample variational inference of bayesian neural networks with arbitrary nonlinearities," 2024.

[19] Yuki Hirayama and Shinya Takamaeda-Yamazaki, "Scalable moment propagation and analysis of variational distributions for practical bayesian deep learning," *IEEE*

Transactions on Neural Networks and Learning Systems, vol. 36, no. 3, pp. 4614–4624, 2025.

[20] James Harrison, John Willes, and Jasper Snoek, "Variational bayesian last layers," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*, 2024.

[21] Adrián Alcolea and Javier Resano, "Bayesian neural networks to analyze hyperspectral datasets using uncertainty metrics," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–10, 2022.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998, vol. 86, pp. 2278–2324.

[23] Alex Krizhevsky, "Learning multiple layers of features from tiny images," pp. 32–33, 2009.

[24] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[25] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, Urmish Thakker, Antonio Torrini, Peter Warden, Jay Cordaro, Giuseppe Di Guglielmo, Javier Duarte, Stephen Gibellini, Videet Parekh, Honson Tran, Nhan Tran, Niu Wenxu, and Xu Xuesong, "Mlperf tiny benchmark," June 2021.

Análisis de Paralelismo y Robustez de Arquitecturas de Deep Learning para HSI en Hardware Analógico

María B. García-Flores[1], Carlos Cañada-Rostro[1], Mercedes E. Paoletti[1], y Juan M. Haut [1]

Resumen— **La clasificación de imágenes hiperespectrales (HSI) requiere arquitecturas con alto grado de paralelismo para manejar la elevada dimensionalidad espectral. El hardware analógico neuromórfico permite ejecutar multiplicaciones matriz-vector de forma masiva en memoria, aunque introduce ruido estocástico en los pesos que degrada la precisión de inferencia. En este trabajo se analiza el impacto del patrón de paralelismo en la robustez frente a ruido analógico mediante `aihwkit`, evaluando nueve arquitecturas representativas, y se propone `Dual-Branch Transformer`, que combina una rama espacial y una espectral mediante ejecución concurrente. Los experimentos sobre cinco conjuntos de datos, con $\sigma \in [0{,}01, 0{,}60]$ y evaluación multi-semilla, muestran que el modelo propuesto alcanza la mayor OA en todos los casos, con mejoras estadísticamente significativas en cuatro de ellos, obteniendo valores entre el 82,93 % y el 96,29 % y una desviación estándar inferior a 1 pp en la mayoría de los casos. Los resultados indican que las arquitecturas con mayor paralelismo efectivo presentan una mayor robustez frente al ruido analógico.**

Palabras clave— **Imágenes hiperespectrales, hardware analógico neuromórfico, transformers, redes neuronales profundas, `aihwkit`.**

I. Introducción

LAS imágenes hiperespectrales capturan cientos de bandas espectrales continuas por píxel, permitiendo discriminar materiales con alta precisión en aplicaciones como teledetección, agricultura de precisión y vigilancia medioambiental [1], [2]. Sin embargo, esta alta dimensionalidad implica elevados requisitos computacionales, lo que motiva el uso de arquitecturas y paradigmas de ejecución altamente paralelos para el procesamiento eficiente de datos HSI.

La clasificación HSI ha experimentado una evolución significativa en la última década. Los métodos clásicos basados en máquinas de vectores de soporte (SVM) [3] y bosques aleatorios (RF) [4], que operan sobre el vector espectral del píxel central, fueron progresivamente superados por redes convolucionales capaces de explotar simultáneamente las dimensiones espacial y espectral del cubo hiperespectral [5], [6]. Más recientemente, los Vision Transformers (ViT) [7], [8] y los modelos de espacio de estados (SSM) [9], [10] han establecido nuevos estados del arte en benchmarks de referencia gracias a su capacidad para modelar dependencias de largo alcance, a costa de una creciente demanda computacional y energética.

[1]Departamento de Tecnología de Computadores y Comunicaciones, Universidad de Extremadura, correo: juanmariohaut@unex.es.

Esta demanda computacional representa un obstáculo crítico para el despliegue de modelos HSI en plataformas con recursos limitados, como satélites de observación terrestre, vehículos aéreos no tripulados o sistemas embebidos. En este contexto, el hardware analógico neuromórfico, basado en computación en memoria mediante matrices de conductancias programables, permite ejecutar multiplicaciones matriz-vector de forma altamente paralela con un consumo energético significativamente inferior al de las unidades de procesamiento gráfico convencionales [11]. No obstante, la variabilidad estocástica en la lectura de los pesos introduce perturbaciones durante la inferencia que degradan la precisión de los modelos entrenados bajo optimización digital [12], [13]. Este ruido de dispositivo constituye el principal obstáculo para el despliegue efectivo de modelos de aprendizaje profundo en este paradigma paralelo.

A pesar del interés creciente en la computación analógica neuromórfica, los estudios existentes se han centrado principalmente en tareas de visión general y procesamiento del lenguaje natural [12], sin abordar las particularidades de los datos HSI, caracterizados por alta dimensionalidad espectral, correlaciones espacio espectrales complejas y conjuntos de datos de tamaño reducido. En consecuencia, el impacto del ruido de dispositivo sobre el comportamiento de distintas arquitecturas en entornos de ejecución paralela no ha sido caracterizado de forma sistemática. En este contexto, se propone `Dual-Branch Transformer`, una arquitectura con paralelismo estructural que combina información espacial y espectral mediante fusión tardía por concatenación para explotar la complementariedad de ambas modalidades bajo ruido analógico.

En este contexto, las principales contribuciones de este trabajo son las siguientes:

- Se presenta una evaluación comparativa de nueve arquitecturas bajo simulación de hardware analógico mediante `aihwkit` [14], abarcando métodos clásicos, redes convolucionales, transformers y modelos de espacio de estados, desde la perspectiva de su robustez en entornos de ejecución paralela.
- Se analiza el comportamiento de un modelo SSM selectivo (Mamba) [9] en tareas de clasificación HSI bajo condiciones de ruido analógico, evidenciando las limitaciones de modelos con dependencias secuenciales en este contexto.
- Se propone el modelo `Dual-Branch`

`Transformer`, que integra información espacial y espectral mediante fusión tardía por concatenación, favoreciendo un mayor paralelismo efectivo y robustez frente al ruido.

El resto del artículo se organiza de la siguiente manera. La Sección II revisa los trabajos relacionados en clasificación HSI y computación analógica neuromórfica. La Sección III describe los conjuntos de datos, el entorno de simulación analógica y las arquitecturas evaluadas. La Sección IV presenta y discute los resultados experimentales, mientras que la Sección V recoge las conclusiones y las líneas de trabajo futuro.

II. Trabajos Relacionados

La clasificación de imágenes hiperespectrales (HSI) ha evolucionado desde métodos clásicos basados en características espectrales hacia arquitecturas de aprendizaje profundo capaces de explotar conjuntamente la información espacial y espectral [1], [2]. Las CNN-2D introdujeron el modelado explícito de la estructura espacial [5], mientras que las CNN-3D extendieron este enfoque mediante convoluciones volumétricas que capturan interacciones espacio espectrales de forma conjunta [6]. No obstante, la naturaleza local del campo receptivo convolucional limita su capacidad para modelar dependencias espectrales de largo alcance y restringe el paralelismo efectivo a nivel local.

Los Transformers abordan esta limitación mediante mecanismos de atención global que permiten capturar dependencias a lo largo de toda la dimensión espectral [7], [8], favoreciendo un mayor grado de paralelismo al operar sobre todos los tokens de forma simultánea. En el dominio HSI, diversos trabajos han explorado la integración de información espacial y espectral dentro de este paradigma, evidenciando la complementariedad de ambas modalidades [1], [2]. El modelo SSFIT [15] sigue esta línea mediante un tokenizador `Conv3D` específico para HSI. Por otra parte, los modelos SSM, como Mamba [9], han sido adaptados a HSI mediante estrategias de serialización del cubo de datos que permiten su procesamiento secuencial [10], lo que introduce dependencias temporales que limitan el paralelismo.

El despliegue de redes neuronales en hardware analógico neuromórfico introduce diversas fuentes de no idealidad, incluyendo ruido de programación, deriva temporal de la conductancia y no linealidades en dispositivos resistivos no volátiles [11]. Estas perturbaciones afectan directamente a las operaciones de multiplicación matriz vector, típicamente ejecutadas de forma altamente paralela en arquitecturas de computación en memoria, generando desviaciones respecto al modelo digital de referencia y degradando la precisión de inferencia.

Para mitigar estos efectos, el entrenamiento consciente del hardware incorpora modelos de ruido diferenciables durante el entrenamiento, permitiendo que la red aprenda representaciones robustas frente a las perturbaciones esperadas en inferencia [14], [16]. Trabajos previos han demostrado la efectividad de este enfoque en tareas de visión general y modelos CNN y Transformers [12], [13]. Sin embargo, su aplicación sistemática al dominio HSI y su impacto sobre el comportamiento de distintas arquitecturas en entornos de ejecución paralela sigue siendo limitada.

En este contexto, este trabajo extiende el análisis de robustez al ruido analógico a un conjunto diverso de arquitecturas HSI, incluyendo modelos basados en atención, modelos SSM y una arquitectura de doble rama que combina explícitamente información espacial y espectral, analizando su comportamiento desde la perspectiva del paralelismo y la robustez en hardware analógico.

III. Metodología

Se evalúan cinco conjuntos de datos HSI estándar: Indian Pines (IP), Pavia University (PU), Houston 2013 (HS), HongHu (HH) y HanChuan (HC), cuyas características se resumen en la Tabla I. Para HH y HC se realiza un submuestreo aleatorio sin reemplazo hasta 6.000 muestras antes de la partición, con semilla fija para garantizar la reproducibilidad. Cada muestra se define como un parche $X \in \mathbb{R}^{H\times W\times B}$, donde $H = W = 9$ y B es el número de bandas.

Todos los conjuntos se normalizan por banda mediante `StandardScaler`, ajustado únicamente sobre el conjunto de entrenamiento. La partición se realiza mediante dos divisiones estratificadas con `random_state`= 42, obteniendo 72 % para entrenamiento, 18 % para validación y 10 % para test. No se impone una separación espacial estricta entre particiones. Las métricas consideradas son OA, AA y Kappa.

Tabla I: Conjuntos de datos HSI evaluados.

Dataset	Sensor	Bandas	Clases	Muestras
IP	AVIRIS	200	16	∼10,249
PU	ROSIS	103	9	∼42,776
HS	CASI-1500	144	15	∼15,029
HH	CASI-1500	48	20	6,000*
HC	AVIRIS	154	14	6,000*

* Submuestreo previo al split.

La simulación de hardware analógico se realiza mediante `aihwkit` con la configuración `InferenceRPUConfig`, que convierte los módulos lineales en capas analógicas con `weight_scaling_omega`= 0,6. El ruido se modela como:

$$\hat{W} = W + E, \quad E_{ij} \sim \mathcal{N}(\prime, \sigma^{\in}), \tag{1}$$

donde W denota los pesos nominales, $\hat{W}$ los pesos perturbados y E un término de ruido gaussiano independiente aplicado elemento a elemento para todo (i, j). El parámetro σ controla el nivel de perturbación. Este modelo se emplea tanto durante el entrenamiento como durante la evaluación.

Durante el entrenamiento se emplea un nivel de ruido σ_{train}, mientras que en evaluación se considera $\sigma_{\text{eval}} \in \{0{,}01, 0{,}05, 0{,}10, 0{,}20, 0{,}40, 0{,}60\}$ sin reentrenamiento. Para Mamba se utiliza $\sigma_{\text{train}} = 0{,}05$ y

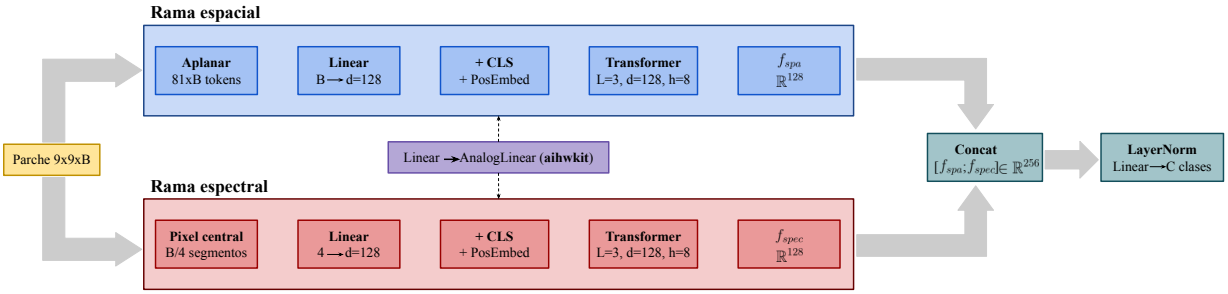


Fig. 1: Arquitectura de `Dual-Branch Transformer`. La rama espacial (azul) aplana el parche 9×9 en 81 tokens y los procesa mediante un Transformer con $L = 3$ bloques, extrayendo $\mathbf{f}_{\text{spa}}$ del token CLS. La rama espectral (naranja) divide la firma del píxel central en segmentos de 4 bandas y aplica la misma arquitectura Transformer, extrayendo $\mathbf{f}_{\text{spec}}$. Ambos vectores se concatenan en $\mathbf{f} \in \mathbb{R}^{2d}$ y se clasifican mediante `LayerNorm` y una proyección lineal. Las capas lineales se convierten a `AnalogLinear` para la simulación de hardware analógico con `aihwkit`.

para el resto $\sigma_{\text{train}} = 0{,}15$. La conversión analógica se aplica únicamente a capas `nn.Linear`.

Todos los modelos se entrenan con *batch size* = 128, AdamW ($lr = 10^{-3}$), entropía cruzada, parada temprana (paciencia = 20) y un máximo de 150 épocas. Los modelos basados en Transformer emplean adicionalmente Muon, con $lr = 5{\times}10^{-3}$, calentamiento lineal de 10 épocas y decaimiento coseno hasta 10^{-5}. Se utilizan 3 semillas {42, 123, 456}, ampliadas a 5 en CNN-2D y Mamba.

Desde el punto de vista computacional, las arquitecturas consideradas presentan distintos patrones de paralelismo que condicionan su comportamiento en hardware analógico. Las CNN explotan paralelismo local mediante operaciones convolucionales independientes, mientras que los Transformers permiten un paralelismo global al operar sobre todos los tokens de la secuencia de forma simultánea. En contraste, los modelos SSM como Mamba introducen dependencias secuenciales que limitan su paralelización efectiva. Este aspecto resulta especialmente relevante en hardware analógico basado en computación en memoria, donde las operaciones de multiplicación matriz-vector se ejecutan de forma altamente paralela.

Se evalúan nueve arquitecturas agrupadas en cuatro familias. ViT procesa secuencias de longitud 81 con dimensión de embedding $d = 128$, $L = 3$ bloques y 8 cabezas de atención. SSFIT emplea tokenización `Conv3D`. En los modelos basados en Transformer, el parche de entrada se transforma previamente en una secuencia de tokens embebidos, sobre la cual se aplican las operaciones de proyección y atención.

Los modelos Transformer operan sobre secuencias $X \in \mathbb{R}^{\mathrm{N}\times\mathrm{d}}$. Las proyecciones se definen como:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V, \tag{2}$$

donde W_Q, W_K y W_V son matrices de proyección aprendibles de tamaño $d \times d$. La operación de atención se define como:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V, \tag{3}$$

lo que permite modelar dependencias globales mediante interacciones entre todos los pares de tokens. Asimismo, esta formulación favorece un alto grado de paralelismo, ya que las operaciones de atención pueden ejecutarse de forma simultánea sobre la secuencia completa.

Mamba modela dependencias secuenciales mediante el siguiente sistema:

$$h_t = \bar{A}h_{t-1} + \bar{B}x_t, \quad y_t = Ch_t, \tag{4}$$

donde x_t, h_t e y_t son vectores de dimensión d, mientras que $\bar{A}$, $\bar{B}$ y C son matrices de tamaño $d \times d$. Esta dependencia recursiva introduce un carácter inherentemente secuencial que limita el paralelismo efectivo en comparación con arquitecturas basadas en atención.

`Dual-Branch Transformer` emplea dos ramas independientes, como se ilustra en la Fig. 1. Esta descomposición permite modelar de forma explícita dependencias espaciales y espectrales de manera separada, y habilita paralelismo estructural al permitir la ejecución concurrente de ambas ramas antes de la etapa de fusión.

La rama espectral define la longitud de la secuencia como:

$$N_{\text{spec}} = \left\lfloor \frac{B}{s} \right\rfloor, \tag{5}$$

donde s es el tamaño del segmento.

Las representaciones generadas por ambas ramas se definen como:

$$\mathbf{f}_{\text{spa}}, \mathbf{f}_{\text{spec}} \in \mathbb{R}^{\mathrm{d}}, \tag{6}$$

y se combinan mediante la operación de fusión:

$$\mathbf{f} = \text{concat}(\mathbf{f}_{\text{spa}}, \mathbf{f}_{\text{spec}}), \tag{7}$$

dando lugar a un vector de dimensión $2d$.

La predicción final se obtiene según:

$$\hat{\mathbf{y}} = \text{softmax}(W_o\ \text{LayerNorm}(\mathbf{f})), \tag{8}$$

donde $\hat{\mathbf{y}} \in \mathbb{R}^{\mathrm{C}}$ y $W_o \in \mathbb{R}^{\mathrm{C}\times 2\mathrm{d}}$. El pipeline completo implementado por la arquitectura de la Fig. 1 queda así formalizado mediante (6)–(8).

Tabla II: Resumen de arquitecturas evaluadas.

Familia	Modelo	Drop.	Seeds	Optim.
Clásicos	RF / MLR / SVM	·	1	·
CNN	CNN-2D / CNN-3D	0.3 / 0.4	5 / 3	AdamW
Transf.	ViT / SSFIT	·	3	AdamW+Muon
SSM	Mamba	·	5	AdamW
Prop.	`Dual-Branch Transformer`	·	3	AdamW+Muon

La concatenación preserva la separación estructural entre modalidades, manteniendo además la independencia computacional de ambas ramas y favoreciendo un mayor paralelismo efectivo en la arquitectura propuesta.

IV. Resultados y Discusión

Se reporta la OA bajo ruido analógico $\sigma = 0{,}10$, correspondiente a un nivel intermedio dentro del rango de perturbación evaluado. Este valor permite comparar las arquitecturas en un régimen no trivial, evitando tanto el caso ideal ($\sigma \to 0$) como escenarios de degradación extrema. Los valores reportados en la Tabla III corresponden a $\sigma = 0{,}10$, mientras que la Fig. 2 muestra la evolución completa en todo el rango de ruido considerado. La significancia estadística de las diferencias entre `Dual-Branch Transformer` y ViT se evalúa mediante un test t de Welch bilateral sobre las distribuciones de OA obtenidas a partir de distintas semillas.

La Tabla III recoge la OA media $\pm$ desviación estándar para todos los modelos. `Dual-Branch Transformer` obtiene la mayor OA en los cinco conjuntos de datos, con una ventaja de entre 0,11 y 1,48 puntos porcentuales sobre ViT, siendo estadísticamente significativa en PU, HS, HH y HC ($p < 0{,}05$). La desviación estándar del modelo propuesto se mantiene por debajo de 1 pp en cuatro de los cinco conjuntos, lo que indica una elevada estabilidad frente a la inicialización y al ruido.

Esta baja varianza sugiere que el patrón de ejecución de la arquitectura limita la amplificación del error inducido por el hardware analógico. La ausencia de dependencias secuenciales profundas restringe la propagación acumulativa del error a lo largo del grafo computacional.

La Fig. 2 muestra la evolución de la OA media en función de σ. `Dual-Branch Transformer` y ViT presentan una degradación progresiva y controlada, manteniendo valores elevados incluso en niveles altos de ruido. Este comportamiento es consistente con arquitecturas que operan sobre todos los tokens de forma simultánea, evitando dependencias secuenciales largas.

En contraste, CNN-3D y Mamba muestran caídas de rendimiento significativamente más acusadas, asociadas a patrones de ejecución con dependencias secuenciales o acoplamientos locales fuertes, donde el error se acumula de forma progresiva a lo largo de la computación.

Los resultados sugieren que el grado de paralelismo efectivo de una arquitectura contribuye al control de la propagación del error en hardware analógico. Las arquitecturas altamente paralelas tienden a distribuir las perturbaciones, mientras que las secuenciales favorecen su acumulación.

Las CNN, aunque paralelizables a nivel local, muestran una variabilidad elevada, lo que indica que el paralelismo espacial no resulta suficiente para garantizar robustez frente a perturbaciones globales en los pesos.

La mejora sistemática de `Dual-Branch Transformer` respecto a ViT se explica por la combinación de paralelismo global y paralelismo estructural entre ramas independientes, que reduce la profundidad efectiva de dependencias y limita la propagación del error.

En conjunto, los resultados muestran que el patrón de paralelismo determina la estabilidad frente a ruido analógico, situando a `Dual-Branch Transformer` como la arquitectura más robusta entre las evaluadas.

V. Conclusiones

En este trabajo se ha presentado una comparativa sistemática de nueve arquitecturas para la clasificación de imágenes hiperespectrales bajo simulación de hardware analógico realista mediante `aihwkit`, evaluada sobre cinco conjuntos de datos estándar y seis niveles de ruido $\sigma \in [0{,}01, 0{,}60]$.

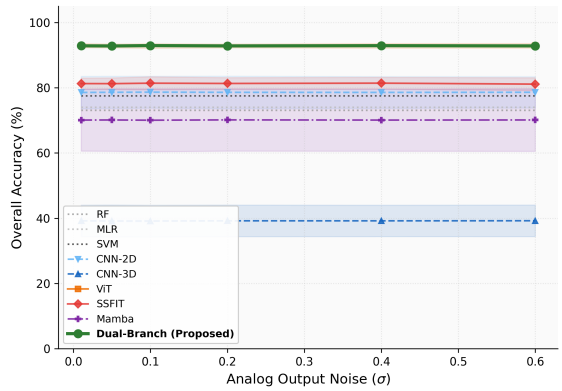
`Dual-Branch Transformer` alcanza la mayor OA en los cinco conjuntos de datos evaluados y muestra una elevada estabilidad entre semillas, con una desviación estándar inferior a 1 pp en cuatro de los cinco casos y mejoras estadísticamente significativas frente a ViT ($p < 0{,}05$) en cuatro escenarios.

Los modelos basados en atención presentan, en conjunto, una mayor estabilidad frente al ruido analógico que las redes convolucionales, mientras que los modelos SSM muestran una elevada variabilidad asociada a la propagación acumulativa del error en estructuras secuenciales.

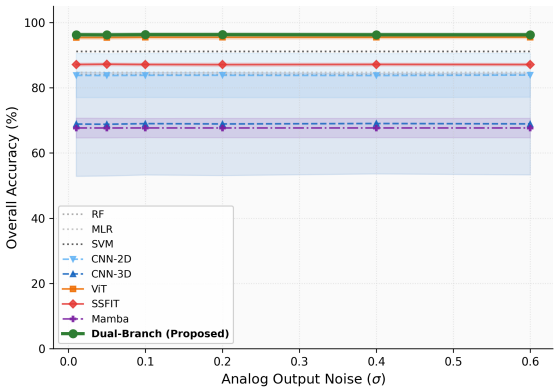
El comportamiento observado está estrechamente relacionado con el patrón de ejecución de cada arquitectura. Aquellas con alto grado de paralelismo efectivo tienden a limitar la propagación del error en comparación con arquitecturas con dependencias secuenciales profundas, mientras que los modelos con dinámica secuencial favorecen la acumulación de perturbaciones a lo largo de la computación.

`Dual-Branch Transformer` combina paralelismo global en la atención con paralelismo estructural entre ramas independientes, reduciendo la profundidad efectiva de dependencias y mejorando la estabilidad bajo ejecución en entornos de computación en memoria.

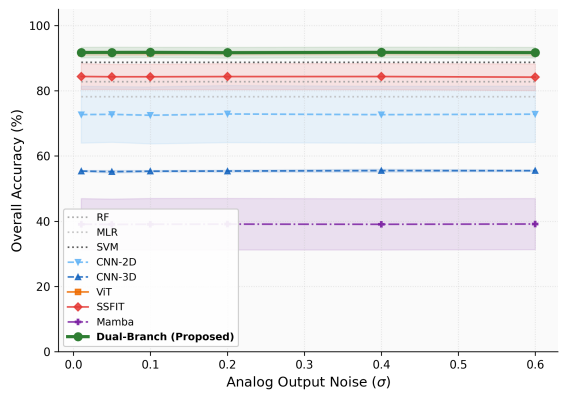
En conjunto, los resultados ponen de manifiesto que el paralelismo no solo actúa como un factor



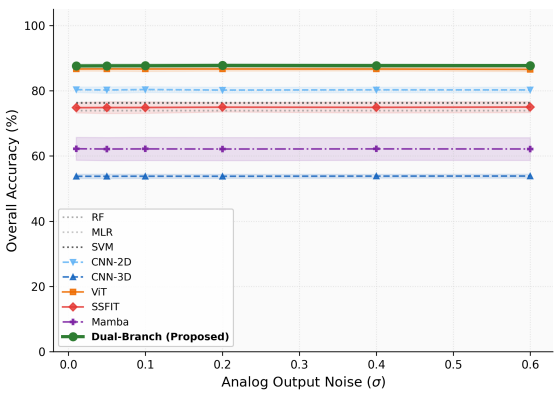
(a) IP



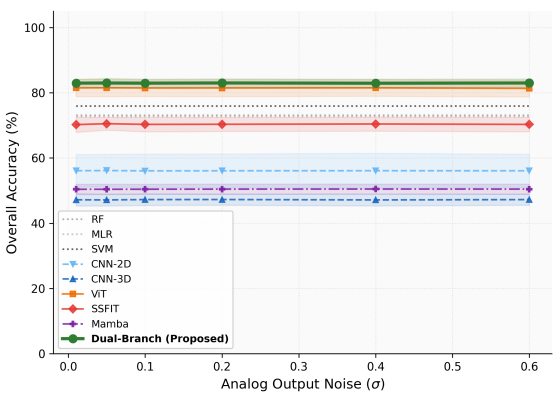
(b) PU



(c) HS



(d) HH



(e) HC

Fig. 2: OA (%) en función del nivel de ruido analógico σ para los cinco conjuntos de datos evaluados. Las bandas sombreadas indican ± 1 desviación estándar entre semillas. `Dual-Branch Transformer` mantiene sistemáticamente la mayor OA a lo largo de todo el rango de σ.

Tabla III: OA (%) bajo ruido analógico $\sigma = 0{,}10$. Formato: media $\pm$ desviación estándar sobre N semillas.

Familia	**Método**	**IP**	**PU**	**HS**	**HH**	**HC**
*Clásicos**	RF	73.10	84.03	82.76	73.91	73.04
	MLR	73.97	84.71	78.14	76.39	72.69
	SVM	77.46	91.11	88.72	76.26	75.87
CNN	CNN-2D	$78{,}64 \pm 4{,}89$	$83{,}81 \pm 6{,}75$	$72{,}48 \pm 8{,}75$	$80{,}37 \pm 0{,}89$	$56{,}05 \pm 5{,}17$
	CNN-3D	$39{,}14 \pm 4{,}85$	$69{,}00 \pm 15{,}72$	$55{,}33 \pm 0{,}26$	$53{,}79 \pm 0{,}64$	$47{,}25 \pm 1{,}80$
Transformer	ViT	$92{,}81 \pm 0{,}71$	$95{,}47 \pm 0{,}50$	$91{,}48 \pm 0{,}55$	$86{,}66 \pm 0{,}72$	$81{,}45 \pm 2{,}56$
	SSFIT	$81{,}39 \pm 1{,}99$	$87{,}12 \pm 0{,}45$	$84{,}28 \pm 3{,}91$	$74{,}86 \pm 1{,}77$	$70{,}30 \pm 2{,}26$
SSM	Mamba	$70{,}04 \pm 9{,}63$	$67{,}67 \pm 2{,}93$	$39{,}07 \pm 7{,}92$	$62{,}17 \pm 3{,}53$	$50{,}41 \pm 1{,}53$
Propuesto	`Dual-Branch Transformer`	$\mathbf{92{,}92 \pm 0{,}13}$	$\mathbf{96{,}29 \pm 0{,}65}^{\dagger}$	$\mathbf{91{,}77 \pm 1{,}58}^{\dagger}$	$\mathbf{87{,}67 \pm 0{,}63}^{\dagger}$	$\mathbf{82{,}93 \pm 1{,}27}^{\dagger}$

* Métodos deterministas, sin evaluación multi-semilla ni simulación analógica en el mismo marco experimental.
† Mejora estadísticamente significativa frente a ViT ($p < 0{,}05$, test t de Welch bilateral).

de eficiencia computacional, sino también como un mecanismo determinante en el control de la propagación del error en hardware analógico, situando a `Dual-Branch Transformer` como una arquitectura adecuada para su despliegue en sistemas de computación en memoria con restricciones energéticas.

Agradecimientos

Los autores agradecen la financiación proporcionada por la Consejería de Economía, Ciencia y Agenda Digital de la Junta de Extremadura y por el Fondo Europeo de Desarrollo Regional (FEDER), en el marco de las ayudas a grupos de investigación de la Junta de Extremadura, expediente GR24035. Asimismo, este trabajo ha sido financiado parcialmente por el proyecto de I+D PID2023-149880NA-I00, financiado por MICIU/AEI/10.13039/501100011033; por la Junta de Extremadura, referencia IB24090; y por una Beca Leonardo 2024 para Investigación Científica y Creación Cultural de la Fundación BBVA. La Fundación BBVA no se hace responsable de las opiniones, declaraciones o contenidos incluidos en el proyecto, ni de los resultados derivados del mismo, que son responsabilidad exclusiva de los autores. Este trabajo también ha recibido financiación del Fondo Europeo de Desarrollo Regional (FEDER) de la Unión Europea, a través del programa Interreg España-Portugal (POCTEP) 2021-2027, bajo la ayuda 0206_RAT_EOS_PC_6_E.

Referencias

[1] Zhaokui Li et al., "Deep learning for hyperspectral image classification: A survey," *Computer Science Review*, vol. 53, 2024.

[2] Muhammad Ahmad et al., "A comprehensive survey for hyperspectral image classification: The evolution from conventional to transformers and mamba models," *Neurocomputing*, 2025.

[3] William S Noble, "What is a support vector machine?," *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

[4] Steven J Rigatti, "Random forest," *Journal of insurance medicine*, vol. 47, no. 1, pp. 31–39, 2017.

[5] Yushi Chen, Hanlu Jiang, Chunyang Li, Xiuping Jia, and Pedram Ghamisi, "Deep feature extraction and classification of hyperspectral images based on convolutional neural networks," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 10, pp. 6232–6251, 2016.

[6] Ying Li, Haokui Zhang, and Qiang Shen, "Spectral-spatial classification of hyperspectral imagery with 3d convolutional neural network," *Remote Sensing*, vol. 9, no. 1, pp. 67, 2017.

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations (ICLR)*, 2021.

[8] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al., "A survey on vision transformer," *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 1, pp. 87–110, 2022.

[9] Albert Gu and Tri Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[10] Guanchun Wang et al., "S^2Mamba: A spatial-spectral state space model for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, 2025.

[11] Fernando Aguirre et al., "Hardware implementation of memristor-based artificial neural networks," *Nature Communications*, vol. 15, 2024.

[12] Malte J. Rasch et al., "Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators," *Nature Communications*, vol. 14, pp. 5282, 2023.

[13] Vinay Joshi, Manuel Le Gallo, Simon Haefeli, Irem Boybat, S. R. Nandakumar, Christophe Piveteau, Martino Dazzi, Bipin Rajendran, Abu Sebastian, and Evangelos Eleftheriou, "Accurate deep neural network inference using computational phase-change memory," *Nature Communications*, vol. 11, pp. 2473, 2020.

[14] Malte J. Rasch, Diego Moreda, Tayfun Gokmen, Manuel Le Gallo, Fabio Carta, Cindy Goldberg, Kaoutar El Maghraoui, Abu Sebastian, and Vijay Narayanan, "A flexible and fast PyTorch toolkit for simulating training and inference on analog crossbar arrays," in *IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2021.

[15] W. Zhao et al., "SSFIT: Spectral-spatial feature integration transformer for hyperspectral image classification," *IEEE Geoscience and Remote Sensing Letters*, 2023.

[16] Manuel Le Gallo et al., "Using the IBM analog in-memory hardware acceleration kit for neural network training and inference," *APL Machine Learning*, vol. 1, pp. 041102, 2023.

Memoria Reconfigurable para Resultados Parciales en Aceleradores de DNN Eficientes

Adrián Navarro[1], José Cano[2], José L. Abellán[1] y Manuel E. Acacio[1]

Resumen— **En los últimos años se han propuesto un gran número de arquitecturas aceleradoras para Redes Neuronales Profundas (DNNs) en el borde (*edge*), centradas principalmente en optimizar la multiplicación de matrices dispersas (*sparse*). La mayoría de estas se centran en la implementación de su flujo de datos (*dataflow*), es decir, en el orden en que las unidades computacionales realizan las operaciones de multiplicación y acumulación, ignorando el impacto de las estructuras de memoria. Para *dataflows* basados en el algoritmo de Gustavson, que han demostrado ser los más eficientes en cargas de trabajo con *sparsity* medio (10-99 %) –presentes en DNNs *sparse*–, hemos observado que la organización de la estructura de memoria encargada de almacenar los resultados parciales puede tener un impacto significativo en el rendimiento. Diferentes modelos de DNN, incluso capas dentro de un mismo modelo, imponen requisitos variables sobre dicha estructura. Los diseños propuestos hasta ahora, considerados rígidos, provocan una degradación de rendimiento debido a su falta de flexibilidad.**

En este trabajo defendemos la necesidad de dotar de adaptabilidad a la memoria encargada de almacenar resultados parciales y proponemos QuFi, una estructura de memoria para resultados parciales reconfigurable diseñada para integrarse fácilmente en cualquier acelerador basado en Gustavson y adaptada a dispositivos para el *edge*. QuFi se implementa como una colección de colas a través de múltiples bancos de memoria, soportando reconfigurabilidad y alto ancho de banda mediante dos mecanismos clave: la fusión de colas y la agrupación dinámica. A través de una evaluación detallada que considera su integración en un acelerador del estado del arte para el *edge*, demostramos que QuFi obtiene una mejora de rendimiento media de 1.64×, una reducción de hasta el 73 % en el tráfico de memoria *off-chip* y un ahorro del 17 % en el área total del acelerador.

Palabras clave— **Acelerador Hardware, Redes Neuronales Profundas, Multiplicación de Matrices Dispersas, Dispositivo Edge, Organización de Memoria, Reducción de Sumas Parciales, Adaptabilidad, Rendimiento, Área.**

I. Introducción y Motivación

El rápido crecimiento en los últimos años del número de aplicaciones basadas en Redes Neuronales Profundas (DNNs) ha impulsado los esfuerzos para desplegarlas en dispositivos en el borde (*edge*), que cuentan con estrictas restricciones de área y consumo energético [1]. Dado que la mayoría de los modelos de DNN son demasiado grandes para estas plataformas, en su lugar se utilizan versiones dispersas (*sparse*), más pequeñas, optimizadas y especializadas para el *edge* de los mismos [2–4].

El núcleo computacional principal en estos modelos es la multiplicación de matriz *sparse* por matriz *sparse* (SpMSpM). En los años recientes se han propuesto varios diseños de aceleradores SpMSpM destinados a cumplir con los requisitos de baja latencia y alta eficiencia energética que el procesamiento de estos modelos impone al hardware [5–15]. En general, el diseño de estos aceleradores se organiza en torno al soporte de uno o varios flujos de datos (*dataflows*), los cuales determinan cómo se procesan y fluyen los datos dentro del sustrato de cómputo del acelerador. De todos los *dataflows* utilizados hasta ahora, aquellos basados en el algoritmo de Gustavson han demostrado ser los más eficientes para los escenarios de *sparsity* medio (10 % – 99 %) observados en las cargas de trabajo de DNN *sparse* [10–12].

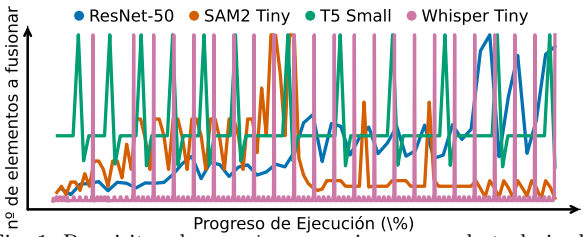


Fig. 1: Requisitos de *merging* en varias cargas de trabajo de DNN modernas para *edge* a lo largo de su ejecución.

El cómputo de una operación SpMSpM en un acelerador basado en Gustavson consta de dos fases principales: multiplicación y reducción (*merging*). En la fase de multiplicación, se calculan los productos de valor por fila para generar filas de resultados parciales, almacenadas en una memoria de resultados intermedia. Luego, en la fase de *merging*, estos resultados parciales se recuperan de dicha memoria y se reducen para obtener la fila de resultados final. Aunque en la mayoría de los casos la fase de multiplicación domina el tiempo de procesamiento, las dimensiones variables en los modelos modernos de DNN para el *edge* y los cambiantes requisitos de *merging* entre capas dentro de un mismo modelo (véase la Fig. 1) provocan que la fase de *merging*, si no se gestiona adecuadamente, represente una fracción sustancial del tiempo de ejecución (hasta un 49 %). En este trabajo, sostenemos que el soporte eficiente para la fase de *merging* requiere de una estructura de memoria especializada y adaptativa que pueda satisfacer las diversas demandas que plantean los diferentes modelos de DNN (y las capas dentro de cada modelo).

Mientras que los trabajos previos se han centrado principalmente en el diseño de la arquitectura del *dataflow* [5,11,14], la organización de la jerarquía de memoria encargada de proporcionar datos a los elementos de cómputo del acelerador ha recibido menos atención. Algunos diseños de aceleradores recientes optan por una memoria dedicada y rígida para almacenar los resultados parciales [8,10,12], la cual carece

[1]Dpto. de Ingeniería y Tecnología de Computadores, Universidad de Murcia, e-mail: {adrian.fenollarn,jlabellan,meacacio}@um.es.
[2]School of Computing Science, University of Glasgow, e-mail: Jose.CanoReyes@glasgow.ac.uk.

de la adaptabilidad necesaria para manejar eficientemente las diversas SpMSpM existentes entre las capas de cada DNN. Otras propuestas del estado del arte combinan los datos de entrada y los resultados parciales en una única estructura de memoria grande basada en caché [6,9,13,15], que puede proporcionar cierto nivel de adaptación para la fase de *merging*, ya que la cantidad de memoria asignada para almacenar los resultados parciales puede variar dinámicamente según las necesidades observadas en un momento determinado. Sin embargo, esto impacta negativamente en la eficiencia energética, resultando inadecuado para dispositivos *edge* debido a sus altas restricciones de área y consumo energético.

En este trabajo proponemos QuFi, una estructura de memoria para *merging* especializada, ligera y adaptativa, diseñada específicamente para una fácil integración en cualquier acelerador basado en Gustavson. Diseñada para ser implementada en dispositivos para el *edge*, QuFi se estructura como una colección de colas y admite reconfigurabilidad mediante técnicas de fusión de colas y agrupación dinámica. También proporciona un alto ancho de banda al paralelizar los accesos a través de los bancos de lectura de cada cola, de manera similar a la organización del banco de registros de una GPU.

A través de una evaluación exhaustiva utilizando cuatro modelos modernos de DNN *sparse* para el *edge* con diferentes tamaños y requisitos de memoria, demostramos que QuFi, en comparación con la Psram [10] (una memoria de resultados parciales rígida) puede proporcionar mejoras significativas de rendimiento: una media de 1,64× más rápido, alcanzando en promedio el 93 % del rendimiento de *merging* máximo del acelerador del estado del arte Flexagon [10], y una reducción sustancial en el tráfico de memoria *off-chip* de hasta un 73 %, requiriendo 4× menos almacenamiento efectivo, significando un ahorro total del 17 % en el área del acelerador.

Las principales contribuciones de este trabajo son las siguientes:

- Análisis de los principales cuellos de botella en la fase de *merging* de los *dataflows* basados en Gustavson, que afectan a los diseños actuales de memoria para *merging*.
- Propuesta de QuFi, una nueva arquitectura de memoria para *merging* diseñada para mitigar estos cuellos de botella.
- Integración de QuFi en un acelerador del estado del arte, seguida de una evaluación exhaustiva que demuestra sus ventajas frente al diseño rígido original, la Psram.

II. Antecedentes y Trabajo Relacionado

A. Dataflows para SpMSpM

Una multiplicación de matrices computa $\mathcal{C}_{MN} = \mathcal{A}_{MK} \times \mathcal{B}_{KN}$. Esta operación se realiza como un bucle triplemente anidado, donde cada uno itera sobre una de las tres dimensiones (M, N y K). Basándonos en el orden de estos bucles, podemos definir tres *dataflows* principales implementados por diferentes aceleradores de SpMSpM: Inner-Product (IP) [5, 10], Outer-Product (OP) [6, 7, 10, 15] y Gustavson (Gust) [8–10]. Otra opción, Gustavson por bloques (*Tiled Gustavson*, T-Gust), también ha sido explotada en trabajos recientes [11, 12, 16], demostrando ser el enfoque más prometedor para escenarios de *sparsity* medio, es decir, el dominio de DNNs *sparse* considerado en este trabajo [12].

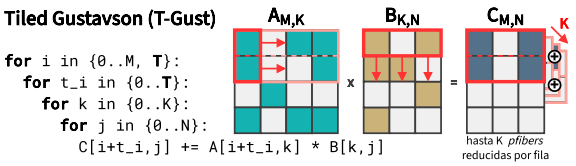


Fig. 2: *Dataflow* para SpMSpM: Gustavson por bloques.

T-Gust expone distintos patrones de reutilización de datos para cada matriz involucrada en un SpMSpM, siendo su única diferencia respecto a Gust que define un tamaño de bloque T que determina la granularidad de iteración sobre la matriz $\mathcal{A}$. Como se muestra en la Fig. 2, la matriz $\mathcal{A}$ necesita leerse solo una vez, con T fibras[1] de $\mathcal{A}$ leídas en paralelo. Por cada valor a_{ik} leído, se debe obtener la k-ésima fibra de $\mathcal{B}$ y computar el producto valor por vector. Esta operación produce una fibra parcial de resultados a la que denominamos *pfiber*, compuesta de sumas parciales (*psums*) de $\mathcal{C}$, y que debe almacenarse temporalmente en una memoria intermedia para su posterior reducción. Este proceso implica que la matriz $\mathcal{B}$ puede leerse hasta $\frac{M}{T}$ veces, ofreciendo oportunidades para la reutilización de datos. Del mismo modo, la reutilización en la matriz $\mathcal{C}$ surge de la necesidad de reducir las *pfibers*. La ventaja clave desde la perspectiva del *dataflow* de T-Gust reside en su adaptabilidad a través de T [11], siendo capaz de generalizar Gust ($T = 1$) y OP ($T = M$) [10]. Sin embargo, aumentar T introduce una relación entre rendimiento y consumo de memoria: mientras que un T mayor mejora la localidad de los accesos a la matriz $\mathcal{B}$, también incrementa el número de *pfibers* que deben almacenarse simultáneamente en la memoria de resultados parciales intermedia.

B. Computación en T-Gust

La computación de T-Gust puede dividirse conceptualmente en dos fases principales: multiplicación y reducción (*merging*). La fase de multiplicación, que domina el tiempo de ejecución total, implica la obtención de valores de $\mathcal{A}$ y $\mathcal{B}$, el cálculo de productos y la generación de *pfibers*. Concretamente, para cada fila i de $\mathcal{A}$, se producen tantas *pfibers* como el número de valores distintos de cero haya en dicha fila ($nnz_i^{\mathcal{A}}$), almacenándose en una memoria para *merging* intermedia. Después, en la fase de *merging* estas *pfibers*, correspondientes a una misma fibra de resultados de $\mathcal{C}$, deben reducirse en una única fibra de resultados finales.

[1]Una fibra (*fiber*) hace referencia a una fila o columna comprimida que contiene una lista de pares de valor no nulo y coordenada, ordenados por coordenada.

Diferentes unidades de *merging*, como el *multi-queue merger* de MatRaptor [8], el *high-radix tree* de Gamma [9] y la *merger-reduction network* (MRN) de Flexagon [10], se han utilizado en aceleradores previos. Todos ellos tienen un límite en el número de *pfibers* que pueden reducirse en un solo paso, dictado por el ancho de su estructura de reducción, denotado como R_w. El *dataflow* T-Gust permite una optimización en hardware: las *pfibers* que se calculan simultáneamente por diferentes Elementos de Procesamiento (PEs) pueden reducirse inmediatamente antes de enviarlas a memoria, reduciendo el número de *pfibers* almacenadas por un factor de $R_w\times$ [9–12]. Por lo tanto, si $nnz_i^{\mathcal{A}} \leq R_w$, no se requiere fase de *merging*, ya que se producirá una fibra final en cada iteración de la fase de multiplicación, eliminando la necesidad de almacenarlas temporalmente y, por ende, reduciendo la presión sobre la memoria intermedia. Sin embargo, cuando $nnz_i^{\mathcal{A}} > R_w$, se realizan varios pasos de *merging* parciales de forma iterativa, requiriendo aproximadamente $\lceil \frac{nnz_i^{\mathcal{A}}}{R_w - 1} + O(\log_{R_w} nnz_i^{\mathcal{A}}) \rceil$ pasos totales (asumiendo $T = 1$). Como resultado, la memoria intermedia responsable de mantener las *psums* necesarias para calcular los valores finales de la matriz $\mathcal{C}$ debe ser tanto eficiente como flexible, admitiendo longitudes de *pfiber* variables, un número variable de *pfibers* concurrentes y acceso simultáneo de lectura/escritura. Sin tal adaptabilidad, la fase de *merging* puede convertirse en un cuello de botella de rendimiento, aumentando significativamente el tiempo total de ejecución.

C. Diseños de memoria en aceleradores SpMSpM

Las propuestas previas basadas en Gust exploran principalmente el espacio de diseño del *dataflow* (es decir, cómo se computan y reducen las *pfibers*), mientras que ponen menos énfasis en la organización de la jerarquía de memoria del acelerador. La mayoría de las propuestas actuales consideran una jerarquía de memoria de dos niveles. El primer nivel proporciona acceso rápido a una pequeña porción de los datos en la matriz $\mathcal{A}$ (que se mantiene estacionaria en los PEs y se accede a través de una estructura FIFO), las fibras de $\mathcal{B}$ y las *pfibers* necesarias para generar la matriz $\mathcal{C}$, que requieren estructuras de memoria más complejas. El segundo nivel, aunque más lento, puede albergar todos los datos necesarios. Dependiendo de cómo se organice la memoria de primer nivel para $\mathcal{B}$ y $\mathcal{C}$, podemos distinguir dos tipos de jerarquías de memoria: basadas en caché y especializadas.

C.1 Memorias basadas en caché

La mayoría de los diseños actuales pertenecen a esta categoría [6,9,11,13–15]. En este caso, se emplea una única y gran estructura de memoria basada en una caché para el primer nivel, donde se almacenan tanto las fibras de $\mathcal{B}$ como las *pfibers* para computar $\mathcal{C}$. Esto simplifica la lógica de control del *dataflow* debido a las transferencias de memoria *on-chip* y *off-chip* gestionadas implícitamente por el hardware y evita realizar un particionado de la operación.

No obstante, tales enfoques suelen ser inadecuados en aceleradores para el *edge* por varias razones. Una de ellas es que ambos tipos de fibras, que exhiben patrones de reuso diferentes, están compitiendo por los mismos recursos de almacenamiento, lo que conduce a escenarios donde no se hace un uso óptimo de la estructura de memoria unificada. Por ejemplo, Gamma [9], un acelerador basado en Gust, introdujo la FiberCache, un diseño de memoria monolítica de gran tamaño (también adoptado en otros trabajos [11,14]), con una capacidad de 3MiB que emplea políticas de reemplazo separadas para las fibras de $\mathcal{B}$ y las parciales de $\mathcal{C}$.

Además, los diseños basados en caché requieren de área adicional para almacenar los metadatos (*tags*), lo cual es necesario para las fibras de $\mathcal{B}$ pero puede omitirse para las *pfibers* de $\mathcal{C}$; así como para solventar desafíos adicionales como la necesidad de una lógica de control sofisticada, las decisiones de reemplazo y la gestión de la distribución de datos a través de múltiples bancos de memoria para mantener un alto ancho de banda. Esta complejidad dificulta el escalado de bancos, aumenta significativamente la latencia y compromete el paralelismo de lecturas y escrituras durante la fase de *merging*. Como resultado, estos problemas provocan un aumento de los costes de área y consumo energético, lo que convierte a las memorias basadas en caché en diseños menos deseables para los aceleradores para el *edge*.

C.2 Memorias especializadas

Otros aceleradores han explorado estructuras de memoria alternativas, especializadas y no unificadas para el primer nivel, siendo así más adecuadas para el *edge*. MatRaptor [8], un acelerador también basado en Gust, implementa colas dentro de cada PE donde se almacena la *pfiber* computada para posteriormente combinarla. Sin embargo, este diseño limita su adaptabilidad pues cada cola se emplea independientemente en cada PE y requiere, además, de *double buffering* para poder permitir operaciones concurrentes de lectura y escritura sobre las colas, exponiendo por tanto una memoria rígida que demanda área adicional.

Flexagon [10], un acelerador multi-*dataflow* diseñado para dispositivos en el *edge*, utiliza una memoria basada en caché para $\mathcal{B}$ e introduce una nueva estructura de memoria llamada Psram, diseñada para almacenar *pfibers* tanto en OP como en Gust. Está construida sobre un *scratchpad* multibanco y está organizada en conjuntos correspondientes a fibras de resultados individuales. Cada fila consta de múltiples líneas, donde cada línea almacena, idealmente, una única *pfiber*. Cuando el tamaño de la línea es insuficiente para almacenar una *pfiber* completa, se emplea un método de combinación de líneas, permitiendo que una *pfiber* abarque múltiples líneas no contiguas. La línea de *pfibers* dentro de un conjunto se determina mediante una búsqueda indexada por la etiqueta de iteración k. Una propuesta similar fue utilizada posteriormente en Trapezoid [12].

A pesar de su versatilidad, PSRAM sigue presentando un diseño rígido y subóptimo. Aunque logra una alta ocupación para OP, está infrautilizada cuando se utiliza T-Gust ya que solo se procesan T filas a la vez, restringiendo así el uso a solo T conjuntos de memoria. Además, aunque admite la ejecución de múltiples *dataflows*, la PSRAM sacrifica capacidad de memoria y no se alinea con las necesidades específicas de T-Gust debido a los diferentes requerimientos de *merging*: OP requiere almacenar muchas matrices parciales con alto *sparsity*, mientras que T-Gust maneja T filas de *pfibers* con menor *sparsity*. Esta limitación en el almacenamiento obliga a escribir parte de las *pfibers* en la memoria principal, incrementándose así el tráfico de memoria global y reduciendo la utilización tanto de los PEs como de la unidad de *merging*. Adicionalmente, la distribución de *pfibers* a través de múltiples bancos de lectura introduce complejidad de control adicional y se requiere de *double buffering* para permitir lecturas y escrituras simultáneas durante la fase de *merging*, lo que incurre en un coste de área adicional.

Por ello, partiendo de una arquitectura de acelerador SpMSpM para el *edge* con estructuras de memoria separadas para el primer nivel, en este trabajo presentamos un diseño reconfigurable para la estructura de memoria utilizada para almacenar las *pfibers* de la matriz $\mathcal{C}$, capaz de adaptarse eficientemente a las necesidades variables de la fase de *merging* que cada carga de trabajo demanda.

III. Diseño de QuFi

La ejecución de la fase de *merging* en T-Gust exhibe características variables, no solo entre diferentes modelos de DNN y entre diferentes capas dentro de un mismo modelo, como se muestra en la Fig. 1, sino también dentro de cada capa. Para poder realizar operaciones de *merging* eficientes, identificamos dos requisitos clave para la estructura de memoria utilizada para mantener las *pfibers* a reducir:

1. Adaptabilidad mediante reconfigurabilidad: Al igual que los*dataflows* [10, 11], la estructura de memoria que proporciona las *pfibers* durante la fase de *merging* debe ser capaz de adaptarse a demandas variables. Los diseños rígidos dan como resultado una infrautilización de las unidades de cómputo (PEs), de almacenamiento (capacidad de memoria) y/o del ancho de banda (utilización de los bancos de memoria).
2. Alto ancho de banda en chip: Se deben admitir múltiples lecturas paralelas de cada *pfiber*. Además, las escrituras simultáneas también deben gestionarse de manera eficiente.

En este trabajo presentamos QuFi, un diseño de memoria para almacenar resultados parciales en un acelerador T-Gust que cubre estos importantes requisitos y garantiza operaciones de *merging* de *pfibers* más eficientes que las propuestas actuales.

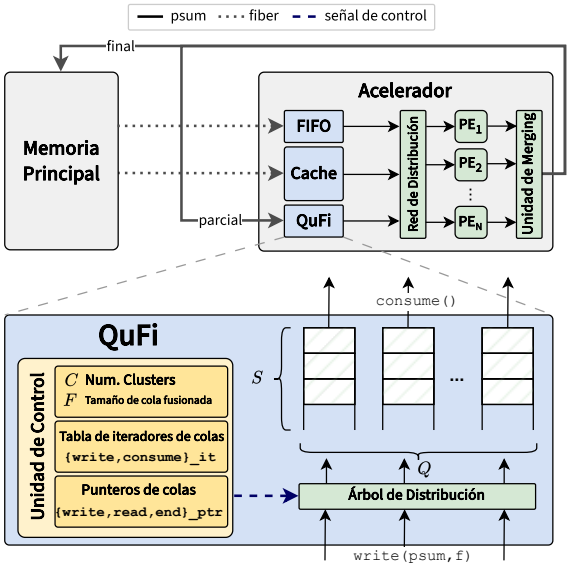


Fig. 3: QuFi integrada en un acelerador T-Gust genérico.

A. Descripción general de QuFi

La Fig. 3 muestra un diagrama de alto nivel de QuFi, sus componentes y su integración dentro de un acelerador flexible SpMSpM genérico T-Gust [17] (que implementa una red de distribución, un *array* de PEs, una unidad de *merging* y memorias discretas no unificadas) adecuado para dispositivos *edge*. QuFi está estructurado como una colección de Q colas, cada una con una capacidad fija de S *psums*, que almacena las *pfibers* a reducir, así como un árbol de distribución para enviar las escrituras entrantes de *pfibers* a las colas. La red de distribución del acelerador se reutiliza para enviar los *psums* de cada *pfiber* dentro de QuFi a la unidad de *merging* e iniciar una operación de *merging*.

QuFi proporciona reconfigurabilidad a través de la fusión de colas y el agrupamiento dinámico. La fusión de colas proporciona un mecanismo para almacenar *pfibers* de tamaño mayor que S *psums* mediante la combinación lógica de F colas consecutivas para cada *pfiber*, permitiendo almacenar *pfibers* de hasta $S \times F$ *psums*. El agrupamiento dinámico divide lógicamente el almacenamiento de QuFi en C clústeres de colas más pequeños e independientes. Cada clúster se comporta como una QuFi independiente, donde cada uno se dedica a almacenar todas las *pfibers* dentro de la misma fila de resultados. El ajuste dinámico de C mejora la utilización de recursos, especialmente cuando el número de *pfibers* a reducir por fila es pequeño.

Dado que QuFi está compuesto por múltiples colas ligeras e independientes, las lecturas y escrituras pueden ocurrir de forma simultánea y en paralelo a través de ellas. Esto permite un alto ancho de banda en chip durante la fase de *merging* al proporcionar la primera *psum* de cada *pfiber* en cada ciclo. Su diseño se beneficia enormemente de una lógica simplificada al implementar estructuras ligeras como colas y árboles de distribución. Esto permite realizar reconfiguraciones dinámicamente con un bajo coste, lo que lo hace más eficiente que un diseño basado en caché en términos de área y energía [8, 10].

B. Arquitectura de QuFi

Considerando $Q, S, C \in \{2^k \mid k \in \mathbb{N}\}$, una QuFi tiene Q colas circulares de S *psums* de capacidad cada una y puede dividirse lógicamente en C clústeres. Cada cola, denotada como q_i, proporciona puertos de lectura y escritura separados que pueden usarse en paralelo, ya que se garantiza que las operaciones de lectura y escritura nunca se realicen simultáneamente en la misma entrada de q_i (lo cual es un requisito en los pasos de *merging* parciales). Para gestionar esto, la unidad de control (ver Fig. 3) mantiene tres punteros para cada q_i: `write_ptr`, que apunta a la siguiente entrada disponible para ser escrita; `read_ptr`, que apunta a la siguiente entrada para ser leída; y `end_ptr`, que apunta a la última entrada disponible para leer. Además, mantiene dos tablas de iteradores, `consume_it` y `write_it`, una para las operaciones `consume()` y otra para `write()`, utilizadas únicamente cuando se aplica la fusión de colas ($F > 1$). La Sección III-F explica detalladamente el funcionamiento de la unidad de control de QuFi durante la fase de *merging*.

Para habilitar tanto la fusión de colas como el agrupamiento (detalladas más adelante) en un acelerador T-Gust flexible para SpMSpM determinado, QuFi reutiliza diseños de red en chip (NoC) especializados adoptados previamente en arquitecturas de aceleradores flexibles. El agrupamiento se admite de manera eficiente a través de topologías basadas en árboles —comúnmente utilizadas en trabajos previos [5, 8–12]— que pueden particionarse dinámicamente desactivando los niveles superiores del árbol basándose en la configuración actual de C (C árboles lógicos con una raíz independiente cada uno). Estas topologías son esenciales no solo para reducir *pfibers*, sino también para distribuir *psums* entre la unidad de *merging* del acelerador y sus colas correspondientes. La fusión de colas, a su vez, se habilita configurando dinámicamente la red de distribución del acelerador de acuerdo con los valores actuales de C, F y las tablas de iteradores de colas.

Como se muestra en la Fig. 3, QuFi emplea múltiples NoCs con funciones distintas. La unidad de *merging* del acelerador realiza la reducción de *pfibers*, generando C *psums* por ciclo. Estas *psums* se enrutan luego a través del árbol de distribución de QuFi, que utiliza enlaces con ancho de 1-*psum* para enviar datos a cada q_i mediante operaciones `write()`. Adicionalmente, la red de distribución del acelerador gestiona las operaciones `consume()`, enviando por *unicast* los datos leídos de las colas a través de los PEs (configurados en *modo forward*) y, consecuentemente, a los puertos de entrada de la unidad de *merging*.

C. Fusión de Colas

C.1 Predicción del tamaño de la *pfiber*

Cuando es necesario almacenar una *pfiber*, QuFi debe conocer su longitud para decidir si activar el mecanismo de fusión de colas o no. Trabajos previos [18, 19] han explotado las características de la carga de trabajo de las DNN *sparse* para predecir la memoria requerida en los búferes intermedios: estas muestran *sparsity* medio (menos del 99 %) y valores distintos de cero distribuidos de manera casi uniforme derivados de las funciones de poda y activación [2]. Bajo estos supuestos, el *sparsity* de la matriz de resultados puede aproximarse probabilísticamente como:

$$d_{\mathcal{C}} = 1 - (1 - d_{\mathcal{A}} d_{\mathcal{B}})^K$$

siendo d_x la densidad de la matriz x. Por ejemplo, con $d_{\mathcal{A}} = 10\,\%$, $d_{\mathcal{B}} = 30\,\%$, y $K = 128$, $d_{\mathcal{C}} \approx 98\,\%$, lo que significa que cada *pfiber* tiene una densidad media del 98 %. Nuestra evaluación experimental confirma este comportamiento (ver Tabla II), donde $d_{\mathcal{C}} \approx 100\,\%$ en todos los *benchmarks* SpMSpM. Cabe destacar que, en modelos de DNN *sparse*, $d_{\mathcal{A}}$ y $d_{\mathcal{B}}$ son generalmente mayores que en el ejemplo anterior, lo que demuestra que, incluso en escenarios más *sparse*, los resultados obtenidos son potencialmente densos. Así, y aprovechando que durante la fase de multiplicación ya se produce implícitamente la primera reducción de hasta R_w *pfibers* (i.e., K en la ecuación), se considera que cada *pfiber* ocupará tanto espacio como una *pfiber* totalmente densa[2] (N *psums*) sin provocar una baja ocupación de memoria, lo que, además, simplifica la lógica del controlador.

C.2 Adaptación de las colas

Idealmente, cada *pfiber* se almacena en su propia cola, lo que permite el *merging* simultáneo de hasta Q *pfibers*. Sin embargo, los diferentes tamaños de las capas de las DNN dificultan el diseño de una memoria *on-chip* rentable que se ajuste a todo el rango de tamaños de *pfiber* (N) sin aumentar el tamaño de la cola (S). Esto conduce a una baja utilización cuando $N < S$ o a una capacidad insuficiente cuando $N > S$. Para equilibrar el rendimiento del acelerador y la capacidad de la memoria *on-chip*, QuFi utiliza la fusión de colas. Esto aumenta el tamaño máximo de las *pfibers* que puede almacenar, pero reduce cuántas *pfibers* se pueden reducir simultáneamente en un solo paso de *merging*. Cuando está habilitado, cada *pfiber* ocupa F colas consecutivas (colas fusionadas), ampliando el tamaño máximo soportado para cada *pfiber* a $S \times F$ *psums*. Por lo tanto, F debe configurarse estáticamente para cada SpMSpM como $\lceil \frac{N}{S} \rceil$, limitando el número total de *pfibers* que QuFi puede almacenar a $\lfloor \frac{Q}{F} \rfloor$.

El ejemplo de la Fig. 4 muestra una QuFi con $Q = 8$ y $S = 8$ configurada con $F = 2$, siendo capaz de almacenar hasta 4 *pfibers* de hasta 16 elementos cada una. Dado que también se aplica el agrupamiento, con $C = 2$, cada clúster contiene solo 2 *pfibers*. F es el mismo para todos los clústeres porque, según nuestro estudio previo, se espera que todas las *pfibers* tengan el mismo tamaño efectivo (N). Nótese que F debe poder variar desde 1 (cada *pfiber* usa una sola cola) hasta $\frac{Q/C}{2}$ (cada *pfiber* usa la mitad de las colas de su clúster, ya que al menos dos *pfibers* deben

[2]El *sparsity* de la matriz de resultados surge después de que se compute una fibra final y se envía a la memoria principal, donde se aplica la función de activación de hardware.

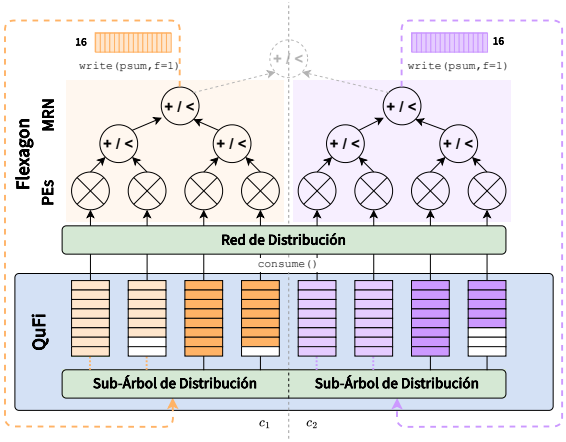


Fig. 4: Ejemplo de QuFi integrado en Flexagon [10]: $PEs = Q = 8$, $S = 8$, $C = 2$ y $F = 2$ para una SpMSpM con $N = 16$. Se utiliza una única MRN, por lo que Q debe coincidir con el número de PEs en este caso. Cada clúster puede albergar 2 *pfibers* y cada *pfiber* abarca 2 colas. Dado que el agrupamiento está habilitado, el nivel superior del árbol de *merging* se desactiva y los resultados se envían a la memoria directamente desde el nivel inferior. La MRN puede suministrar hasta C *psums* por ciclo, es decir, las siguientes *psums* de las *pfibers* actuales que se están reduciendo. Obsérvese que los PEs están configurados en *modo forward* durante el *merging*.

reducirse simultáneamente) para asegurar que todas las *pfibers* se almacenen en QuFi. Si F excede el valor máximo (es decir, si incluso aplicando la fusión de colas no hay capacidad suficiente para al menos dos *pfibers*), las *psums* restantes de cada *pfiber* deben almacenarse en la memoria principal y leerse de nuevo durante la siguiente fase de *merging*. Dado que F es un parámetro estático para cada SpMSpM en base a sus dimensiones, nuestro estudio preliminar ha demostrado que generalmente es más conveniente reducir el valor de C (dinámico, detallado a continuación) para garantizar que F esté dentro del rango deseado, minimizando el tráfico *off-chip* de QuFi.

D. Agrupamiento Dinámico

Tener pocas *pfibers* para reducir puede provocar la infrautilización del acelerador y de sus recursos de memoria durante la fase de *merging*, lo que implica un bajo rendimiento. Esto ocurre a menudo con matrices pequeñas donde también se aplica una *merging* previo mediante una unidad de *merging*, es decir, se almacenan $\lceil \frac{nnz_i^A}{R_w} \rceil$ *pfibers* en QuFi (siendo R_w el número de *pfibers* que se pueden reducir en paralelo en una sola operación de *merging*). Por ejemplo, considerando $R_w = 64$, $Q = 64$, y $nnz_0^A = 256$, solo se enviarán 4 *pfibers* a la memoria intermedia, obteniendo así una utilización máxima del 6.25 % tanto de los puertos de lectura como de la capacidad de almacenamiento (asumiendo una cola por *pfiber*). Con el agrupamiento, se pueden procesar múltiples filas de resultados simultáneamente, lo que permite mejorar potencialmente la ocupación de las colas, así como la utilización de los puertos de lectura/escritura.

QuFi opera de forma independiente entre clústeres, comportándose como memorias lógicas separadas. Como resultado, cada clúster consta de $\frac{Q}{C}$ colas. C se configura dinámicamente para cada bloque (*tile*) basándose en el número de *pfibers* esperadas dentro del bloque actual. Siguiendo el ejemplo de la Fig. 4, un QuFi de $Q = 8$ podría configurarse con $C = 2$, teniendo efectivamente 2 QuFis lógicas más pequeñas con $Q_i = 4$ cada una. En este ejemplo, cada clúster puede asignarse al *merging* de las diferentes *pfibers* de una fila de la matriz de resultados, dando así soporte a T-Gust. En este caso, C determina el tamaño del bloque (T). El valor óptimo de C depende de factores como el *dataflow* del acelerador, el número de elementos distintos de cero de cada fila (nnz_i^A) y la jerarquía de memoria para las matrices $\mathcal{A}$ y $\mathcal{B}$. Técnicas como el reordenamiento de filas [5, 9] pueden mejorar aún más la ocupación de la memoria. Sin embargo, la exploración de esta configuración está fuera del alcance de este trabajo.

E. Operaciones Soportadas e Interfaz

Para facilitar la integración en las infraestructuras de aceleradores existentes, QuFi expone una interfaz similar a la utilizada por Psram, definiendo las siguientes cuatro operaciones:

`configure(C, F)`: establece C y F, configurando el árbol de distribución de QuFi y sus registros de control. Todas las colas se vacían en este punto.

`write(psum, f_idx)`: escribe una *psum* correspondiente a una *pfiber* con índice f_{idx}. Los datos deben enviarse al puerto de entrada correspondiente de la porción del árbol de distribución del clúster de destino. Se pueden ejecutar hasta C escrituras simultáneamente particionando el árbol de distribución como se explicó anteriormente, siendo C el número de clústeres configurados.

`consume()`: consume la primera *psum* de cada *pfiber* almacenada. Se leen simultáneamente hasta $\lfloor \frac{Q}{F} \rfloor$ *psums* y se envían a los PEs a través de la red de distribución del acelerador junto con una señal de validación para indicar si quedan *psums* pendientes para reducir o no.

`merge()`: indica que se puede iniciar un nuevo paso de *merging*, actualizando sus registros de control.

F. Integración de QuFi en un Acelerador T-Gust

La Fig. 5 muestra un diagrama de la interacción entre un acelerador T-Gust y QuFi, representando cómo se ejecutan paso a paso las fases de multiplicación y *merging* para computar concurrentemente C filas de resultados (asumiendo $Q = R_w = PEs$).

❶ El acelerador debe determinar C dinámicamente en este paso basándose en el número de *pfibers* que se espera almacenar (nnz_i^A). F se determina estáticamente al comienzo de la ejecución. La configuración se establece mediante una operación `configure()`. QuFi se reinicia, es decir, se vacían todas las colas y se reinicializan los registros.

❷ El acelerador comienza la fase de multiplicación. Realizará un total de $\lceil \frac{nnz_i^A \times C}{PEs} \rceil$ pasos de multiplicación. En cada paso, cada PE calculará una *pfiber* que consecuentemente se reducirán a través del árbol de *merging* (ver Fig. 4). Después de eso, se escribirá un total de C *pfibers* en QuFi en paralelo mediante una operación `write()` para cada *psum*. Nótese que si so-

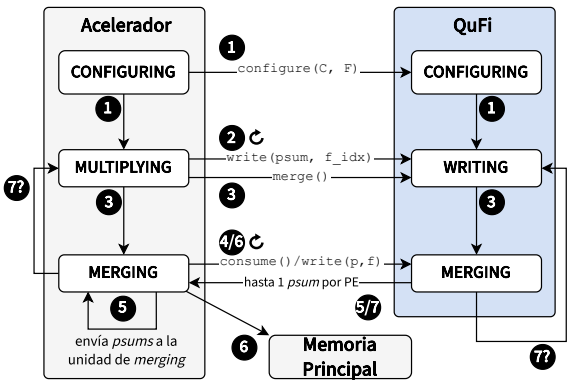


Fig. 5: Diagrama de flujo entre el acelerador y QuFi para calcular una fila de resultados.

lo se necesita un paso de multiplicación, se producirá una fibra final y los datos se enviarán directamente a la memoria principal. Con cada escritura en una q_i, se incrementa el puntero $\texttt{write_ptr}_i$. En caso de usar $F > 1$, la tabla de iteradores de escritura controla en qué q_i se debe escribir una *psum* perteneciente a una f_{idx}. La entrada de la tabla (`write_it`) se actualiza solo cuando la cola anterior de la misma *pfiber* se ha llenado (es decir, $\texttt{write_ptr}_i = \texttt{read_ptr}_i$). Esta fase terminará cuando se hayan calculado todas las *pfibers* dentro del bloque actual o cuando QuFi esté llena (requiriendo un *merging* parcial).

❸ La fase de multiplicación se detiene o finaliza. El acelerador notifica entonces mediante `merge()` que comienza la fase de *merging*. QuFi reinicia los iteradores de escritura y consumo y establece $\texttt{end_ptr}_i \leftarrow \texttt{write_ptr}_i$ para cada q_i, definiendo su rango de lectura como $[\texttt{read_ptr}_i, \texttt{end_ptr}_i)$. En caso de que no sea el último paso de multiplicación, esta fase de *merging* será un *merging* parcial, produciendo una nueva *pfiber* que se almacenará en QuFi con la reducción de todas las *pfibers* generadas en todos los pasos de multiplicación realizados hasta el momento (en este caso, el acelerador debe controlar el número máximo de *pfibers* que puede enviar).

❹ Mediante una operación `consume()`, el acelerador solicita la primera *psum* de cada *pfiber* a reducir. Si $F = 1$, QuFi leerá de todas las q_i. De lo contrario, consultará la tabla de iteradores de consumo y leerá de cada q_i a la que apunte. La primera *psum* de cada q_i se determina mediante su puntero $\texttt{read_ptr}_i$, incrementándolo después de la lectura y consumiendo así dicha *psum*. Cada entrada de la tabla de iteradores de consumo (`consume_it`) se actualizará cuando una q_i se haya leído por completo (es decir, $\texttt{read_ptr}_i = \texttt{end_ptr}_i$).

❺ Para cada q_i seleccionada para la lectura, QuFi envía la primera *psum* al puerto de salida correspondiente. Los pasos ❹ y ❺ se repiten hasta que se hayan consumido todos los *psums*, intercalando ambos pasos en *pipeline* ciclo a ciclo. Mientras tanto, el acelerador realiza el *merging* de *pfibers* a través de su unidad de *merging* mientras recibe las *pfibers*. Nótese que los punteros $\texttt{write_ptr}_i$ y $\texttt{read_ptr}_i$ nunca apuntarán a la misma entrada de la cola, lo que permite operaciones paralelas de `consume()` y `write()` en la misma q_i.

Tabla I: Parámetros de configuración de Flexagon implementando Psram o QuFi.

Flexagon	1 GHz; 64 PEs; 32 bits de precisión (valor+coord.)
Caché L1 (B)	1MiB, tamaño de bloque de 128B, 16 vías, 64 bancos
Memoria ppal.	LPDDR5X (100 ns, 34.1GB/s)
Psram	256KiB (64 filas; 4KiB por fila)
QuFi	64KiB ($Q = 64$, $S = 256$; 1KiB por cola)

❻ Si se han obtenido fibras finales después del *merging* (es decir, se completó el último paso de multiplicación), el acelerador las enviará directamente a la memoria principal. De lo contrario, las *pfibers* se escribirán de nuevo en QuFi mediante `write()` (con $f_{idx} = 1$ para cada c_i) para su posterior reducción. El proceso de escritura se realizará de la misma manera que en el paso ❷.

❼ Una vez finalizada la fase de *merging*, se inicia un nuevo paso de multiplicación, continuando el proceso desde el paso ❷. Si fuera el último, se habrían obtenido las C fibras finales, permitiendo que el cómputo proceda al siguiente bloque.

IV. Metodología

A. Entorno de Simulación

Evaluamos la eficacia de QuFi integrándola en Flexagon [10], un acelerador de DNNs para el *edge* de última generación con memorias *on-chip* no unificadas (véase la Sección II). Para la implementación y evaluación, utilizamos STONNE [17], un simulador a nivel de ciclo para aceleradores de DNN para el *edge*. Dado que STONNE también se empleó para modelar Flexagon, validamos nuestra configuración base de Flexagon+Psram reproduciendo fielmente sus resultados publicados. Posteriormente, ampliamos la unidad de control de Flexagon para dar soporte a T-Gust. La Tabla I enumera los principales parámetros de configuración tanto para el Flexagon original como para el mejorado con QuFi, que sustituye a la Psram. Configuramos $Q = PEs$, ya que Flexagon conecta una única unidad de *merging* a cada PE para maximizar el ancho de banda de reducción (R_w). El área y la energía de cada configuración de memoria *on-chip* se estiman utilizando CACTI 7.0 [24] en tecnología de 28nm.

B. Benchmarks

Para demostrar el impacto práctico de QuFi, evaluamos un conjunto representativo de modelos de DNN *sparse* utilizados en el *edge*, que incluye tres modelos modernos basados en *transformers* y una red neuronal convolucional ampliamente utilizada (véase la Tabla II). Para un estudio detallado por capas, también extraemos múltiples operaciones SpMSpM (incluyendo casos *sparse-dense* de modelos basados en *transformers*) que cubren un amplio rango de dimensiones de matriz, patrones de *sparsity* y demandas computacionales y de *merging* (segunda columna en la Tabla II). Esto asegura que nuestra evaluación recoja diversas características de cargas de trabajo (véase la Fig. 1) e ilustre cómo varían los requisitos de *merging* entre operaciones.

Tabla II: Modelos y capas seleccionados para la evaluación. Los valores de $\mathcal{C}$ son previos a aplicar las funciones de activación.

Modelo (Params–*Sparsity*, Num. Ops.)	**Operación**	M	K	N	$d_{\mathcal{A}}$	$d_{\mathcal{B}}$	$d_{\mathcal{C}}$	**Tamaño (MiB)** $\mathcal{A}/\mathcal{B}/\mathcal{C}$		
ResNet-50 (2016) [20]	30 (conv2d) [R30]	196	2304	256	39.7 %	55.8 %	100 %	0.68	1.25	0.19
Clasificación de Imágenes, 25M–50 %, 54	47 (conv2d) [R47]	49	1024	2048	41.7 %	62.7 %	99.6 %	0.08	5.02	0.38
SAM2 Tiny (2024) [21]	51 (FFN 1) [S51]	1024	768	3072	100 %	67.2 %	100 %	3.00	6.05	12.0
Segmentación de Imágenes, 39M–30 %, 108	52 (FFN 2) [S52]	1024	3072	768	99.6 %	67.6 %	100 %	11.9	6.08	3.00
T5 Small (2020) [22]	4 (QKV proj.) [T4]	15	512	512	100 %	79.8 %	100 %	0.03	0.79	0.03
Generación Text2Text, 60M–50 %, 96	5 (FFN 1) [T5]	15	512	2048	100 %	81.6 %	100 %	0.03	3.26	0.11
Whisper Tiny (2022) [23]	6 (QKV proj.) [W6]	1500	384	384	100 %	50.2 %	99.5 %	2.19	0.28	2.19
Reconocimiento Automático de Voz, 38M–40 %, 827	7 (FFN 1) [W7]	1500	384	1536	100 %	53.9 %	100 %	2.19	1.22	8.78

V. Resultados Experimentales

Comparamos la versión base de Flexagon con una Psram de 256KiB y 64 filas frente a nuevas configuraciones de aceleradores equipados con QuFi de diferentes tamaños: 32KiB ($S = 128$), 64KiB ($S = 256$) y 128KiB ($S = 512$). También incluimos una versión ideal de QuFi, capaz de almacenar *pfibers* completas sin requerir fusión de colas. Esta configuración representa efectivamente el rendimiento máximo alcanzable durante la fase de *merging* utilizando QuFi.

A. Evaluación End-to-End

La Fig. 6 compara el rendimiento *end-to-end* de cada configuración para cada modelo de DNN, con un desglose del tiempo consumido por las fases de multiplicación y *merging*. Como se puede observar, todas las configuraciones de QuFi obtienen un mejor rendimiento general que la Psram, incluso cuando se utiliza una memoria 8 veces menor (32KiB). Esto se traduce en una mejora media de 1,64× (64KiB QuFi) para la inferencia end-to-end. Cabe destacar que la fase de *merging* puede convertirse en un cuello de botella de rendimiento importante, consumiendo hasta el 49 % del tiempo total de ejecución cuando no está debidamente optimizada (p. ej., Psram, SAM2). En cambio, una QuFi ideal reduce la sobrecarga a solo el 1,8 % (p. ej., T5-Small), una mejora que QuFi puede casi alcanzar con un tamaño de memoria *on-chip* significativamente menor.

Además, la Fig. 7 contabiliza el tráfico total *off-chip* generado por cada tipo de matriz en todas las operaciones SpMSpM de cada modelo de DNN. Como se muestra, una QuFi de 32KiB ya elimina por completo, para nuestros *benchmarks*, el tráfico *off-chip* de *psums* para las operaciones de lectura/escritura durante la fase *merging* (véase la Sección V-B.1), lo que resulta en reducciones considerables del tráfico total *off-chip* que van desde el 13 % (T5-Small, 32MiB) hasta el 73 % (SAM2, 6.4GiB) en comparación con Psram.

B. Evaluación por Capas

Para una evaluación detallada, analizamos capas representativas de los modelos de DNN. La Fig. 8 muestra el rendimiento, mientras que la Fig. 9 presenta el número de pasos de *merging* y el número de *pfibers* reducidas por paso. Como podemos observar, QuFi logra un rendimiento de *merging* cercano al ideal configurando F como se describe en la Sección III-C.2 y ajustando C (y, por tanto, T) para maximizar el número de *pfibers* almacenadas si-

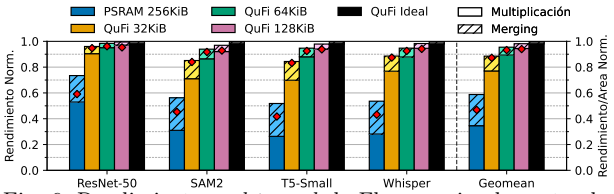


Fig. 6: Rendimiento *end-to-end* de Flexagon implementando Psram y QuFi normalizado a QuFi ideal. Rendimiento/Área normalizado al Flexagon base.

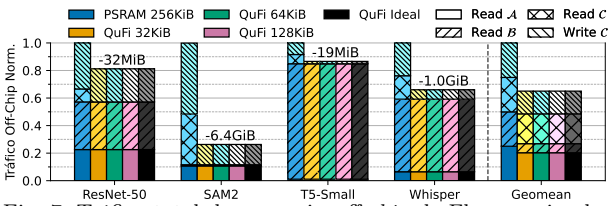


Fig. 7: Tráfico total de memoria *off-chip* de Flexagon implementando Psram y QuFi normalizado a QuFi ideal. La reducción total de tráfico de QuFi se muestra sobre las barras.

multáneamente, reduciendo así el tráfico de memoria *off-chip* y los pasos de *merging* requeridos.

B.1 Impacto del tamaño de las *pfibers*

Con la configuración original, una Psram solo puede almacenar de forma efectiva *pfibers* de hasta 512 elementos (es decir, $N \leq 512$) mediante su técnica de combinación de vías, a costa de reducir el número máximo de *pfibers* que puede fusionar simultáneamente, similar al enfoque de QuFi. Esto es problemático especialmente en capas grandes (R47, S51, S52, T5, W7) donde $N > 512$, pues las *psums* restantes de las *pfibers* deben escribirse y leerse de nuevo en la memoria principal, incurriendo en movimiento de datos *off-chip* de 7MiB, 440MiB, 188MiB, 2.4MiB y 118MiB respectivamente. Esto implica una latencia adicional que puede representar hasta el 8.8 % (S51) del tiempo total de la operación SpMSpM.

Gracias al diseño adaptable de QuFi, estos límites son más flexibles, soportando eficientemente hasta $N \leq 4096$ en la configuración más pequeña de 32KiB, lo suficiente para cubrir el valor de N más grande en todas las operaciones SpMSpM de los modelos (S52). Esto demuestra que, para los modelos de DNN para el *edge*, una QuFi pequeña puede manejar eficazmente todo el rango de SpMSpM con poco o ningún tráfico de memoria *off-chip* para el *merging*, superando así significativamente a los aceleradores equipados con Psram del estado del arte.

Sin embargo, el parámetro S establece la granularidad mínima para almacenar una *pfiber*, lo que puede provocar pasos de *merging* adicionales ya que caben menos *pfibers* en memoria simultáneamente

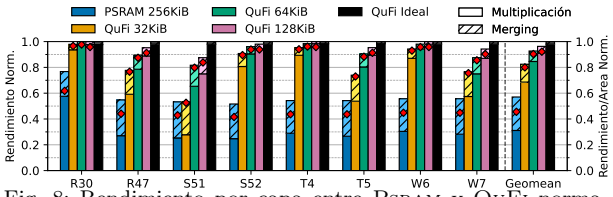


Fig. 8: Rendimiento por capa entre PSRAM y QUFI normalizado a QUFI ilimitado. Rendimiento/Área normalizado al Flexagon base.

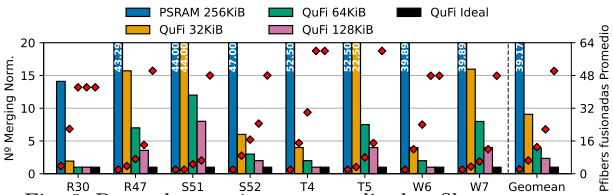


Fig. 9: Pasos de *merging* y promedio de *pfibers* por paso.

cuando S es pequeño (requiriendo fusión de colas, que se analiza más adelante) o una baja ocupación de memoria debido a la fragmentación interna cuando S es grande. Para adaptarse mejor al amplio rango de tamaños de *pfiber* (N), seleccionamos para nuestra evaluación diferentes configuraciones de S para QUFI. Esta muestra que QUFI, para cada configuración de S, logra tasas de ocupación de memoria promedio del 71 % ($S = 128$), 48 % ($S = 256$) y 30 % ($S = 512$) en todos los pasos de *merging* de todos los modelos completos, lo que sugiere que sería más efectivo aumentar Q en lugar de S cuando se superan valores mayores o iguales a 512. En contraste, la PSRAM, que carece de adaptabilidad dinámica, solo puede utilizar T conjuntos (con $T = 2$ en la mayoría de las capas) de los 64 simultáneamente, lo que conduce a una ocupación de memoria muy baja (menos del 3 % en promedio).

B.2 Impacto del número de *pfibers*

En QUFI, las configuraciones de S más grandes provocan una configuración de colas fusionadas (F) más pequeña, lo que permite un mayor ancho de banda en la fusión ya que se pueden combinar más *pfibers* en un solo paso de *merging* (hasta R_w en el escenario de QUFI ideal). Como muestra la Fig. 9, esto aumenta eficazmente el número de *pfibers* que se reducen de media en un solo paso, decrementando significativamente el tiempo de la fase de *merging*. Por ejemplo, una PSRAM de 256KiB requiere un promedio de 39,17× pasos de *merging* adicionales en comparación con el número mínimo de pasos necesarios utilizando la misma unidad de *merging*, mientras que un QUFI de 64KiB requiere 3,96× más pasos que el ideal.

Nótese que esta diferencia se debe a los pasos de *merging* parcial que tanto la PSRAM como QUFI deben realizar debido a las limitaciones de tamaño de memoria, a diferencia del caso ideal en que solo se realizan pasos de *merging* finales. También se puede observar que, para QUFI, el número de pasos es casi proporcional a la capacidad de la estructura de memoria utilizada para mantener las *pfibers*, pasando de 9,07× (32KiB) a 2,34× (128KiB) pasos de *merging* adicionales en promedio en comparación con el ideal. En estos casos, a pesar de tener que realizar pasos de *merging* adicionales debido a la fusión de colas y a una menor capacidad para aplicar agrupamiento, se puede observar en la Fig. 8 que su impacto en el rendimiento es poco significativo, ya que la fase de multiplicación sigue requiriendo la mayor parte del tiempo de cómputo.

C. Resultados de RTL

El área base de Flexagon ocupa 4,25mm^2. Una PSRAM de 256KiB añade 1,03mm^2 de área, lo que resulta en una sobrecarga de 1,24× sobre el acelerador base. En cambio, una instancia de 64KiB de QUFI requiere solo 0,10mm^2, introduciendo una sobrecarga mínima de 1,02×. Esta reducción significativa del área se debe a la drástica simplificación de la lógica de control de memoria, a pesar de reducir el almacenamiento efectivo solo en 4×. Como se muestra en la Fig. 6 y en la Fig. 8, los diamantes rojos representan las comparaciones de rendimiento/área para ambos tipos de memoria integrados en Flexagon, donde una QUFI de 64KiB demuestra una mejora promedio de 1,63× sobre una PSRAM de 256KiB.

VI. Conclusión

En este trabajo presentamos QUFI, una estructura de memoria para *merging* especializada y adaptable diseñada específicamente para ser integrada en aceleradores T-Gust que ejecutan modelos de DNN *sparse* para el *edge*. A diferencia de los diseños rígidos o basados en caché propuestos anteriormente, QUFI ofrece una alta adaptabilidad y alto de ancho de banda a través de una estructura reconfigurable que emplea la fusión de colas y agrupamiento dinámico.

Nuestra evaluación muestra que QUFI mejora significativamente el rendimiento, logrando una eficiencia de *merging* cercana a la ideal a la vez que reduce el área del acelerador en un 17 % en comparación con diseños del estado del arte como PSRAM. Estos resultados resaltan la importancia de utilizar memorias adaptativas en aceleradores T-Gust y posicionan a QUFI como una solución práctica y eficiente para los sistemas de IA para el *edge* de próxima generación.

Agradecimientos

Este trabajo ha sido financiado por el proyecto MICIU/AEI/10.13039/501100011033/ y el programa "FEDER Una manera de hacer Europa", de la UE, en el marco de la subvención PID2022-136315OB-I00; por MICIU/AEI/10.13039/501100011033 y por el "NextGenerationEU/PRTR de la Unión Europea", en el marco de la subvención RYC2021-031966-I. Este trabajo también ha sido financiado parcialmente por el proyecto de la UE dAIEDGE (n.º de subvención 101120726) y el programa "Innotave UK Horizon Europe Guarantee" (n.º de subvención 10090788). Adrián Navarro ha sido financiado mediante la ayuda 22294/FPI/23 de Fundación Séneca, Agencia de Ciencia y Tecnología de la Región de Murcia.

Referencias

[1] Marek Żyliński, Amir Nassibi, Ildar Rakhmatulin, Adil Malik, Christos Michael Papavassiliou, and Danilo P. Mandic, "Deployment of artificial intelligence models on edge devices: A tutorial brief," *TCAS-II*, vol. 71, no. 3, 2024.

[2] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proc. IEEE*, vol. 108, no. 4, 2020.

[3] Ofir Zafrir, Ariel Larey, Guy Boudoukh, Haihao Shen, and Moshe Wasserblat, "Prune once for all: Sparse pre-trained language models," *arXiv:2111.05754*, 2021.

[4] Elias Frantar and Dan Alistarh, "SparseGPT: Massive language models can be accurately pruned in one-shot," *arXiv:2301.00774*, 2023.

[5] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna, "SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training," in *HPCA*, 2020.

[6] Subhankar Pal, Jonathan Beaumont, Dong-Hyeon Park, Aporva Amarnath, Siying Feng, Chaitali Chakrabarti, Hun-Seok Kim, David Blaauw, Trevor Mudge, and Ronald Dreslinski, "OuterSPACE: An outer product based sparse matrix multiplication accelerator," in *HPCA*, 2018.

[7] Zhekai Zhang, Hanrui Wang, Song Han, and William J. Dally, "SpArch: Efficient architecture for sparse matrix multiplication," in *HPCA*, 2020.

[8] Nitish Srivastava, Hanchen Jin, Jie Liu, David Albonesi, and Zhiru Zhang, "MatRaptor: A sparse-sparse matrix multiplication accelerator based on row-wise product," in *MICRO*, 2020.

[9] Guowei Zhang, Nithya Attaluri, Joel S. Emer, and Daniel Sanchez, "Gamma: leveraging Gustavson's algorithm to accelerate sparse matrix multiplication," in *ASPLOS*, 2021.

[10] Francisco Muñoz-Martínez, Raveesh Garg, Michael Pellauer, José L. Abellán, Manuel E. Acacio, and Tushar Krishna, "Flexagon: A multi-dataflow sparse-sparse matrix multiplication accelerator for efficient DNN processing," in *ASPLOS*, 2023.

[11] Zhiyao Li, Jiaxiang Li, Taijie Chen, Dimin Niu, Hongzhong Zheng, Yuan Xie, and Mingyu Gao, "Spada: Accelerating sparse matrix multiplication with adaptive dataflow," in *ASPLOS*, 2023.

[12] Y. Yang, J. S. Emer, and D. Sanchez, "Trapezoid: A versatile accelerator for dense and sparse matrix multiplications," in *ISCA*, 2024.

[13] Xiaoyang Lu, Boyu Long, Xiaoming Chen, Yinhe Han, and Xian-He Sun, "ACES: Accelerating sparse matrix multiplication with adaptive execution flow and concurrency-aware cache optimizations," in *ASPLOS*, 2024.

[14] Kai Zhong, Zhenhua Zhu, Guohao Dai, Hongyi Wang, Xinhao Yang, Haoyu Zhang, Jin Si, Qiuli Mao, Shulin Zeng, Ke Hong, Genghan Zhang, Huazhong Yang, and Yu Wang, "FEASTA: A flexible and efficient accelerator for sparse tensor algebra in machine learning," in *ASPLOS*, 2024.

[15] Gwangeun Byeon, Seongwook Kim, Hyungjin Kim, Sukhyun Han, Jinkwon Kim, Prashant Nair, Taewook Kang, and Seokin Hong, "Avalanche: Optimizing cache utilization via matrix reordering for sparse matrix multiplication accelerator," in *ISCA*, 2025.

[16] Kaustubh Shivdikar, Nicolas Bohm Agostini, Malith Jayaweera, Gilbert Jonatan, José L. Abellán, Ajay Joshi, John Kim, and David Kaeli, "NeuraChip: Accelerating GNN Computations with a Hash-Based Decoupled Spatial Accelerator," in *ISCA*, 2024.

[17] Francisco Muñoz-Martínez, José L. Abellán, Manuel E. Acacio, and Tushar Krishna, "STONNE: Enabling cycle-level microarchitectural simulation for dnn inference accelerators," in *IISWC*, 2021.

[18] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *ISCA*, 2017.

[19] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *JETCAS*, vol. 9, no. 2, 2019.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016.

[21] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer, "SAM 2: Segment anything in images and videos," *arXiv:2408.00714*, 2024.

[22] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *JMLR*, vol. 21, no. 140, 2020.

[23] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever, "Robust speech recognition via large-scale weak supervision," *arXiv:2212.04356*, 2022.

[24] Rajeev Balasubramonian, Andrew B. Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas, "CACTI 7: New tools for interconnect exploration in innovative off-chip memories," *TACO*, vol. 14, no. 2, 2017.

Aplicaciones de la computación de altas prestaciones (2/3)

Paralelizando el método TSLLS para la modelización de paneles fotovoltaicos

X. Moreno-Vassart[1], Vicente Galiano[1], F. Javier Toledo[2] y Victoria Herranz[2]

Resumen— **El modelo fotovoltaico de un solo diodo constituye una de las herramientas más utilizadas para representar la curva corriente–tensión de celdas y módulos fotovoltaicos, ya que combina interpretabilidad física, capacidad descriptiva y un coste computacional moderado. Dentro de los métodos recientes de identificación paramétrica, el procedimiento *Two-Step Linear Least-Squares* (TSLLS) destaca por su capacidad para trabajar de forma ciega a partir de puntos experimentales de la curva I-V, sin requerir estimaciones iniciales de I_{sc}, V_{oc} o del punto de máxima potencia. Esta propiedad le permite automatizar el ajuste de grandes colecciones de curvas y lo ha convertido en una referencia especialmente adecuada para el tratamiento de repositorios masivos.**

La aplicación del TSLLS sobre el repositorio público del *National Renewable Energy Laboratory* (NREL), con más de un millón de curvas, pone de manifiesto no solo la robustez del método, sino también la necesidad de reconsiderar su estructura algorítmica desde la perspectiva del paralelismo. En este trabajo se propone una reformulación paralela del algoritmo apoyada en dos estrategias complementarias. La primera consiste en la diferenciación previa de curvas para organizar el procesamiento en lotes más homogéneos desde el punto de vista tecnológico, geométrico y computacional. La segunda consiste en la paralelización interna de los bucles del algoritmo TSLLS, esto es, en la distribución entre distintos procesos de las iteraciones que recorren las combinaciones candidatas dentro de cada curva. De este modo, varios hilos o procesos exploran simultáneamente las configuraciones admisibles y convergen después en una reducción final basada en el menor RMSE. La propuesta se fundamenta en la estructura geométrica del propio modelo y en desarrollos recientes del grupo de Toledo y colaboradores sobre parametrización, representación eficiente de curvas y análisis de puntos notables de curvas fotovoltaicas.

I. Introducción

La modelización eléctrica de módulos fotovoltaicos es un elemento central en la predicción del comportamiento energético, la supervisión del estado de degradación y el diseño de estrategias de seguimiento del punto de máxima potencia [1], [2], [3]. Entre las distintas formulaciones disponibles, el modelo de un solo diodo se ha consolidado como una de las más empleadas, al proporcionar un equilibrio razonable entre fidelidad física y complejidad numérica [4], [5], [1]. Su uso se extiende tanto al ajuste de curvas experimentales como al análisis a partir de datos de fabricante y a la simulación de módulos y campos fotovoltaicos [6], [7], [8].

La dificultad principal no reside en la escritura de la ecuación del modelo, sino en la extracción fiable de sus parámetros. La bibliografía ha propuesto estrategias muy diversas: métodos analíticos, procedimientos cuasi-explícitos, formulaciones reducidas, técnicas de ajuste lineal, enfoques iterativos y algoritmos metaheurísticos [9], [10], [11], [12], [13], [14]. Sin embargo, muchos de estos métodos dependen de una elección inicial adecuada, de la disponibilidad de datos operativos específicos o de un coste computacional que dificulta su uso en escenarios de procesamiento masivo [12], [14], [15].

En este contexto, el trabajo de Toledo, Blanes y Galiano de 2018 introdujo el método TSLLS, que explota la estructura geométrica del modelo de un diodo y separa el problema en una etapa lineal y otra exponencial, ambas resueltas mediante mínimos cuadrados lineales [15]. El rasgo más relevante del método es su carácter ciego: basta disponer de $N \geq 5$ puntos experimentales de la curva I-V, sin necesidad de conocer de antemano I_{sc}, V_{oc}, el punto de máxima potencia o una semilla numérica cercana a la solución [15]. En los casos de estudio clásicos, sus resultados son competitivos con los mejores métodos documentados, y en un segundo estadio de refinamiento alcanzan incluso mejoras sobre la literatura previa [12], [14], [15].

La relevancia del TSLLS se incrementa todavía más al comprobar que pudo aplicarse de forma automática a un gran repositorio del NREL compuesto por 1 025 599 curvas I-V, procedentes de 22 módulos, ocho tecnologías y tres emplazamientos climáticamente distintos [15], [16], [17]. El propio artículo original indica que ese procesamiento se automatizó mediante un script en Python ejecutado sobre un clúster de 20 máquinas [15]. Sin embargo, aunque ese antecedente demuestra la viabilidad del despliegue a gran escala, no desarrolla una formulación explícita de paralelización del algoritmo ni una discusión detallada de su estructura interna desde la óptica del paralelismo.

El objetivo del presente trabajo es precisamente cubrir ese hueco. Se propone una reformulación paralela del TSLLS en dos niveles. El primero es una paralelización macroscópica basada en la diferenciación previa de curvas, con el fin de construir

[1]Dpto. de Ingeniería de Computadores, Universidad Miguel Hernández de Elche, e-mail: {xmoreno-vassart,vgaliano}@umh.es

[2]Centro de Investigación Operativa, Universidad Miguel Hernández de Elche, e-mail: {javier.toledo,mavi.herranz}@umh.es

lotes de procesamiento más homogéneos y mejorar el balance de carga. El segundo es una paralelización microscópica por bucles, en la que las iteraciones internas del TSLLS —encargadas de recorrer las combinaciones candidatas de segmentos lineales y exponenciales— se distribuyen entre varios hilos o procesos concurrentes. Ambas estrategias respetan la filosofía del TSLLS original y se apoyan en desarrollos posteriores relacionados con la parametrización del modelo, la obtención eficiente de puntos de la curva, la geometría de puntos singulares y la interpretación estadística de grandes repositorios fotovoltaicos [10], [18], [19], [17], [20].

El método TSLLS se encuentra disponible a través de la plataforma `pvmodel.umh.es`, concebida como una herramienta de acceso libre para la obtención de los parámetros del modelo de un diodo a partir de datos experimentales. No obstante, en el desarrollo y mantenimiento de la plataforma, se han detectado dificultades cuando el número de puntos es elevado, afectando a la calidad de servicio ofrecida a otros usuarios y perceptible en la degradación en los tiempos de cómputo. La paralelización de dicho método facilita un mejor uso de los recursos, mejorando asimismo la respuesta de la herramienta.

II. Modelización fotovoltaica y antecedentes

II-A. Modelo de un solo diodo

La ecuación del modelo de un diodo para un panel fotovoltaico con n_s celdas en serie y n_p ramas en paralelo puede expresarse como [15], [18]

$$I = n_p I_{ph} - n_p I_{sat}\left(\exp\left(\frac{V/n_s + IR_s/n_p}{nV_T}\right) - 1\right) - n_p\frac{V/n_s + IR_s/n_p}{R_{sh}}, \tag{1}$$

donde I_{ph} es la fotocorriente, I_{sat} la corriente de saturación del diodo, R_s la resistencia serie, R_{sh} la resistencia en derivación, n el factor de idealidad y $V_T = kT/q$ la tensión térmica [4], [5], [15].

Siguiendo la reparametrización empleada por Toledo y colaboradores [15], [20], se introducen los parámetros

$$A = n_p I_{ph}\frac{R_{sh}}{R_{sh}+R_s}, \qquad B = n_p I_{sat}\frac{R_{sh}}{R_{sh}+R_s}, \qquad C = e^{1/(n_s n V_T)}, \qquad D = e^{R_s/(n_p n V_T)}, \qquad E = \frac{n_p}{n_s(R_{sh}+R_s)} \tag{2}$$

con lo que la ecuación adopta la forma compacta

$$I = A - B\left(C^V D^I - 1\right) - EV. \tag{3}$$

Equivalentemente,

$$I = A + B - EV - BC^V D^I. \tag{4}$$

La importancia de 4 es geométrica. Permite identificar una componente lineal, $A+B-EV$, asociada a la asíntota oblicua de la curva, y una componente exponencial, $BC^V D^I$, responsable del codo característico [10], [15]. Esta descomposición no solo estructura el método TSLLS, sino que también proporciona la base conceptual para su reorganización paralela.

II-B. Panorama de métodos de extracción de parámetros

Los métodos de identificación del modelo de un diodo pueden clasificarse, de forma amplia, en procedimientos basados en hoja de datos, en procedimientos basados en puntos experimentales I-V, y en métodos híbridos que combinan ambos tipos de información [15]. Dentro de estas familias se encuentran formulaciones reducidas [6], [12], procedimientos cuasi-explícitos [21], ajustes lineales [13], estrategias de búsqueda reducida [14] y algoritmos metaheurísticos [11].

Laudani, Riganti-Fulginei y Salvini propusieron una formulación reducida de alta precisión para el modelo de un diodo a partir de curvas experimentales, pero con dependencia sensible respecto a ciertos datos operativos, en particular I_{sc}, V_{oc} y el punto de máxima potencia [12]. Por su parte, Cárdenas y colaboradores desarrollaron una búsqueda en espacio reducido que mejora la eficiencia del ajuste, aunque sigue requiriendo una inicialización suficientemente próxima al óptimo para asegurar convergencia y reducir el número de iteraciones [14]. Frente a ello, el TSLLS elimina la dependencia de conjeturas iniciales y de información auxiliar, lo que lo vuelve especialmente atractivo en escenarios automatizados [15].

II-C. El método TSLLS

El TSLLS se basa en una idea muy simple y poderosa[15]: extraer por separado, aunque no independientemente, la parte lineal y la parte exponencial de la ecuación 4. Dados N puntos experimentales $\{(V_j, I_j)\}_{j=1}^{N}$, la primera etapa asume que el inicio de la curva puede aproximarse por

$$I \approx K - EV, \qquad K = A + B, \tag{5}$$

y estima K y E mediante un ajuste de mínimos cuadrados lineales sobre un subconjunto inicial de puntos [15]. La segunda etapa utiliza esa estimación para reescribir la parte exponencial como otro problema lineal en las variables $\ln(B)$, $\ln(C)$ y $\ln(D)$, ahora sobre un subconjunto final de la curva [15]. Finalmente, el algoritmo explora combinaciones ad-

misibles de ambos subconjuntos y selecciona aquella que minimiza el RMSE global [15].

El método fue contrastado sobre dos casos clásicos de la literatura y sobre el repositorio NREL. En los casos de estudio habituales, la precisión inicial del TSLLS es del mismo orden que la de los mejores trabajos previos, y el refinamiento posterior mejora aún más los resultados [12], [14], [15]. En el repositorio NREL, Toledo et al. reportan un RMSE medio de $3{,}07 \times 10^{-3}$ sin refinamiento y de $2{,}46 \times 10^{-3}$ con refinamiento, utilizando más de un millón de curvas procesadas de forma automática [15]. El hecho de que ese resultado se obtenga sin estimaciones iniciales es una de las principales fortalezas del método.

III. Motivación para la paralelización

La conveniencia de paralelizar TSLLS se deriva de dos hechos complementarios. El primero es externo al algoritmo: los grandes repositorios fotovoltaicos son heterogéneos, tanto en tecnología como en morfología de curva y condiciones de operación [16], [17]. El segundo es interno: el propio TSLLS contiene un espacio natural de candidatos, formado por las combinaciones admisibles de tramos lineales y exponenciales, cuya evaluación puede desacoplarse de manera concurrente [15].

Esta motivación se refuerza con los trabajos recientes del mismo entorno científico. La parametrización del modelo de un diodo ha sido explotada para el cálculo rápido de soluciones de la ecuación implícita [18], para la representación eficiente de la curva I-V mediante selección adaptativa de puntos [19], para el análisis de relaciones entre puntos notables de curvas procedentes del NREL [17] y para el estudio geométrico asociado a distancias euclídeas a la curva [20]. En conjunto, estos trabajos sugieren que el modelo no debe verse solo como una ecuación a resolver, sino como una estructura geométrica susceptible de reorganización computacional.

En consecuencia, la paralelización que se propone aquí no pretende alterar el criterio matemático del TSLLS, sino rediseñar su ejecución. El problema no es obtener otros parámetros distintos, sino alcanzar la misma lógica de selección con una explotación más eficiente de recursos multinúcleo o distribuidos.

IV. Estrategia 1: diferenciación de curvas en el análisis NREL

IV-A. Heterogeneidad del repositorio

El repositorio utilizado por Toledo et al. incluye 1 025 599 curvas I-V de 22 módulos, ocho tecnologías y tres localizaciones —Cocoa, Eugene y Golden— con medidas registradas cada cinco minutos [15], [16], [17]. Moreno-Vassart et al. vuelven a utilizar ese repositorio y detallan la distribución de tecnologías y módulos, poniendo de manifiesto la fuerte heterogeneidad del conjunto de datos [17]. No todas las curvas presentan la misma complejidad numérica ni la misma facilidad de ajuste: la baja irradiancia, el sombreado parcial y ciertos estados de degradación pueden inducir soluciones menos estables o parámetros no físicos [15], [22].

Por tanto, repartir simplemente el número total de curvas entre procesos disponibles no garantiza un buen balance de carga. En un lote uniformemente dimensionado pueden coexistir curvas casi triviales con otras que obliguen a recorrer un espacio de candidatos mucho más costoso. Esta disparidad provoca tiempos de finalización muy distintos entre tareas aparentemente equivalentes y reduce la eficiencia global del sistema.

IV-B. Propuesta de diferenciación previa

Se propone una fase previa de clasificación de curvas antes de la ejecución de TSLLS. Esta diferenciación puede realizarse en tres niveles:

1. **Metadatos de origen:** tecnología, emplazamiento, identificador de módulo y franja temporal [16], [17].
2. **Rasgos geométricos de la curva:** longitud efectiva del tramo casi lineal, pendiente inicial, curvatura en el codo o relaciones entre puntos notables, siguiendo la intuición geométrica desarrollada en trabajos previos [10], [17], [19].
3. **Coste computacional estimado:** número de combinaciones potencialmente viables, probabilidad de requerir refinamiento y sensibilidad del ajuste.

La idea no es imponer una taxonomía cerrada, sino construir colas de trabajo más homogéneas. Por ejemplo, curvas de una misma tecnología y con una forma similar del codo pueden agruparse para ser procesadas por un mismo conjunto de procesos. Del mismo modo, curvas que muestren indicios de sombreado parcial o baja irradiancia pueden derivarse a una cola con políticas de verificación más estrictas.

IV-C. Ventajas computacionales y metodológicas

Esta diferenciación previa aporta varias ventajas. En primer lugar, mejora el balance de carga, porque reduce la varianza del tiempo por lote. En segundo lugar, permite adaptar la granularidad del paralelismo al tipo de curvas dominante. En tercer lugar, facilita la lectura científica de los resultados, ya que la precisión del método no se analiza solo mediante promedios globales, sino también por familias de comportamiento [17]. En cuarto lugar, abre la posibilidad de aplicar estrategias de refinamiento selectivo, reservando los procedimientos

más costosos para las curvas que realmente lo necesiten.

V. Estrategia 2: paralelización de los bucles internos del TSLLS

V-A. Distribución de bucles y concurrencia natural

El núcleo del TSLLS consiste en explorar diferentes combinaciones de subconjuntos iniciales y finales de la curva para determinar, respectivamente, la parte lineal y la parte exponencial [15]. En una implementación secuencial, esta exploración se materializa mediante bucles anidados que recorren las combinaciones admisibles y calculan, para cada iteración, un conjunto candidato de parámetros junto con su error asociado. Por ello, aunque en una descripción conceptual pueda hablarse de *bloques candidatos*, desde el punto de vista de implementación la segunda estrategia de paralelización consiste realmente en distribuir entre procesos las iteraciones de esos bucles internos.

Si denotamos por

$$\mathcal{B} = \{\mathcal{B}_\infty, \mathcal{B}_\in, \ldots, \mathcal{B}_{\Updownarrow}\}$$

el conjunto de configuraciones candidatas generado por dichos bucles, cada elemento B_i representa una hipótesis concreta sobre la estructura de la curva. La concurrencia no nace, por tanto, de una descomposición externa arbitraria, sino de la posibilidad de evaluar en paralelo distintas iteraciones del recorrido interno del algoritmo. Esa independencia convierte el espacio de candidatos en un paralelismo de grano medio especialmente adecuado para arquitecturas multinúcleo.

V-B. Definición de la iteración de trabajo

Cada iteración asignada a un proceso, asociada a un candidato B_i, realiza:

i) selección del subconjunto inicial para la parte lineal;
ii) ajuste por mínimos cuadrados para estimar K y E;
iii) selección del subconjunto final para la parte exponencial;
iv) ajuste lineal de $\ln(B)$, $\ln(C)$ y $\ln(D)$;
v) recuperación de A y transformación a parámetros físicos;
vi) evaluación del RMSE y, opcionalmente, del MAE;
vii) validación física o descarte por restricciones de plausibilidad.

El resultado puede almacenarse como una tupla

$$T_i = \big(B_i, \theta_i, \mathrm{RMSE_i}, \mathrm{MAE_i}, \nu_i\big),$$

donde θ_i representa el vector de parámetros y ν_i resume las banderas de validez.

V-C. Reducción global

Una vez evaluadas en paralelo las iteraciones distribuidas de los bucles internos, un proceso agregador central ejecuta una reducción final:

$$B^\star = \arg \min_{B_i \in \mathcal{B}_{\text{validos}}} \mathrm{RMSE(B_i)}.$$

Esta etapa conserva exactamente el criterio del TSLLS original [15]. No se altera el objetivo de optimización; simplemente se evita que la exploración del conjunto $\mathcal{B}$ dependa de un único flujo secuencial. Si se desea, la mejor solución seleccionada puede someterse después a un refinamiento local, como en el trabajo original [15].

V-D. Diagrama de flujo

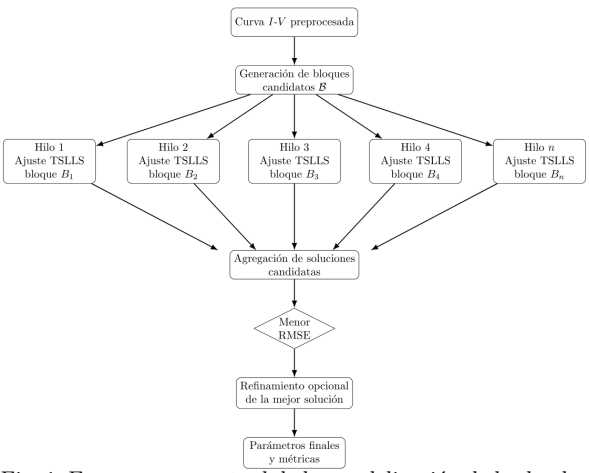


Fig. 1: Esquema conceptual de la paralelización de los bucles internos del TSLLS.

VI. Consideraciones de implementación en Python

La implementación práctica en Python puede organizarse en dos niveles. Para la distribución de lotes de curvas, resultan naturales `multiprocessing` o `concurrent.futures.ProcessPoolExecutor`. Para la evaluación de las iteraciones de los bucles internos dentro de una misma curva, también puede emplearse un `ProcessPoolExecutor` o, cuando predominan cálculos vectorizados en bibliotecas numéricas que liberan el *Global Interpreter Lock*, un esquema basado en hilos [15], [19].

No obstante, el rendimiento real depende de tres decisiones de diseño: la granularidad del bloque, la localidad de datos y la reproducibilidad del criterio final. Si el bloque es demasiado pequeño, la sobrecarga de sincronización puede anular el beneficio del paralelismo. Si las curvas se copian de forma repetida, el consumo de memoria puede convertirse en el cuello de botella. Y si no se fijan reglas de

desempate, dos ejecuciones distintas podrían seleccionar candidatos diferentes con RMSE casi idéntico.

VII. Resultados experimentales

VII-A. Resultados de referencia en la literatura

Antes de incorporar los resultados de la implementación paralela propuesta, conviene fijar una línea base extraída de la literatura. El trabajo original de TSLLS reporta, sobre el repositorio NREL, un RMSE medio de $3{,}07 \times 10^{-3}$ y un MAE medio de $2{,}45 \times 10^{-3}$ sin refinamiento; con refinamiento, esos valores descienden a $2{,}46 \times 10^{-3}$ y $1{,}99 \times 10^{-3}$, respectivamente [15]. Además, dicho trabajo indica que el procesamiento masivo se automatizó mediante un script en Python sobre un clúster de 20 máquinas [15]. Estos valores no corresponden a la implementación paralela aquí propuesta, pero sí constituyen una referencia obligada para cualquier comparación futura.

VII-B. Diseño experimental de la versión paralela

Para evaluar la reformulación propuesta se sugiere distinguir dos niveles de experimento.

VII-B.1. Nivel 1: diferenciación de curvas.

Debe medirse el tiempo total de ejecución del procesamiento masivo cuando las curvas se reparten de manera uniforme y cuando se distribuyen tras una fase de clasificación previa. Interesa comparar:

- tiempo total;
- desviación estándar del tiempo por lote;
- *speedup*;
- eficiencia paralela;
- estabilidad del RMSE medio respecto al algoritmo secuencial.

VII-B.2. Nivel 2: paralelización de bucles internos.

En este segundo nivel conviene registrar:

- tiempo medio por curva;
- número de iteraciones/candidatos evaluados;
- porcentaje de bloques descartados por inviabilidad;
- coste de la etapa de reducción;
- impacto temporal y numérico del refinamiento final.

VII-C. Métricas

En el plano computacional, las métricas a tener en cuenta son las siguientes:

$$S(p) = \frac{T_1}{T_p}, \qquad E(p) = \frac{S(p)}{p}, \tag{6}$$

siendo T_1 el tiempo secuencial, T_p el tiempo con p procesos, $S(p)$ el *speedup* y $E(p)$ la eficiencia.

La métrica de ajuste de la práctica del TSLLS original [15]:

$$RMSE = \sqrt{\frac{1}{N}\sum_{j=1}^{N}\left(I_j^p - I_j\right)^2}, \tag{7}$$

donde I_j es la corriente observada e I_j^p la predicha por el modelo, es la misma en todas las ejecuciones con independencia de la paralelización empleada.

VII-D. Análisis de la paralelización por diferenciación de curvas

Los tiempos obtenidos sobre 100 curvas sintéticas de 500 puntos muestran una reducción monótona del tiempo total de ejecución al aumentar el número de procesos de trabajo. En particular, el tiempo pasa de 2959,56 s en ejecución secuencial a 171,48 s con 32 procesos, lo que equivale a un *speedup* de 17,26 (Tabla I). La eficiencia paralela desciende progresivamente desde 0,934 con 2 procesos hasta 0,539 con 32 procesos, comportamiento que resulta coherente con la aparición de costes de coordinación, sincronización y reparto de carga, así como con la presencia de una fracción no paralelizable del flujo de trabajo.

Tabla I: Resultados experimentales de la paralelización por diferenciación de curvas sobre 100 curvas sintéticas de 500 puntos.

Workers	Tiempo total (s)	Speedup	Eficiencia
1	2959.561	1.000	1.000
2	1584.654	1.868	0.934
4	860.410	3.440	0.860
8	484.937	6.103	0.763
16	279.137	10.603	0.663
24	224.620	13.176	0.549
32	171.476	17.259	0.539

Desde el punto de vista computacional, los resultados indican que la estrategia de diferenciación de curvas es claramente eficaz para reducir el tiempo de procesamiento total. El crecimiento del *speedup* es especialmente intenso entre 1 y 16 procesos, mientras que a partir de ese punto siguen observándose mejoras, aunque con rendimientos marginales decrecientes. Este comportamiento es el esperado en un esquema de paralelización de grano grueso, donde la disminución del tiempo de cómputo convive con un aumento relativo de las sobrecargas de gestión a medida que crece el paralelismo disponible.

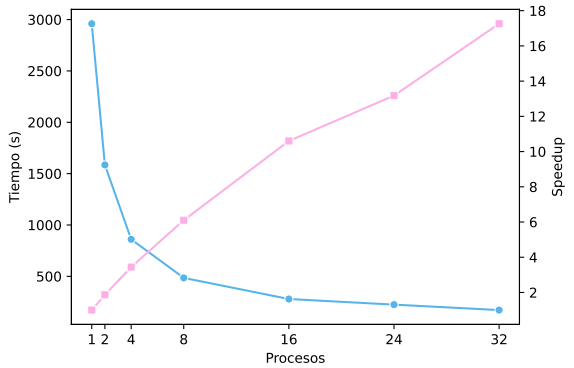


Fig. 2: Paralelización por diferenciación de curvas

VII-E. Análisis de la paralelización de los bucles internos del TSLLS

La paralelización de los bucles internos del TSLLS permite acelerar la extracción de parámetros de curvas individuales. La figura 1 muestra el esquema conceptual de esta paralización, donde el espacio de candidatos está ligado al número de puntos N que conforman la curva I-V. Valores pequeños de N limitan el impacto de la paralelización. Para este estudio se ha tomado un valor fijo de 500 puntos, igual que en el análisis anterior (Subsección VII-D).

Al aprovechar la propia anatomía del algoritmo, se analizan los tiempos promedios de la ejecución de cada una de las 50 curvas de 500 puntos (V, I) de la muestra selecionada. Con 2 procesos se obtiene un *speedup* de 1,682 y por tanto una eficiencia de 0,841 (Tabla II) demostrando así la utilidad de esta mejora. Sin embargo, 500 puntos no son suficientes para sacar el máximo provecho de esta paralelización como lo muestra la brusca bajada de la eficiencia a partir de 8 procesos. Casos en el que la sobrecarga de sincronización toma un papel más dominante en los tiempos de ejecución, siendo especialmente notable a partir de 32 procesos, con tiempos de ejecución en aumento.

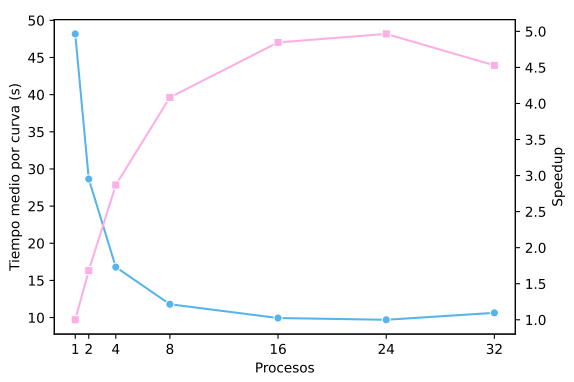


Fig. 3: Paralelización de los bucles internos del TSLLS

Tabla II: Resultados experimentales de la paralelización de los bucles internos del TSLLS sobre 50 curvas sintéticas de 500 puntos.

Workers	Tiempo medio (s)	Speedup	Eficiencia
1	48.175	1.000	1.000
2	28.649	1.682	0.841
4	16.792	2.869	0.717
8	11.795	4.085	0.511
16	9.940	4.847	0.3035
24	9.703	4.965	0.2077
32	10.638	4.529	0.1427

VIII. Conclusiones

Este trabajo ha analizado dos estrategias complementarias de paralelización aplicadas al procedimiento TSLLS para la extracción de parámetros en curvas I-V. La primera, basada en la diferenciación previa de curvas y en su distribución entre procesos, actúa sobre el problema a nivel macroscópico y persigue un reparto de carga más equilibrado. La segunda, centrada en la paralelización de los bucles internos del algoritmo, explota la independencia entre configuraciones candidatas durante la evaluación de una curva individual.

Los resultados experimentales muestran que la estrategia de diferenciación de curvas ofrece un comportamiento más robusto y escalable como solución general de procesamiento masivo. Su principal ventaja reside en que reduce de forma sostenida el tiempo total de ejecución, mantiene una eficiencia razonable incluso con un número elevado de procesos.

La paralelización de los bucles internos del TSLLS solo resulta aconsejable cuando cada curva contiene un número elevado de puntos. En curvas con pocos datos, la sobrecarga debida a la paralización reduce notablemente su eficacia. Por ello, este segundo enfoque no se recomienda como estrategia general y debe reservarse exclusivamente para curvas de alta densidad de datos.

Agradecimientos

F. Javier Toledo was supported by Grant PID2022-136399NB-C22 funded by MICIU/AEI/10.13039/501100011033. V. Galiano's work has received funding from the Valencian Ministry of Innovation, Universities, Science and Digital Society (Generalitat Valenciana) under PID2021-123627OB-C55 funded by MICIU/AEI/10.13039/501100011033, by the European Union NextGenerationEU/PRTR. The work of V. Herranz, X. Moreno-Vassart, and V. Galiano has also been supported by Grant PID2024-158682OB-C33 funded by MICIU/AEI /10.13039/501100011033 and by the European Union's FEDER.

Referencias

[1] W. De Soto, S. A. Klein, and W. A. Beckman, "Improvement and validation of a model for photovoltaic array performance," *Solar Energy*, vol. 80, no. 1, pp. 78–88, 2006.

[2] M. G. Villalva, J. R. Gazoli, and E. R. Filho, "Comprehensive approach to modeling and simulation of photovoltaic arrays," *IEEE Transactions on Power Electronics*, vol. 24, no. 5, pp. 1198–1208, 2009.

[3] J. Blanes, F. Toledo, S. Montero, and A. Garrigós, "In-site real-time photovoltaic i-v curves and maximum power point estimator," *IEEE Transactions on Power Electronics*, vol. 28, no. 3, pp. 1234–1240, 2013.

[4] J. Charles, M. Abdelkrim, Y. Muoy, and P. Mialhe, "A practical method of analysis of the current-voltage characteristics of solar cells," *Solar Cells*, vol. 4, pp. 169–178, 1981.

[5] D. Chan, J. R. Phillips, and J. C. H. Phang, "A comparative study of extraction methods for solar cell model parameters," *Solid-State Electronics*, vol. 29, no. 3, pp. 329–337, 1986.

[6] A. Laudani, F. Riganti-Fulginei, A. Salvini, and F. Mancilla-David, "Reduced-form of the photovoltaic five-parameter model for efficient computation of parameters," *Solar Energy*, vol. 97, pp. 122–127, 2013.

[7] A. Laudani, F. Riganti-Fulginei, and A. Salvini, "Identification of the one-diode model for photovoltaic modules from datasheet values," *Solar Energy*, vol. 108, pp. 432–446, 2014.

[8] F. J. Toledo, Jose M. Blanes, V. Galiano, and A. Laudani, "In-depth analysis of single-diode model parameters from manufacturer's datasheet," *Renewable Energy*, vol. 163, pp. 1370–1384, 2021.

[9] F. J. Toledo, J. M. Blanes, A. Garrigós, and J. A. Martínez, "Analytical resolution of the electrical four-parameters model of a photovoltaic module using small perturbation around the operating point," *Renewable Energy*, vol. 43, pp. 83–89, 2012.

[10] F. J. Toledo and J. M. Blanes, "Geometric properties of the single-diode photovoltaic model and a new very simple method for parameters extraction," *Renewable Energy*, vol. 72, pp. 125–133, 2014.

[11] M. F. AlHajri, K. M. El-Naggar, M. R. AlRashidi, and A. K. Al-Othman, "Optimal extraction of solar cell parameters using pattern search," *Renewable Energy*, vol. 44, pp. 238–245, 2012.

[12] A. Laudani, F. Riganti-Fulginei, and A. Salvini, "High performing extraction procedure for the one-diode model of a photovoltaic panel from experimental i-v curves by using reduced forms," *Solar Energy*, vol. 103, pp. 316–326, 2014.

[13] L. H. I. Lim, Z. Z. Ye, J. Ye, D. Yang, and H. Du, "A linear identification of diode models from single i-v characteristics of pv panels," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 7, pp. 4181–4193, 2015.

[14] A. A. Cárdenas, M. Carrasco, F. Mancilla-David, A. Street, and R. Cárdenas, "Experimental parameter extraction in the single-diode photovoltaic model via a reduced-space search," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 2, pp. 1468–1476, 2017.

[15] F. Javier Toledo, Jose M. Blanes, and Vicente Galiano, "Two-step linear least-squares method for photovoltaic single-diode model parameters extraction," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 8, pp. 6301–6310, 2018.

[16] B. Marion, A. Anderberg, C. Deline, J. del Cueto, M. Muller, G. Perrin, J. Rodriguez, S. Rummel, T. J. Silverman, F. Vignola, R. Kessler, J. Peterson, S. Barkaszi, M. Jacobs, N. Riedel, L. Pratt, and B. King, "New data set for validating pv module performance models," in *2014 IEEE 40th Photovoltaic Specialist Conference (PVSC)*, 2014, pp. 1362–1366.

[17] X. Moreno-Vassart, F. Javier Toledo, Victoria Herranz, and Vicente Galiano, "Relationships between remarkable points in photovoltaic i-v curves," *Renewable Energy*, vol. 237, pp. 121661, 2024.

[18] Fco. Javier Toledo, M. Victoria Herranz, José Manuel Blanes, and Vicente Galiano, "Quick and accurate strategy for calculating the solutions of the photovoltaic single-diode model equation," *IEEE Journal of Photovoltaics*, vol. 12, no. 2, pp. 493–500, 2022.

[19] F. Javier Toledo, Vicente Galiano, Victoria Herranz, and Jose M. Blanes, "Efficient computation of the photovoltaic single-diode model curve by means of a piecewise linear self-adaptive representation," *Journal of Computational Science*, vol. 75, pp. 102199, 2024.

[20] F. Javier Toledo, Vicente Galiano, Jose M. Blanes, Victoria Herranz, and Efstratios Batzelis, "Photovoltaic single-diode model parametrization. an application to the calculus of the euclidean distance to an i-v curve," *Mathematics and Computers in Simulation*, vol. 225, pp. 794–819, 2024.

[21] F. J. Toledo and J. M. Blanes, "Analytical and quasi-explicit four arbitrary point method for extraction of solar cell single-diode model parameters," *Renewable Energy*, vol. 92, pp. 346–356, 2016.

[22] M. Piliougine, R. A. Guejia-Burbano, G. Petrone, F. J. Sánchez-Pacheco, L. López, and M. Sidrach-de Cardona, "Parameters extraction of single diode model for degraded photovoltaic modules," *Renewable Energy*, vol. 164, pp. 674–686, 2021.

[23] Vicente Galiano, Jose M. Blanes, F. Javier Toledo, M. Victoria Herranz, and Antonino Laudani, "Pvmodel: An online photovoltaic modelling tool," in *2022 IEEE 21st Mediterranean Electrotechnical Conference (MELECON)*, 2022, pp. 693–697.

Extensión del simulador WorkflowSim para el procesamiento de flujos de trabajo con datos continuos

Anna Marín[1], Rosana Tomàs[2], Concepció Roig[2], Vitor da Silva[2], Fernando Guirado[2]

Resumen— **La ejecución de flujos de trabajo científicos (*workflows*) en infraestructuras de *cloud computing* se ha centrado históricamente en cargas finitas y aisladas. Sin embargo, muchas aplicaciones científicas —desde la física de partículas hasta la genómica— generan datos de forma continua y requieren un procesamiento sostenido que respete las dependencias entre tareas. Este trabajo presenta una extensión de WorkflowSim para simular ese escenario mediante tres modos de ejecución: no segmentado (*no pipelined*), segmentado (*pipelined*) y segmentado con asignación fija (*pipelined sticky*). La propuesta replica las tareas del workflow sobre múltiples instancias de datos e introduce dependencias entre ellas para modelar un flujo continuo sin modificar el núcleo del simulador. Los experimentos muestran que los modos segmentados reducen los tiempos de inactividad del sistema de procesamiento, mejoran la productividad y permiten analizar de forma más realista el efecto de la localidad de datos.**

Palabras clave— **Simulación, WorkflowSim, flujos de trabajo, flujo de datos continuo, cloud computing, pipeline, streaming.**

I. Introducción

El uso de *workflows* científicos se ha consolidado como un estándar para gestionar procesos complejos en entornos distribuidos. Tradicionalmente, estos se modelan como Grafos Dirigidos Acíclicos (DAG) cuyas tareas se asignan de forma estática a los procesadores del sistema, para luego realizar la ejecución del workflow según este mapeo [1]. Sin embargo, muchas aplicaciones reales, como las provenientes de la radioastronomía, la física de partículas o la genómica, trabajan con un flujo de datos de entrada ininterrumpido que requiere un procesamiento continuo en forma de *pipeline* [2], [3], [4].

En este contexto, la simulación es una herramienta indispensable para evaluar el rendimiento de estos sistemas sin incurrir en los elevados costes económicos y temporales de las infraestructuras *cloud* reales [5]. Aun así, simuladores de referencia actuales, como *WorkflowSim* [6], fueron diseñados originalmente para la ejecución de una carga de trabajo completa sobre un lote de datos único y finito. Esta limitación impide el análisis de sistemas donde la llegada de datos es continua, lo cual dificulta el estudio de métricas críticas como la productividad (*throughput*) o la estabilidad del sistema a largo plazo. Así, a pesar de la robustez que presenta *WorkflowSim*, se ha visto que su arquitectura original impone una rigidez estructural: la simulación finaliza obligatoriamente tras completar una única instancia del grafo. Esta limitación impide el estudio de sistemas con capacidad de procesar un flujo de datos continuo.

Para dotar a *WorkflowSim* de nuevas capacidades, se ha realizado una extensión del simulador que permite la simulación de flujos continuos de entrada de datos. El objetivo principal ha sido transformar un motor de ejecución estático en uno capaz de gestionar múltiples instancias de datos de entrada sobre un mismo DAG. Esta extensión no se limita a una mera repetición de ejecuciones, sino que se basa en la replicación estructurada de tareas, la incorporación de dependencias entre instancias y la extensión de los componentes de planificación y ejecución para mantener la continuidad del flujo, sin comprometer la precisión de los modelos de sobrecarga (*overhead*) ya existentes en la herramienta base.

Con esta ampliación, se pone a disposición una plataforma capaz de validar estrategias de planificación en escenarios de *streaming*, permitiendo una modelización mucho más cercana a los entornos de producción reales.

Las siguientes secciones se estructuran de la siguiente manera: la Sección II presenta el marco teórico y el estado del arte; la Sección III describe la extensión implementada y los modos de ejecución; la Sección IV detalla el marco experimental y los resultados; y la Sección V recoge las conclusiones y líneas de trabajo futuro.

II. Marco teórico y estado del arte

En esta sección se describe el funcionamiento de los flujos de trabajo científicos y la arquitectura del simulador base utilizado.

A. Aplicaciones de procesamiento continuo

Existen diversos dominios científicos y de ingeniería donde las aplicaciones de procesamiento continuo ya está en producción o en despliegue activo.

A1. Fuentes de luz sincrotrón e imagen de rayos X. Las instalaciones de radiación sincrotrón generan volúmenes de datos enormes de forma continua [7]. El European XFEL produce entre 10 y 15 GB/s por detector, y el LCLS-II superará los 20 GB/s. El *pipeline* de procesamiento incluye al menos tres etapas —preprocesamiento, reconstrucción tomográfica y postprocesamiento— con intensidad de cálculo muy variable. El marco PvaPy de Argonne National Laboratory procesa imágenes de ptico-

[1]Dpto. de Ingeniería Informática, Universitat de Lleida. Email: amn11@alumnes.udl.cat
[2]Dpto. de Ingeniería Informática, Universitat de Lleida. Emails: {rosana.tomas, concepcio.roig, vitor.dasilva, fernando.guirado}@udl.cat

grafía a 8 kHz distribuyendo el trabajo entre múltiples GPUs con latencias inferiores al segundo [8].

A2. Física de partículas y experimentos nucleares de alta energía. En el LHC del CERN [3], los detectores generan flujos masivos de datos que deben filtrarse y clasificarse en tiempo real. El experimento sPHENIX procesa colisiones a 3 MHz con un ancho de banda de lectura limitado a 15 kHz [9]. El reto consiste en identificar eventos físicos raros dentro de un volumen inmenso de colisiones de fondo; si las datos no se filtran dentro del intervalo entre colisiones, se pierden de forma irreversible.

A3. Radioastronomía. Los radiotelescopios producen datos a ritmos que superan el tráfico de Internet mundial [2]. El SKA1-mid generará 15,8 TB/s; el ASKAP digitaliza a 103 TB/s; el CHIME procesa 13,1 TB/s sin interrupción. Todas estas señales requieren correlaciones cruzadas entre pares de antenas en tiempo real y FFTs multidimensionales: cualquier retardo implica pérdida permanente de datos.

A4. Aprendizaje automático en tiempo real. Desplegar inferencia de ML sobre datos en flujo exige baja latencia, actualización continua de modelos para evitar la deriva de concepto, y coordinación entre datos en *streaming* y datos históricos.

A5. Visión por computador y procesamiento de vídeo. Procesar flujos de vídeo 4K a 30–60 FPS con redes neuronales profundas para detección, seguimiento o inspección de calidad representa un reto computacional considerable [10]. Los modelos como YOLO [11] ejecutan millones de operaciones por fotograma.

A6. Sismología y geofísica. Los sistemas de alerta sísmica dependen de procesar flujos de sensores en tiempo real: filtrado, FFT y correlación cruzada entre estaciones [12]. El marco SeisFLINK [13], construido sobre Apache Flink, procesa señales continuas de centenares de sensores distribuidos geográficamente.

A7. Genómica y bioinformática. Un secuenciador moderno genera aproximadamente 1 TB al día [4]. Los datos pasan por control de calidad, alineamiento contra el genoma de referencia e identificación de variantes. Arquitecturas basadas en Kafka [14] permiten encadenar estos módulos como servicios de *streaming*, sustituyendo los *pipelines* basados en ficheros que resultan insuficientes cuando la escala crece [15].

B. Flujos de trabajo

Un *workflow*, o flujo de trabajo, se modela como un Grafo Dirigido Acíclico (DAG), donde los nodos representan tareas computacionales y las aristas representan las dependencias de datos entre ellas. En el modelo tradicional de simulación, el sistema procesa una única instancia de este grafo: se empieza con las tareas iniciales, se ejecutan siguiendo la orden de precedencia y la simulación finaliza cuando la última tarea (nodo de salida) se ha completado, todo ello para la única instancia de entrada.

C. Arquitectura de WorkflowSim

WorkflowSim se define como un entorno de simulación construido sobre la base de *CloudSim* [6], [16] que incorpora una capa especializada para la gestión de flujos de trabajo científicos.

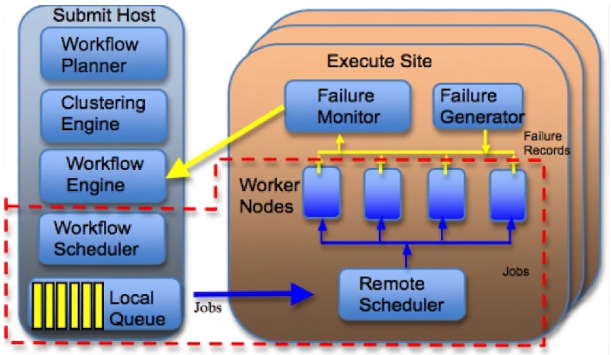


Fig. 1: Arquitectura original de WorkflowSim y sus componentes principales [6]

Su arquitectura interna se ilustra en la Figura 1, donde se pueden apreciar los componentes fundamentales que articulan de forma jerárquica el ciclo de vida de la ejecución, una vez introducido el modelo de la aplicación:

- **WorkflowParser:** Es el componente encargado de interpretar los archivos de descripción del grafo DAG de entrada (habitualmente en formato DAX). Su función principal es la traducción de los nodos del grafo en objetos de tareas asimilables por el motor de simulación, gestionando las dependencias lógicas de datos entre ellas.
- **WorkflowPlanner:** Actúa como el motor de planificación estática. Es responsable de determinar el mapeo global de recursos de forma previa a la fase de computación, asignando las tareas a los recursos de computación según la topología del grafo.
- **ClusteringEngine:** Implementa estrategias de agrupación de tareas para consolidar unidades de ejecución mayores (*jobs*). Este proceso es crítico para optimizar la eficiencia del sistema, ya que permite reducir significativamente la sobrecarga (*overhead*) de planificación y los tiempos de latencia asociados al envío de tareas pequeñas.
- **WorkflowEngine:** Coordina los *schedulers* y actúa como intermediario entre el motor de *clustering* y las VMs (Máquinas Virtuales). Mantiene las listas de trabajos enviados y recibidos, y gestiona la comunicación de eventos de finalización de tareas hacia el planificador.
- **WorkflowScheduler:** Representa el motor de ejecución dinámica. Gestiona en tiempo real la asignación de las unidades de ejecución a las VMs disponibles, asegurando que se respeten las restricciones de precedencia y disponibilidad de recursos.

D. Limitaciones para el procesamiento continuo

La base del simulador, *CloudSim*, implementa un modelo de simulación basado en una cola de eventos que se detiene cuando la cola queda vacía. El *WorkflowParser* realiza una lectura única del archivo de entrada al inicio, generando toda la carga de forma síncrona. Una vez completada la ejecución, no existe capacidad de aceptar nuevas entradas. Para soportar un flujo continuo, es necesario permitir múltiples instancias del mismo DAG en una sola simulación y definir explícitamente el orden de procesamiento de los datos. Para ello, en el presente trabajo se implementa una extensión de WorkflowSim, de tal forma que para cada dato de entrada se replica una instancia del DAG y se incorpora los métodos de sincronización necesarios para respetar el orden de procesamiento de los datos.

En el marco referencial actual, la mayoría de simuladores de *cloud* no permiten gestionar de forma integrada el retardo entre la llegada de datos consecutivos. Sin este control, métricas como la productividad (*throughput*) en régimen estacionario o los cuellos de botella derivados de la acumulación de carga no pueden evaluarse con precisión. En la extensión de WorkflowSim presentada en este trabajo, este retardo se modela como un desplazamiento temporal por dato en el momento de envío de los trabajos al datacenter.

III. Extensión de WorkflowSim

La modificación del simulador WorkflowSim se ha implementado como un paquete Java independiente (`org.dataflow`) que extiende las clases originales mediante herencia, sin alterar el código fuente base. Esto garantiza la compatibilidad con todas las funcionalidades existentes y facilita el mantenimiento futuro.

A. Arquitectura de la extensión

La estrategia de diseño combina los patrones *Template Method* y *Factory Method*: cada componente de WorkflowSim (Parser, Planner, Engine, Scheduler) dispone de una subclase `Dataflow*` que sobreescribe tanto el flujo de control (*Template Method*) como los métodos de instanciación (*Factory Method*) para incorporar los modos de ejecución extendidos. De este modo, la cadena completa de procesamiento se redirige hacia la lógica de flujo continuo de forma transparente, sin necesidad de modificar las clases originales.

La Figura 2 muestra el diagrama de clases de la extensión con sus relaciones de herencia respecto a las clases originales de WorkflowSim. A continuación se describen las nuevas clases incorporadas:

- **DataflowTask/DataflowJob**: extienden `Task` y `Job` con dos atributos clave: `dataInstanceId` (identifica la instancia de datos, p. ej. $D^{(0)}, D^{(1)}, \ldots$) y `templateTaskId` (identificador de la tarea en el DAG plantilla). Estos campos permiten en todo momento saber qué tarea del workflow original se está ejecutando y para qué instancia de datos.
- **DataflowParser**: lee el fichero DAX una sola vez y, para cada instancia de datos, replica el DAG completo con identificadores únicos, manteniendo la estructura de dependencias intacta dentro de cada instancia. Además, crea dependencias *inter-instancia* entre instancias consecutivas cuyo tipo varía según el modo de ejecución (véase la Sección III-C).
- **DataflowPlanner**: sobreescribe el planificador original para seleccionar el algoritmo de planificación y coordinar la replicación de instancias.
- **DataflowScheduler**: extiende el *scheduler* con soporte para asignación predefinida (respetando las decisiones del planificador) y modo de asignación fija (*sticky*).
- **DataflowClusteringEngine**: sobreescribe los *factory methods* del motor de *clustering* original para instanciar los componentes `Dataflow*` en lugar de los originales.
- **DataflowParameters**: configuración estática global que centraliza el modo de ejecución (no pipelined, pipelined, pipelined sticky), el número de instancias, el algoritmo de planificación y todos los parámetros configurables de la simulación.

B. Modos de ejecución del simulador

La adaptación de WorkflowSim para simular aplicaciones con flujo continuo de datos se ha realizado integrando tres modos distintos de ejecución, según el tratamiento temporal de los datos de entrada que se realiza durante la simulación.

A continuación se describen los tres modos de ejecución, denominados **no pipelined**, **pipelined** y **pipelined sticky**.

Para formalizar la notación, se denota como $D^{(i)}$ a la instancia número i del DAG y como $T_j^{(i)}$ a la tarea T_j para la instancia k del DAG.

B.1 Modo 1: no pipelined

Es el modo más básico. Todas las instancias del DAG se generan al inicio y se procesan una tras otra: la instancia $D^{(k+1)}$ no empieza hasta que $D^{(k)}$ ha completado todas sus tareas. El mapping de tareas de cada instancia del DAG es independiente. El *makespan* crece casi linealmente con el número de instancias N ($makespan \approx N \times makespan_1$), ya que las VMs permanecen inactivas entre instancias consecutivas. Este modo de ejecución permite la simulación de una entrada continua de datos, pero no existe solapamiento entre instancias.

B.2 Modo 2: pipelined

Se introduce el solapamiento de instancias de los DAGs. Cuando una tarea $T_i^{(k)}$ finaliza en una VM, la tarea $T_j^{(k+1)}$ puede empezar inmediatamente si sus dependencias están satisfechas, aunque la instancia $D^{(k)}$ no haya concluido. Las VMs permanecen ocupadas prácticamente todo el tiempo, acercando la pro-

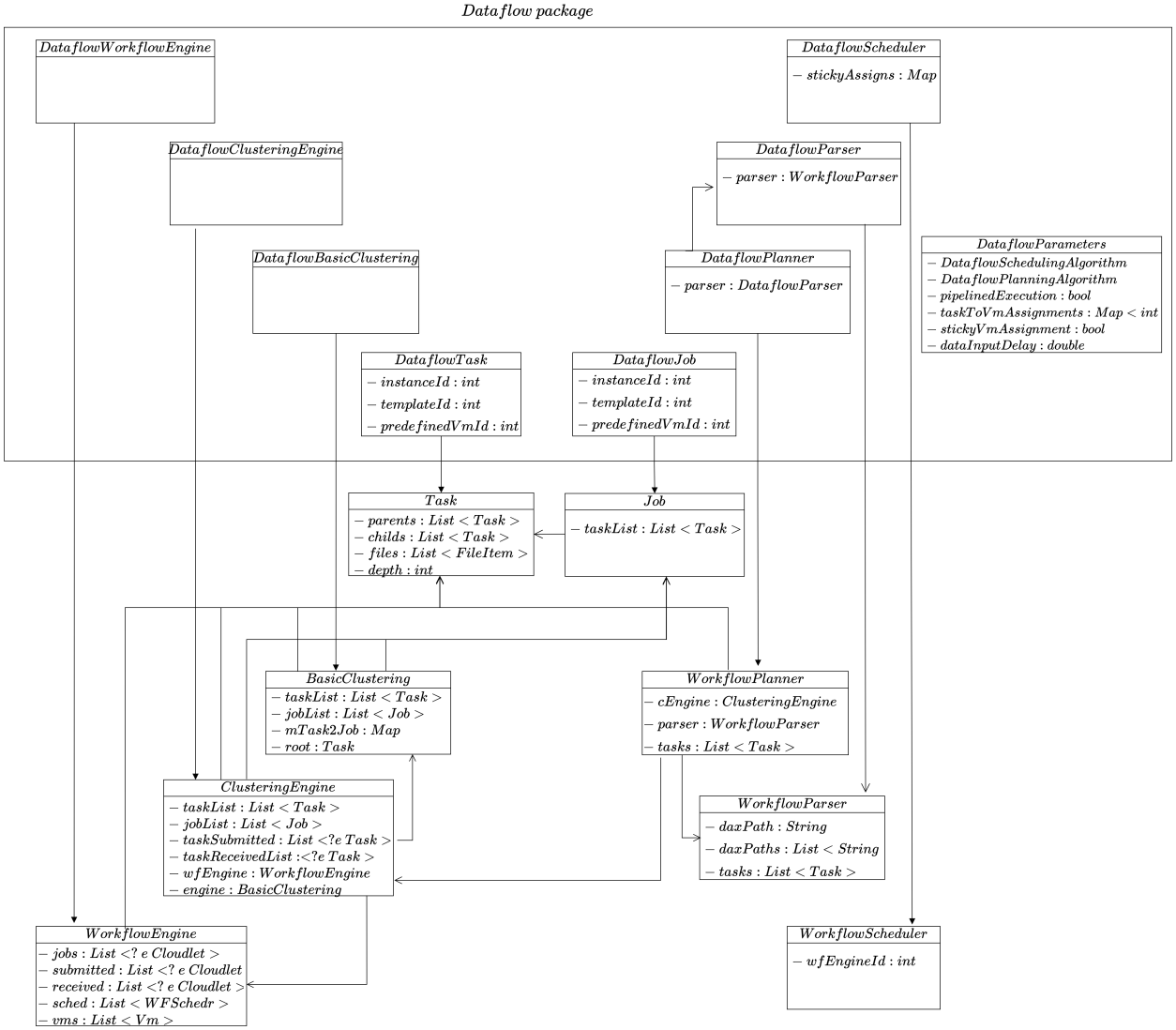


Fig. 2: Diagrama de clases de la extensión `org.dataflow`.

ductividad al límite teórico impuesto por la etapa más lenta del DAG.

El inconveniente de este modo es que el *scheduler* puede asignar tareas del mismo tipo (*template*) a VMs distintas en cada instancia, lo que genera transferencias de datos innecesarias entre VMs y pierde la localidad de datos.

B.3 Modo 3: pipelined sticky

El tercer modo añade la asignación fija (*sticky*): la primera instancia ($D^{(0)}$) se planifica normalmente (por el *scheduler* o por el planificador estático, si se utiliza uno); todas las instancias posteriores ($D^{(1)}, D^{(2)}, \ldots$) heredan automáticamente la misma asignación tarea–VM. El *scheduler* busca, para cada tarea de una instancia $D^{(k)}$, el `templateTaskId` correspondiente a $D^{(0)}$ y le asigna la misma VM.

El resultado es un *pipeline* con una asignación fija, donde cada VM se encarga siempre de las mismas etapas del DAG, eliminando transferencias redundantes y manteniendo la coherencia del pipeline entre instancias.

C. Modelo de dependencias inter-instancia

La clave que diferencia los modos es cómo el `DataflowParser` conecta las instancias entre sí mediante **dependencias inter-instancia**. El parser lee el fichero DAX una sola vez para obtener el DAG plantilla ($D^{(0)}$, con tareas $T_1, T_2, \ldots, T_m$) y lo replica $N-1$ veces. Cada réplica $D^{(k)}$ mantiene la estructura interna intacta, pero la forma de conectar instancias consecutivas varía:

no pipelined (modo 1): todas las tareas hoja de $D^{(k-1)}$ se conectan con todas las tareas raíz de $D^{(k)}$. Esto garantiza que ninguna tarea de $D^{(k)}$ pueda empezar hasta que $D^{(k-1)}$ haya finalizado completamente, forzando una ejecución estrictamente secuencial entre instancias.

pipelined y pipelined sticky: se crean dependencias *uno a uno*: cada tarea $T_i^{(k)}$ depende de $T_i^{(k-1)}$ (la misma tarea plantilla de la instancia anterior). De este modo, la tarea $T_j^{(1)}$ puede empezar en cuanto $T_j^{(0)}$ finalice, aunque otras tareas de $D^{(0)}$ sigan en ejecución. Esto permite la superposición de etapas y genera el efecto de *pipeline* visible en los diagramas de Gantt.

La diferencia entre los modos de ejecución pipelined y pipelined sticky no reside en las dependencias inter-instancia (que son idénticas), sino en la asignación de VMs: en el modo pipelined sticky, cada tarea T_i se asigna siempre a la misma VM para todas las instancias.

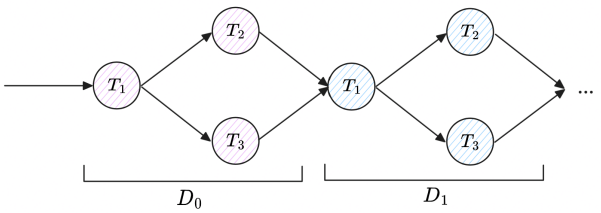


Fig. 3: Modelo de dependencias inter-instancia: no pipelined. Todas las hojas de $D^{(k-1)}$ conectan con todas las raíces de $D^{(k)}$.

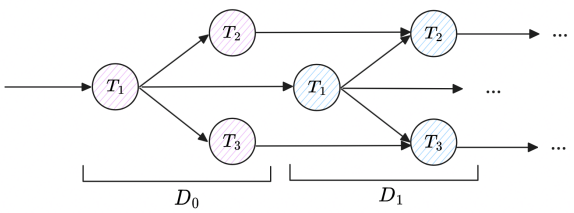


Fig. 4: Modelo de dependencias inter-instancia: pipelined. Cada tarea $T_i^{(k)}$ depende solo de $T_i^{(k-1)}$.

En las Figuras 3 y 4 se ilustran las dependencias generadas para un DAG formado por tres tareas (T_1, T_2, T_3) para dos instancias de entrada $(D^{(0)}, D^{(1)})$ según los modos de ejecución Figura 3 no pipelined y Figura 4 pipelined y pipelined sticky. Se puede apreciar cómo en el modo no pipelined el DAG de la instancia $D^{(1)}$ no empieza hasta haber finalizado por completo la ejecución de la instancia $D^{(0)}$. En cambio, en los modos pipelined, las precedencias de cada tarea vienen dadas por las propias del DAG y de las instancias: por ejemplo, la tarea $T_1^{(1)}$ puede empezar en cuanto finaliza $T_1^{(0)}$, sin esperar a la finalización completa de $D^{(0)}$.

D. Parámetros configurables

Para evitar la necesidad de modificar código cada vez que se desea cambiar un experimento, toda la configuración se pasa por línea de comandos. Los parámetros principales incluyen: el fichero DAX de entrada, el número de VMs, el número de instancias de datos, el modo de ejecución (pipelined / no pipelined), la asignación *sticky*, el algoritmo de *scheduling*, el algoritmo de planificación y el retardo de llegada entre instancias (`dataInputDelay`). Este último se aplica como un desplazamiento temporal por instancia al enviar los trabajos al datacenter.

IV. Marco de experimentación

En esta sección se ilustra el funcionamiento del simulador WorkflowSim con los tres modos de ejecución descritos en la Sección III. El objetivo experimental es doble: (i) cuantificar el impacto de cada modo de ejecución sobre el *makespan* y (ii) verificar visualmente el efecto de solapamiento y de asignación fija tarea–VM. Al final de la sección se muestra qué comportamientos aparecen en cada modo y en qué condiciones el modo *pipelined sticky* aporta beneficio adicional.

Para la simulación se ha utilizado como caso de estudio el workflow Montage_25 del proyecto Pegasus [1], cuyo DAG de tareas se ilustra en la Figura 5. Como se puede observar, es un DAG con 25 tareas y topología fan-out/fan-in, donde las tareas ID00, ID01, ID02, ID03 e ID04 son tareas raíz (sin predecesores) y la tarea ID24 es tarea hoja (sin sucesores).

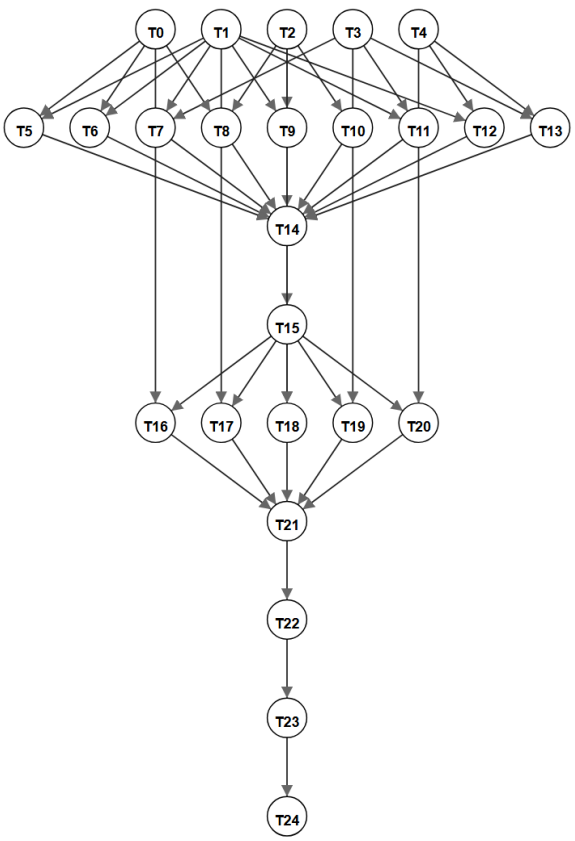


Fig. 5: DAG del workflow Montage_25.

Los experimentos se realizan sobre el simulador extendido descrito en la Sección III, utilizando Montage_25 como caso de estudio principal, con 3 VMs homogéneas (1 PE, 1 000 MIPS, 512 MB RAM) y 5 instancias de datos consecutivas. El algoritmo de *scheduling* utilizado es FCFS (*First Come First Served*) en todas las configuraciones, para aislar el efecto del modo de ejecución sin que el algoritmo de asignación influya en los resultados. La simulación es determinista, lo que garantiza reproducibilidad total. En la Tabla I se resumen los parámetros predefinidos de los experimentos.

Tabla I: Parámetros predefinidos de los experimentos.

Parámetro	Valor
Workflow	Montage_25 (Pegasus)
Número de tareas	25
Topología	fan-out/fan-in
Número de VMs	3
VM (por defecto)	1 PE, 1 000 MIPS, 512 MB RAM
Instancias de datos	5
Scheduler	FCFS
Modos evaluados	no pipelined, pipelined, pipelined sticky
Retardo entre llegadas	`dataInputDelay`=20 s (cuando aplica)
Transfer rate (TR)	BW=1000, es decir, ancho de banda simulado entre VMs
Escenario sin TR	NOTR (TR desactivado)

A. Comparación de los tres modos de ejecución

Las Figuras 6, 7 y 8 muestran los diagramas de Gantt correspondientes a los tres modos de ejecución del simulador.

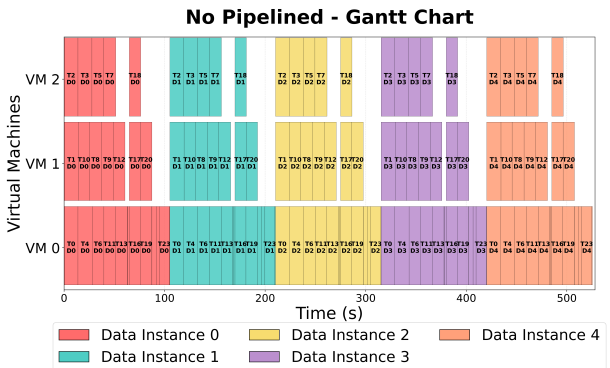


Fig. 6: no pipelined: las instancias se procesan secuencialmente. Las VMs quedan inactivas entre instancias.

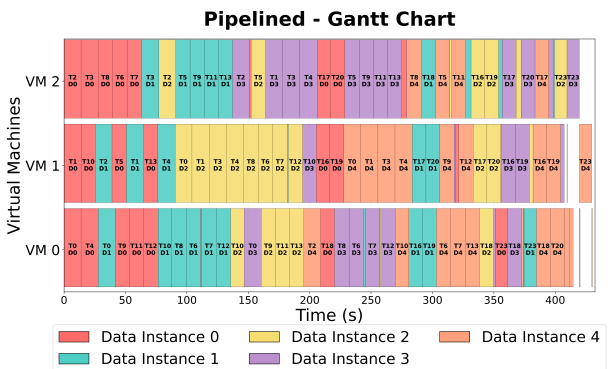


Fig. 7: pipelined: las instancias se solapan, las VMs permanecen ocupadas casi todo el tiempo.

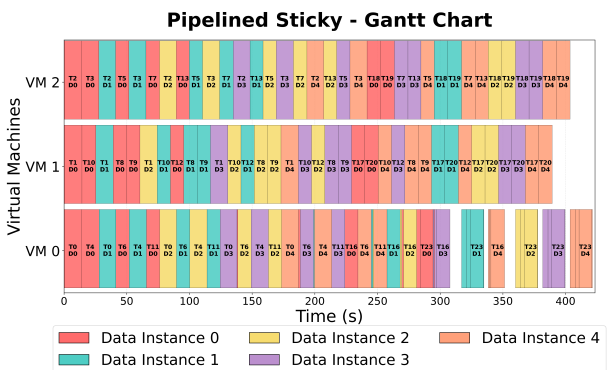


Fig. 8: pipelined sticky: solapamiento con asignación fija tarea–VM.

En el modo **no pipelined** (Figura 6), se observa claramente cómo las VMs quedan inactivas entre instancias consecutivas: la instancia $D^{(k+1)}$ no empieza hasta que todas las tareas de $D^{(k)}$ han finalizado. Esto genera un patrón escalonado con huecos de inactividad significativos. El *makespan* crece de forma casi lineal con el número de instancias.

La ejecución **pipelined** (Figura 7) elimina estos huecos: las tareas de instancias distintas se ejecutan simultáneamente en cuanto sus dependencias están satisfechas. Las VMs permanecen ocupadas prácticamente todo el tiempo, lo que se traduce en una reducción drástica del *makespan* y un aumento significativo del *throughput*. Sin embargo, se puede observar que las tareas del mismo tipo no siempre se ejecutan en la misma VM entre instancias distintas, generando transferencias de datos innecesarias.

El modo **pipelined sticky** (Figura 8) mantiene el solapamiento pero fuerza que cada tarea se ejecute siempre en la misma VM que en $D^{(0)}$. El resultado es un patrón de *pipeline* estable y regular, donde cada VM procesa las mismas tareas en todas las instancias.

B. Resultados cuantitativos

La Tabla II resume las métricas principales obtenidas para los tres modos de ejecución con la configuración descrita.

Tabla II: Resultados para Montage_25 con 5 instancias, 3 VMs y FCFS.

Métrica	no pipelined	pipelined	p. sticky
Makespan (s)	525,43	430,21	421,40
Reducción	—	18,1 %	19,8 %

El modo pipelined reduce el *makespan* un **18,1 %** respecto al modo no pipelined (de 525,43 s a 430,21 s), al eliminar los huecos de inactividad entre instancias. El modo pipelined sticky mejora ligeramente este resultado hasta un **19,8 %** de reducción (421,40 s), al evitar transferencias de datos redundantes entre VMs gracias a la asignación fija.

B.1 Efecto de los costes de transferencia

Para evaluar la influencia de los costes de transferencia, se repiten los experimentos desactivando el límite de ancho de banda entre VMs (configuración NOTR: *No Transfer Rate*). En los experimentos con transferencia habilitada se utiliza **TR = BW 1000**, donde BW representa el ancho de banda simulado del enlace entre VMs en la configuración por defecto del modelo. En NOTR, el modo pipelined sin asignación fija obtiene un *makespan* de 384,58 s, mientras que el modo pipelined sticky asciende a 411,74 s. Este resultado es coherente: sin costes de transferencia, la rigidez de la asignación fija genera desequilibrios de carga que penalizan el rendimiento, mientras que el modo pipelined puede redistribuir tareas libremente.

B.2 Efecto del retardo entre llegadas

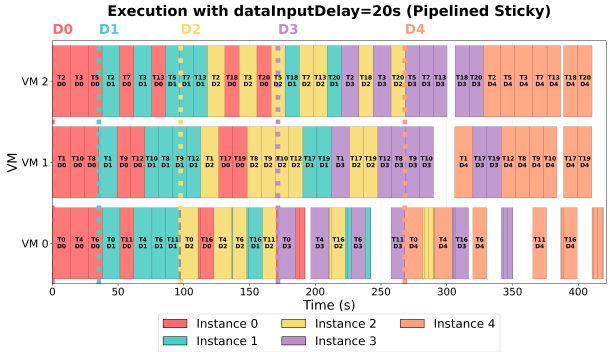


Fig. 9: Diagrama de Gantt para Montage_25 (5 instancias, 3 VMs, pipelined sticky, `dataInputDelay=20 s`). Las líneas punteadas marcan la inyección de cada instancia.

La Figura 9 muestra una ejecución con `dataInputDelay=20 s`. Se observa cómo el inicio de cada instancia $(D^{(0)}, D^{(1)}, \ldots)$ queda desplazado progresivamente, validando que el periodo de los datos se aplica en el envío de trabajos al datacenter.

B.3 Verificación del modo Sticky

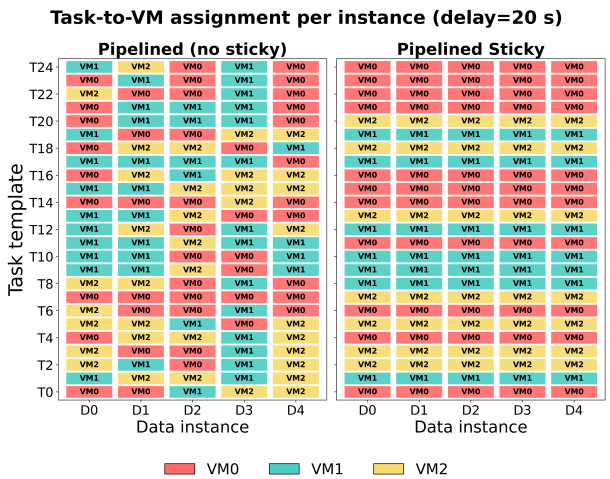


Fig. 10: Asignación tarea–VM por instancia de datos: pipelined (izquierda) frente a pipelined sticky (derecha). En modo pipelined sticky, cada tarea se enruta siempre a la misma VM en todas las instancias.

La Figura 10 resume visualmente la diferencia entre ejecución segmentada con y sin asignación fija usando dos ejecuciones exportadas por el simulador (CSV de resultados). En modo pipelined sticky, la asignación tarea–VM de la primera instancia se mantiene para las instancias posteriores, mientras que en modo pipelined puede variar entre instancias.

En conjunto, este resultado confirma que el modo *pipelined sticky* aporta en la localidad de datos y regularidad de ejecución. A la vez, también evidencia que su ventaja depende del coste de transferencia entre VMs: cuando dicho coste existe, sticky mejora el rendimiento, cuando se anula (NOTR), la flexibilidad de *pipelined* puede resultar más eficiente.

V. Conclusiones y trabajo futuro

Este trabajo presenta una extensión del simulador WorkflowSim que permite la ejecución de workflows con entrada continua de datos. Esto permite el análisis de la ejecución en entornos de cómputo cloud de workflows de diversas campos científicos, cuyo comportamiento se modela com un Grafo Dirigido Acíclico (DAG), que responden a este patrón de entrada continua.

La extensión implementada transforma el motor estático original de WorkflowSim en uno capaz de gestionar múltiples instancias del mismo DAG, sin modificar el código base del simulador. Con respecto al tratamiento de las diversas instancias del flujo continuo de entrada, se han desarrollado tres modos de ejecución: (a) no pipelined, en que se ejecuta todo el DAG para una instancia de datos antes de que pueda empezar la siguiente, (b) pipelined, que introduce el solapamiento entre instancias, de forma que una tarea puede empezar su ejecución cuando la instancia previa de la misma tarea ha finalizado y, (c) pipelined sticky, que actúa como el modo pipelined, pero fijando la asignación de tareas a procesadores del DAG original para todas las instancias de datos. El funcionamiento de los nuevos modos de ejecución integrados en WorkflowSim, se han ilustrado a través de la simulación del workflow Montage_25 del proyecto Pegasus, para diversas instancias de entrada, como caso de estudio. Dicho workflow se modela mediante un DAG de 25 tareas con topología fan-out/fan-in. En el análisis de los distintos modos de ejecución implementados, se destacan las siguientes características:

- El modo no pipelined reproduce fielmente el comportamiento de un planificador *batch* tradicional, con huecos de inactividad entre instancias y crecimiento lineal del *makespan*.
- El modo pipelined elimina estos huecos permitiendo la ejecución simultánea de tareas de diferentes instancias, reduciendo significativamente el *makespan* y aumentando el *throughput*.
- El modo pipelined sticky añade la garantía de localidad de datos al mantener cada tarea en la misma VM a lo largo de todas las instancias, generando un patrón de *pipeline* regular especialmente beneficioso cuando los costes de transferencia son elevados.

La extensión implementada es completamente compatible con todos los algoritmos de planificación y *scheduling* disponibles en WorkflowSim, lo que la convierte en una herramienta versátil para la investigación en planificación de flujos de trabajo con datos continuos.

Como líneas de trabajo futuro directamente relacionadas con la extensión, se plantean: (i) la incorporación de modelos de VMs heterogéneas y topologías de red en el modelo de pipeline para estudiar el impacto en la localidad de datos, (ii) la simulación de fallos y mecanismos de recuperación de instancias afectadas durante la ejecución en modo *streaming*, y (iii) la implementación de métricas avanzadas de *throughput* en régimen estacionario y detección automática de cuellos de botella en el pipeline.

Agradecimientos

El presente trabajo ha sido realizado en el marco del proyecto PID2021-122580NB-I00, financiado por el Ministerio de Ciencia e Innovación

Referencias

[1] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J. Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, and Kent Wenger, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[2] Peter E. Dewdney, Peter J. Hall, Richard T. Schilizzi, and T. Joseph W. Lazio, "The Square Kilometre Array," *Proceedings of the IEEE*, vol. 97, no. 8, pp. 1482–1496, 2009.

[3] Lyndon Evans and Philip Bryant, "LHC machine," *Journal of Instrumentation*, vol. 3, no. 08, pp. S08001, 2008.

[4] Sara Goodwin, John D. McPherson, and W. Richard McCombie, "Coming of age: ten years of next-generation sequencing technologies," *Nature Reviews Genetics*, vol. 17, no. 6, pp. 333–351, 2016.

[5] Rajkumar Buyya, Christian Vecchiola, and S. Thamarai Selvi, *Mastering Cloud Computing: Foundations and Applications Programming*, Morgan Kaufmann, 2013.

[6] Weiwei Chen and Ewa Deelman, "WorkflowSim: A toolkit for simulating scientific workflows in distributed environments," *Proc. IEEE 8th Int. Conf. E-Science*, pp. 1–8, 2012.

[7] Thomas Tschentscher, Christian Bressler, Jan Grünert, Anders Madsen, Adrian P. Mancuso, Michael Meyer, Andreas Scherz, Harald Sinn, and Ulf Zastrau, "Photon beam transport and scientific instruments at the European XFEL," *Applied Sciences*, vol. 7, no. 6, pp. 592, 2017.

[8] Sinisa Veseli, Nicholas Schwarz, Daniel Jarosz, and Daniel B. Allan, "Real-time streaming data reconstruction and processing at synchrotron light sources using the PvaPy framework," *Journal of Synchrotron Radiation*, vol. 30, pp. 1056–1066, 2023.

[9] Dung Dao et al., "Real-time AI-driven streaming readout for sPHENIX tracking detectors using FPGA-based GNN," in *Proceedings of the 23rd International Conference on Real Time Conference (RT2023)*, 2023.

[10] NVIDIA, "DeepStream SDK developer guide," 2020.

[11] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, "You only look once: Unified, real-time object detection," pp. 779–788, 2016.

[12] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas, "Apache Flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[13] Mahdi Sarrafzadeh et al., "SeisFLINK: A scalable real-time seismic waveform processing framework based on Apache Flink," in *IEEE International Conference on Big Data*, 2023.

[14] Jay Kreps, Neha Narkhede, and Jun Rao, "Kafka: A distributed messaging system for log processing," pp. 1–7, 2011.

[15] Hamid Mushtaq, Zaid Al-Ars, and Peter Bertels, "Streaming distributed DNA sequence alignment using Apache Kafka and Apache Flink," *bioRxiv*, 2017.

[16] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

Plataforma de Validación de Precisión Adaptativa para Simulaciones Lattice Boltzmann con Política Automática de Asignación de Precisión

Marcos Ramos Montes y José Flich Cardo[1]

Resumen— **Este trabajo presenta una plataforma de validación de precisión adaptativa para simulaciones Lattice Boltzmann D2Q9 con operador BGK. El interés de este tipo de técnicas radica en que la reducción controlada de precisión puede disminuir el coste computacional, los accesos a memoria y el consumo energético sin comprometer la fidelidad física de la simulación. La plataforma combina una referencia multiprecisión basada en la biblioteca MPFR (*Multiple Precision Floating-Point Reliable*) con una representación dual HIGH/LOW por celda y soporta dos políticas de control: una política manual basada en umbrales físicos y una política automática basada en un índice de actividad.**

En el modo automático, el sistema calcula el gradiente de velocidad, la vorticidad, la tasa de deformación y la distancia a obstáculos, normaliza estas magnitudes mediante cuantiles del propio flujo y construye un índice de actividad para cada celda. A partir de dicho índice, se identifica de forma preliminar la fracción objetivo de celdas menos activas que pueden operar en baja precisión. En la presente implementación CPU, esta selección se materializa mediante la determinación de un umbral asociado a la fracción objetivo, obtenido a partir de una clasificación global del índice. La asignación final se corrige posteriormente mediante restricciones físicas y mediante un mecanismo de validación numérica (*precision gate*) que rechaza conversiones no aceptables.

La evaluación se realiza en dos escenarios representativos: flujo alrededor de un cilindro y cilindros en tándem. Los resultados muestran que la política automática permite alcanzar valores próximos al 50 % de celdas en baja precisión, manteniendo prácticamente invariante el número de Strouhal y conservando errores relativos L_2 acotados en velocidad y densidad.

Palabras clave— **Lattice Boltzmann, precisión adaptativa, multiprecisión, MPFR, validación física.**

I. Introducción

El método Lattice Boltzmann (LBM) constituye una alternativa eficaz a los enfoques tradicionales de dinámica de fluidos computacional por su formulación mesoscópica, su regularidad algorítmica y su elevada capacidad de paralelización [1]. Estas propiedades hacen de LBM un candidato natural para arquitecturas de altas prestaciones, como GPUs y FPGAs, donde la localidad de datos y la simplicidad operativa pueden explotarse eficientemente.

En este contexto, la precisión numérica se convierte en un factor determinante. El uso uniforme de alta precisión mejora la robustez, pero incrementa el coste computacional, los accesos a memoria y, potencialmente, el consumo energético del sistema. Por el contrario, la reducción indiscriminada de precisión puede comprometer la fidelidad física de la simulación. Ello motiva el estudio de esquemas de precisión heterogénea y mixta, ampliamente analizados en computación científica como una vía para reducir coste manteniendo exactitud [2], [3].

En el caso de LBM, trabajos recientes han mostrado que configuraciones de precisión reducida pueden conservar adecuadamente la dinámica del flujo bajo determinadas condiciones [4], [5]. Sin embargo, estas aproximaciones suelen aplicar formatos numéricos de forma uniforme o estática, sin explotar la variabilidad espacial de la dinámica local. De forma complementaria, la literatura sobre HPC en LBM ha mostrado el interés de formatos reducidos y personalizados en GPUs y FPGAs, tanto en implementaciones generales como especializadas [6], [7], [8], [9], [10].

El problema metodológico de los esquemas adaptativos simples reside en que la asignación de precisión suele depender de umbrales empíricos ajustados caso a caso. Aunque esa aproximación puede resultar útil desde el punto de vista experimental, dificulta la defensa metodológica del mecanismo y reduce su capacidad de generalización. En este trabajo se aborda precisamente este punto proponiendo una política automática de asignación de precisión que sustituye el ajuste manual de umbrales absolutos por una decisión guiada por métricas físicas locales y una fracción objetivo de celdas en baja precisión.

La contribución principal de este trabajo es doble. En primer lugar, se propone un sistema de precisión adaptativa por celda guiado por indicadores físicos locales, sustentado en un índice de actividad y complementado con un mecanismo de validación numérica explícita. En segundo lugar, se evalúa experimentalmente dicho sistema en dos escenarios representativos, comparando configuraciones manuales y automáticas mediante fracción de celdas en baja precisión, estabilidad del mecanismo de control, número de Strouhal y errores relativos L_2 de velocidad y densidad. El objetivo es evaluar rigurosamente el impacto físico de la política adaptativa propuesta, manteniendo una fracción significativa de celdas operando en baja precisión.

Los resultados muestran que la política automática alcanza un uso efectivo de baja precisión has-

[1]Departamento de Informática de Sistemas y Computadores (DISCA), Universitat Politècnica de València, Valencia, España. e-mail: marramon@upv.es, jflich@disca.upv.es.

ta valores próximos al 50 % del dominio, manteniendo prácticamente invariante el número de Strouhal y conservando errores relativos L2 acotados en velocidad y densidad. Este resultado confirma el potencial que tienen las técnicas de selección dinámica de precisión en modelos LBM. En este trabajo nos centramos exclusivamente en la presentación de la herramienta que nos valida este hecho.

El resto del artículo se organiza de la siguiente forma. La Sección II revisa el trabajo relacionado. La Sección III describe el modelo numérico y la política de precisión adaptativa. La Sección IV presenta la metodología experimental. La Sección V recoge los resultados obtenidos y su interpretación física. Finalmente, la Sección VI resume las conclusiones y líneas futuras.

II. Trabajo relacionado

A. Precisión reducida y mixta en computación científica

La precisión mixta y la reducción controlada de precisión se han consolidado como estrategias clave para mejorar rendimiento y eficiencia energética en computación científica. Higham y Mary [2] analizan los fundamentos numéricos de estos enfoques y describen cómo combinar formatos distintos manteniendo estabilidad y control del error. En una perspectiva más reciente, Kashi et al. [3] subrayan el interés de diseñar políticas de precisión sensibles al problema físico y al contexto computacional.

B. Precisión numérica en LBM

El LBM presenta una estructura particularmente favorable para la exploración de formatos numéricos reducidos [1]. Lehmann et al. [4] estudian el impacto de formatos de 64, 32 y 16 bits y muestran que, bajo determinadas condiciones, es posible preservar la fidelidad física. En la misma línea, Tiribocchi et al. [5] exploran formulaciones más ligeras orientadas a reducir el coste manteniendo soluciones físicamente coherentes. La utilización de MPFR [11] como referencia de alta precisión resulta especialmente útil para cuantificar con rigor el efecto de la reducción de precisión.

A diferencia de estos trabajos, aquí el objetivo no es únicamente estudiar el impacto de formatos reducidos, sino construir una plataforma de validación que combine asignación adaptativa de precisión, validación numérica explícita y validación física sistemática.

C. LBM y arquitecturas HPC

Las implementaciones de LBM sobre GPUs y FPGAs han demostrado una gran capacidad para explotar paralelismo masivo y formatos personalizados [6], [7], [8], [9], [10]. En ese contexto, disponer de una política de precisión por celda físicamente interpretable constituye una base relevante para futuras integraciones sobre arquitecturas heterogéneas.

III. Modelo numérico y política de precisión adaptativa

A. Modelo Lattice Boltzmann

La plataforma de validación se basa en el método Lattice Boltzmann bidimensional D2Q9 con operador BGK [1]. El estado del sistema se describe mediante funciones de distribución f_i asociadas a un conjunto discreto de velocidades $\mathbf{c_i}$. A partir de ellas se obtienen las magnitudes macroscópicas de densidad y velocidad:

$$\rho = \sum_i f_i \tag{1}$$

$$\mathbf{u} = \frac{\mathbf{1}}{\rho} \sum_{\mathbf{i}} \mathbf{f_i}\, \mathbf{c_i} \tag{2}$$

Cada iteración se compone de colisión y propagación. La colisión se modela mediante:

$$f_i^{new} = f_i - \omega\,(f_i - f_i^{eq}) \tag{3}$$

donde f_i^{eq} es la distribución de equilibrio y ω controla la relajación hacia el equilibrio local. A continuación, en la fase de propagación, las poblaciones se desplazan a los nodos vecinos de acuerdo con las velocidades discretas del modelo:

$$f_i(\mathbf{x} + \mathbf{c_i}, \mathbf{t} + \mathbf{1}) = \mathbf{f_i^{new}}(\mathbf{x}, \mathbf{t}) \tag{4}$$

B. Política de precisión adaptativa

La plataforma implementa un sistema dual HIGH/LOW por celda. Ambos niveles de precisión se basan en aritmética multiprecisión mediante MPFR y se diferencian únicamente en la precisión de mantisa configurada. En las campañas consideradas en este trabajo, la configuración LOW emplea 16 bits de precisión de mantisa y la configuración HIGH 64 bits de precisión de mantisa. A diferencia de los formatos IEEE-754 clásicos, MPFR no se describe mediante un reparto fijo de bits de signo, exponente y mantisa, sino mediante una precisión configurable con la misma semántica de coma flotante y redondeo correcto.

Sobre esta base se distinguen dos modos de operación:

- **manual**, que reproduce el comportamiento clásico basado en umbrales físicos definidos por el usuario;
- **auto**, que calcula internamente la clasificación HIGH/LOW a partir de indicadores físicos, un índice de actividad y una fracción objetivo de celdas en baja precisión.

En ambos modos, la plataforma calcula para cada celda fluida, entendida como cada nodo del dominio que participa en la evolución del flujo y no pertenece a la geometría sólida ni a un obstáculo, un conjunto de indicadores físicos locales. En particular, se define el módulo del gradiente de velocidad

$$g(c) = \|\nabla \mathbf{u}(\mathbf{c})\|, \tag{5}$$

la vorticidad

$$\omega(c) = \left| \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y} \right|, \tag{6}$$

la tasa de deformación

$$\sigma(c) = \|S(c)\|, \tag{7}$$

donde

$$S(c) = \frac{1}{2} \left(\nabla \mathbf{u}(\mathbf{c}) + \nabla \mathbf{u}(\mathbf{c})^{\mathbf{T}} \right) \tag{8}$$

es el tensor simétrico de deformación, y la distancia discreta a obstáculos $d(c)$ calculada como la distancia, en número de celdas de la malla, a la frontera sólida más cercana.

En el modo **manual**, estas magnitudes se escalan mediante umbrales cuya determinación depende del fichero de configuración. En este caso, el usuario fija explícitamente la sensibilidad del criterio adaptativo para cada magnitud física considerada.

En el modo **auto**, las magnitudes físicas se reescalan con respecto a la distribución estadística del propio flujo. Para el gradiente de velocidad, la vorticidad y la tasa de deformación se emplean los cuantiles q_{50} y q_{90}, mientras que para la distancia a obstáculos se utilizan q_{10} y q_{50}. A partir de dichos cuantiles se construyen las variables normalizadas:

$$\hat{g}(c) = \mathrm{clip}\left(\frac{\mathrm{g(c)} - \mathrm{q}^{\mathrm{g}}_{50}}{\mathrm{q}^{\mathrm{g}}_{90} - \mathrm{q}^{\mathrm{g}}_{50}}, 0, 1 \right), \tag{9}$$

$$\hat{\omega}(c) = \mathrm{clip}\left(\frac{\omega(\mathrm{c}) - \mathrm{q}^{\omega}_{50}}{\mathrm{q}^{\omega}_{90} - \mathrm{q}^{\omega}_{50}}, 0, 1 \right), \tag{10}$$

$$\hat{\sigma}(c) = \mathrm{clip}\left(\frac{\sigma(\mathrm{c}) - \mathrm{q}^{\sigma}_{50}}{\mathrm{q}^{\sigma}_{90} - \mathrm{q}^{\sigma}_{50}}, 0, 1 \right), \tag{11}$$

y

$$\hat{d}(c) = 1 - \mathrm{clip}\left(\frac{\mathrm{d(c)} - \mathrm{q}^{\mathrm{d}}_{10}}{\mathrm{q}^{\mathrm{d}}_{50} - \mathrm{q}^{\mathrm{d}}_{10}}, 0, 1 \right), \tag{12}$$

donde $\mathrm{clip(x}, 0, 1)$ denota la saturación del valor x en el intervalo $[0, 1]$. De este modo, $\hat{g}$, $\hat{\omega}$ y $\hat{\sigma}$ crecen con la actividad local del flujo, mientras que $\hat{d}$ crece al aproximarse al obstáculo.

A partir de estas variables se construye un índice de actividad por celda,

$$s(c) = w_g\, \hat{g}(c) + w_\omega\, \hat{\omega}(c) + w_\sigma\, \hat{\sigma}(c) + w_d\, \hat{d}(c), \tag{13}$$

donde w_g, w_ω, w_σ y w_d son pesos positivos configurables que controlan la contribución relativa de cada término. Se utiliza este índice porque la exigencia física local no viene determinada por una única magnitud, sino por la combinación de varios indicadores y por su posición relativa dentro del flujo completo. Esta construcción permite expresar en una sola variable la actividad relativa de cada celda y elimina la dependencia de umbrales absolutos ajustados específicamente para cada escenario. Aunque los pesos del índice son configurables, en los experimentos presentados se mantienen fijos en todas las configuraciones consideradas.

La política automática no se reevalúa en cada paso temporal, sino con la periodicidad definida por el parámetro `adaptive_every`. En cada actualización adaptativa se recalculan los indicadores físicos, se reconstruye el índice de actividad y se obtiene una nueva clasificación de las celdas fluidas. Entre dos actualizaciones consecutivas, el mapa de precisión se mantiene.

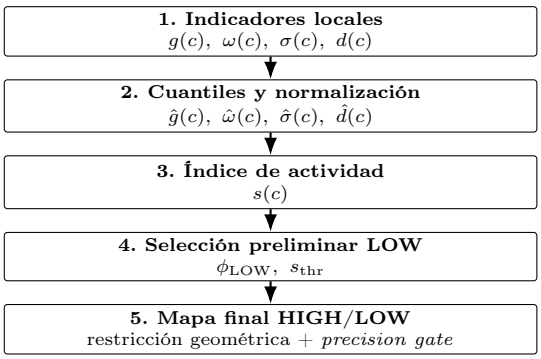


Fig. 1: Esquema compacto de la política automática. A partir de indicadores físicos locales se construye un índice de actividad, se seleccionan candidatas a LOW mediante una fracción objetivo y un umbral asociado, y la asignación final se corrige con restricción geométrica y *precision gate*.

Una vez calculado el índice de actividad en todas las celdas fluidas, la política automática identifica la fracción objetivo ϕ_{LOW} de celdas con menor actividad relativa y define a partir de ella el umbral de selección s_{thr}. En la presente implementación CPU, esta selección se materializa mediante una clasificación global del índice de actividad. Por tanto, el umbral no representa un valor físico absoluto fijado a priori, sino el valor del índice correspondiente a la fracción del dominio con menor actividad relativa. Las celdas con $s(c) \leq s_{\mathrm{thr}}$ se identifican preliminarmente como candidatas a operar en baja precisión.

Esta selección inicial no constituye una asignación rígida. En primer lugar, se introduce una restricción conservadora en la vecindad inmediata de los obstáculos, forzando alta precisión cuando la distancia discreta es menor o igual a una celda. En segundo lugar, la corrección numérica del sistema se garantiza mediante un mecanismo de validación, denominado *precision gate*, que evalúa cada conversión candidata HIGH→LOW directamente sobre las poblaciones distribucionales del método LBM. En función del modo configurado, la conversión se acepta únicamente si las poblaciones son exactamente representables o si el error de redondeo se mantiene por debajo de una tolerancia dada. En caso contrario, la celda se mantiene en alta precisión.

Este diseño desacopla la decisión física de la validación numérica. La Figura 1 resume las etapas principales de la política automática de asignación de precisión. La lógica detallada del procedimiento se recoge en el Algoritmo 1.

IV. Metodología experimental

Se consideran dos escenarios representativos:

- **E1**: flujo alrededor de un cilindro.
- **E2**: cilindros en tándem.

En ambos casos se compara una configuración manual de referencia frente a dos configuraciones auto-

Algoritmo 1 Política automática basada en índice de actividad

if iteración de actualización adaptativa **then**
  Calcular $g(c)$, $\omega(c)$, $\sigma(c)$ y $d(c)$ en celdas fluidas
  Obtener q_{50}, q_{90} para g, ω, σ y q_{10}, q_{50} para d
  for all celda fluida c **do**
    Calcular $\hat{g}(c)$, $\hat{\omega}(c)$, $\hat{\sigma}(c)$ y $\hat{d}(c)$
    Construir índice de actividad $s(c)$
  end for
  Determinar s_{thr} asociado a ϕ_{LOW}
  Marcar como candidatas LOW las celdas con $s(c) \leq s_{\text{thr}}$
  for all celda candidata LOW **do**
    if $d(c) \leq 1$ **or** *precision gate*(c) rechaza LOW **then**
      Reasignar c a HIGH
    end if
  end for
  Aplicar mapa final de precisión
end if

máticas:

- **t45**: política automática con fracción objetivo de celdas en baja precisión del 45 %.
- **t50**: política automática con fracción objetivo del 50 %.

La evaluación se realiza mediante cuatro grupos de métricas:

- porcentaje final de celdas en baja precisión;
- tasa de rechazo del mecanismo de control de precisión;
- número de Strouhal obtenido a partir de la señal de velocidad en una sonda fija;
- errores globales y locales respecto a los valores en alta precisión, medidos mediante normas relativas L_2 de velocidad y densidad y error máximo en velocidad.

Los errores relativos L_2 se calculan tomando como referencia los valores en alta precisión MPFR. El error máximo en velocidad se emplea como indicador complementario de posibles degradaciones locales. El número de Strouhal se estima a partir de la señal temporal de velocidad transversal registrada en una sonda fija situada en la región de estela, es decir, aguas abajo del obstáculo en el sentido del flujo, donde se manifiesta el desprendimiento periódico de vórtices. La fracción temporal de uso de baja precisión por celda se define como el cociente entre el número de iteraciones en las que una celda opera en modo LOW y el número total de iteraciones consideradas en el análisis.

En la Tabla I, la columna LOW (%) indica el porcentaje final de celdas que operan en baja precisión, Gate (%) representa la fracción de conversiones candidatas a LOW rechazadas por el *precision gate*, *St* denota el número de Strouhal, $L_2(u)$ y $L_2(\rho)$ son los errores relativos globales en velocidad y densidad respecto a la referencia en alta precisión, y $L_\infty(u)$ representa el error máximo en velocidad.

Además, se analizan mapas espaciales de uso de baja precisión y campos macroscópicos finales para estudiar la coherencia física y espacial del mecanismo adaptativo.

V. Resultados

El primer y principal resultado de la evaluación es que la política automática permite reducir de forma significativa la precisión efectiva del dominio, alcanzando configuraciones cercanas al 50 % de celdas LOW, sin comprometer la calidad física de la simulación.

A. Interpretación física del uso de baja precisión

La Figura 2 muestra la relación entre la fracción temporal de uso de baja precisión y tres indicadores físicos locales: distancia a obstáculo, vorticidad y módulo del gradiente de velocidad, para los escenarios E1 y E2. En ambos casos se observa un comportamiento consistente con la lógica del mecanismo adaptativo: la fracción LOW aumenta en regiones físicamente menos activas y disminuye al incrementarse la intensidad local del flujo.

La dependencia respecto a la vorticidad y al gradiente de velocidad es particularmente clara. En ambos escenarios, la fracción LOW permanece elevada en regiones de baja actividad y desciende de forma acusada en los intervalos de mayor vorticidad y mayor gradiente, lo que indica que la política automática concentra la alta precisión en zonas donde la dinámica local es más exigente.

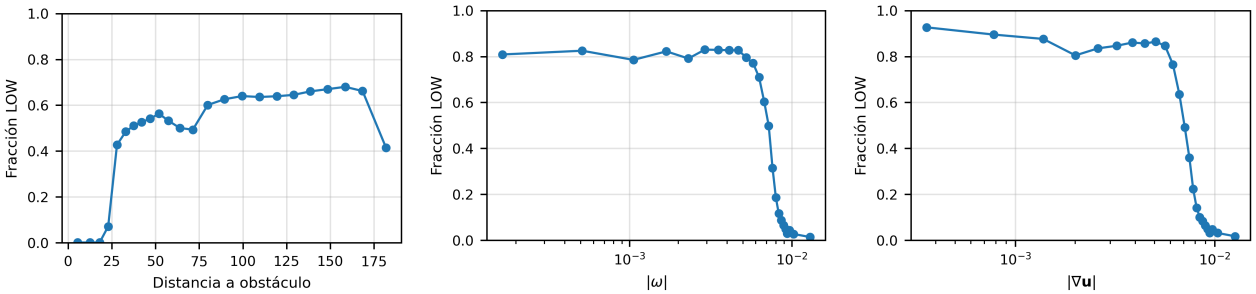
La relación con la distancia a obstáculo también presenta una tendencia global coherente, con mayor uso de baja precisión en regiones alejadas de la frontera sólida. No obstante, esta relación no es estrictamente monótona, especialmente en E2, donde aparecen oscilaciones locales asociadas a la interacción entre estelas y al hecho de que la distancia, por sí sola, no caracteriza completamente la actividad física del flujo. Además, las desviaciones observadas en los intervalos extremos de distancia se atribuyen previsiblemente al menor número de celdas presentes en las colas de la distribución.

En conjunto, estas correlaciones indican que la política automática genera una distribución espacial de precisión coherente con la estructura física del flujo.

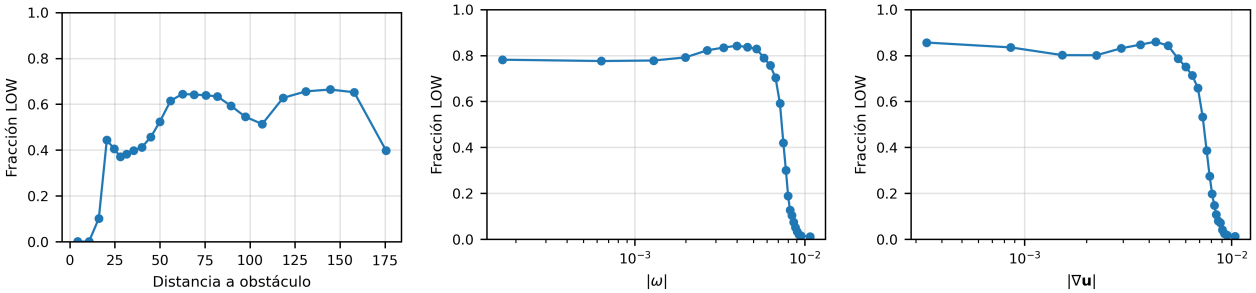
B. Distribución espacial del uso de baja precisión

La Figura 3 muestra la fracción temporal de uso de baja precisión por celda para la configuración *t50*. Se observa que la alta precisión se concentra en regiones físicamente activas, especialmente en las proximidades de los obstáculos y en zonas de estela compleja, mientras que la baja precisión domina en regiones de menor actividad física.

Este resultado refuerza la interpretabilidad física del mecanismo adaptativo y su coherencia espacial con la dinámica del flujo.

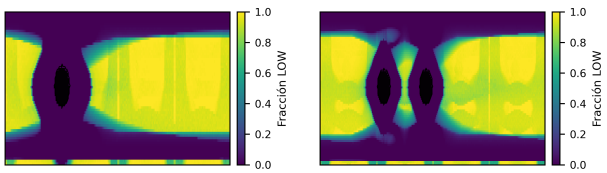


(a) E1: fracción LOW vs distancia (b) E1: fracción LOW vs vorticidad (c) E1: fracción LOW vs gradiente



(d) E2: fracción LOW vs distancia (e) E2: fracción LOW vs vorticidad (f) E2: fracción LOW vs gradiente

Fig. 2: Relación entre la fracción temporal de uso de baja precisión y tres indicadores físicos locales en los escenarios E1 y E2: distancia a obstáculo, vorticidad y módulo del gradiente de velocidad. En ambos casos, la fracción LOW aumenta en regiones físicamente menos activas. La dependencia es especialmente clara para vorticidad y gradiente, mientras que la relación con la distancia presenta oscilaciones locales, más visibles en E2, debidas a la complejidad espacial del flujo.



(a) E1 (b) E2

Fig. 3: Fracción temporal de uso de baja precisión por celda para la configuración *t50*.

Tabla I: Resumen de resultados para los modos manual y automático. LOW (%) indica el porcentaje final de celdas en baja precisión, Gate (%) la tasa de rechazo del *precision gate*, St el número de Strouhal, $L_2(u)$ y $L_2(\rho)$ los errores relativos globales en velocidad y densidad, y $L_\infty(u)$ el error máximo en velocidad.

Caso	Cfg.	LOW (%)	Gate (%)	St	$L_2(u)$	$L_2(\rho)$	$L_\infty(u)$
E1	man	37.1	1.76	0.0606	1.58e-4	1.25e-4	7.3e-5
E1	t45	44.5	1.48	0.0606	1.44e-4	6.4e-5	6.9e-5
E1	t50	49.5	1.44	0.0606	1.48e-4	8.9e-5	7.1e-5
E2	man	32.8	1.98	0.0591	2.68e-4	6.7e-5	1.09e-4
E2	t45	44.1	1.85	0.0591	4.42e-4	1.20e-4	2.03e-4
E2	t50	49.2	2.10	0.0591	3.64e-4	1.63e-4	1.61e-4

C. Validación física

La Tabla I recoge la comparación directa entre la configuración manual y las configuraciones automáticas consideradas.

El número de Strouhal permanece prácticamente invariante entre configuraciones, lo que indica que la dinámica global del flujo se conserva adecuadamente.

En cuanto a los errores respecto a los valores en alta precisión, tanto la velocidad como la densidad presentan errores relativos L_2 acotados en todos los casos. Además, la inclusión del error máximo en velocidad muestra que no aparecen degradaciones locales severas asociadas al uso de baja precisión en las configuraciones automáticas consideradas. En E1, las configuraciones automáticas mantienen errores globales y locales muy próximos a la configuración manual incluso al aproximarse al 50 % de celdas LOW. En E2, la configuración *t50* resulta más robusta que *t45* tanto en $L_2(u)$ como en $L_\infty(u)$, lo que sugiere, al menos en este escenario, que una asignación más definida de la fracción objetivo puede reducir zonas ambiguas en la clasificación de precisión.

Además, el mecanismo de validación numérica (*precision gate*) desempeña un papel clave en la estabilidad del sistema, evitando conversiones a baja precisión que introducirían errores no aceptables en las poblaciones distribucionales. Este control explícito permite desacoplar la agresividad en la reducción de precisión de la robustez numérica.

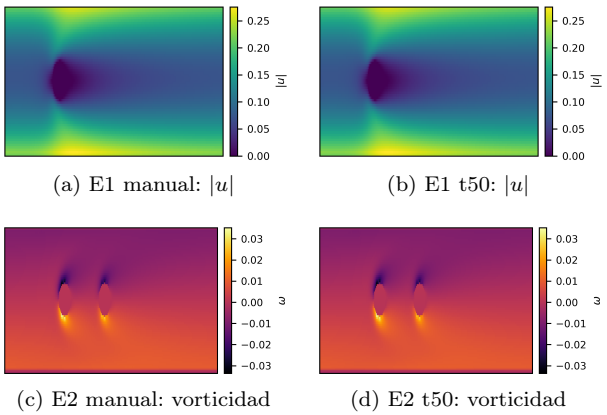
D. Comparación de campos macroscópicos

La Figura 4 compara visualmente campos macroscópicos de la solución manual y la configuración *t50*. En E1 se representa el módulo de la velocidad y en E2 la vorticidad. Las diferencias observadas son pequeñas y no alteran la estructura global del flujo, reforzando la validez física de la configuración automática.

VI. Conclusiones

Este trabajo presenta una plataforma de validación de precisión adaptativa para simulaciones Lattice Boltzmann basada en un esquema dual HIGH/LOW por celda implementado íntegramente mediante aritmética multiprecisión.

La principal aportación del trabajo es el diseño de una plataforma que permite evaluar de forma controlada el impacto de la reducción de precisión sobre la fidelidad física de la simulación. Esta plataforma



(a) E1 manual: $|u|$ (b) E1 t50: $|u|$

(c) E2 manual: vorticidad (d) E2 t50: vorticidad

Fig. 4: Comparación de campos macroscópicos entre la configuración manual y la política automática *t50*.

integra mecanismos de decisión basados en indicadores físicos locales junto con un mecanismo explícito de validación numérica (*precision gate*), permitiendo desacoplar la asignación de precisión de la robustez del resultado final.

Sobre esta base, se introduce una política automática de asignación de precisión que elimina la dependencia de umbrales empíricos mediante el uso de cuantiles y un índice de actividad que integra varios indicadores físicos locales en una única medida de actividad relativa.

Los resultados experimentales muestran que la plataforma alcanza configuraciones con aproximadamente un 50 % de celdas en baja precisión manteniendo la fidelidad física del sistema. En particular, se observa estabilidad en el número de Strouhal, errores relativos L_2 acotados en velocidad y densidad y ausencia de degradaciones locales severas en el campo de velocidades.

En conjunto, el trabajo muestra que es posible sustituir una política basada puramente en umbrales empíricos por una política automática más sistemática, alcanzando una fracción significativa de celdas LOW sin degradación física apreciable en las métricas analizadas.

Como trabajo futuro, la plataforma desarrollada constituye una base natural para su migración a GPU. En ese contexto, la lógica física del criterio global podría reformularse mediante mecanismos paralelos de estimación de umbral o percentiles, evitando una clasificación global completa y preservando el mismo marco de validación física establecido en este trabajo.

Agradecimientos

Este trabajo ha contado con la financiación del Ministerio de Ciencia, Innovación y Universidades mediante una ayuda para la Formación de Profesorado Universitario (FPU) concedida a Marcos Ramos Montes, referencia FPU22/02844.

Referencias

[1] Sauro Succi, *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*, Oxford University Press, 2001.

[2] Nicholas J. Higham and Theo Mary, "Mixed precision algorithms in numerical linear algebra," *Acta Numerica*, vol. 31, pp. 347–414, 2022.

[3] Amir Kashi, Hao Lu, William Brewer, M. Shankar, and Fei Wang, "Mixed-precision numerics in scientific applications: Survey and perspectives," *arXiv preprint arXiv:2412.19322*, 2024.

[4] Moritz Lehmann, Mathias J. Krause, Giorgio Amati, Marcello Sega, Jens Harting, and Stephan Gekle, "Accuracy and performance of the lattice boltzmann method with 64-bit, 32-bit, and customized 16-bit number formats," *Physical Review E*, vol. 106, pp. 015308, 2022.

[5] Andrea Tiribocchi, Andrea Montessori, Giorgio Amati, Massimo Bernaschi, Filippo Bonaccorso, Enzo Orlandini, Sauro Succi, and Marco Lauricella, "Lightweight lattice boltzmann," *The Journal of Chemical Physics*, vol. 158, no. 10, pp. 104101, 2023.

[6] Michał Januszewski and Marcin Kostur, "Sailfish: A flexible multi-gpu implementation of the lattice boltzmann method," *Computer Physics Communications*, vol. 185, no. 9, pp. 2350–2368, 2014.

[7] Kentaro Sano and Satoru Yamamoto, "Fpga-based streaming computation for lattice boltzmann method," in *International Conference on Field Programmable Technology*, 2007, pp. 233–236.

[8] Yoshiaki Kono, Kentaro Sano, and Satoru Yamamoto, "Scalability analysis of tightly-coupled fpga-cluster for lattice boltzmann computation," in *International Conference on Field Programmable Logic and Applications*, 2012, pp. 120–127.

[9] Xiaojun Zhai, Minsi Chen, Sahar Soheilian Esfahani, Abbes Amira, Faycal Bensaali, Julien Abi Nahed, Sarada Dakua, Abdulla Al-Ansari, and Ayman Zakaria, "Zynq soc based lattice-boltzmann simulation environment," in *IEEE GCC Conference and Exhibition*, 2019.

[10] Christoph Feichtinger, Johannes Habich, Johannes Götzfried, Takayuki Aoki, and Arndt Bode, "A flexible high-performance lattice boltzmann fpga implementation," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 1, pp. 1–17, 2012.

[11] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann, "Mpfr: A multiple-precision binary floating-point library with correct rounding," *ACM Transactions on Mathematical Software*, vol. 33, no. 2, pp. 13, 2007.

Aceleración de la Transcodificación Espacial en VVC para Aplicaciones LCEVC en Ultra HD

Pedro Jesús Martínez-Herrero[1], Gabriel Cebrián-Márquez[1], Antonio Jesús Díaz-Honrubia[2] y Pedro Cuenca[1]

Resumen— **Dada la elevada complejidad computacional de los estándares de codificación de vídeo recientes, este trabajo propone una estrategia rápida de transcodificación homogénea bajo el estándar Versatile Video Coding (VVC). El transcodificador propuesto reutiliza información de la secuencia codificada a resolución completa para generar una versión de la secuencia a menor resolución, con el fin de disminuir la complejidad algorítmica que presenta la codificación bajo este último estándar. Esta propuesta resulta de gran interés para escenarios de distribución adaptativa (ABR) y entornos de computación de altas prestaciones al disminuir la demanda de recursos que el proceso requiere a gran escala. Además, se plantea una estrategia de integración con el estándar Low Complexity Enhancement Video Coding (LCEVC), con el objetivo de aprovechar su arquitectura multicapa, así como su diseño de baja complejidad destinado a dispositivos con recursos limitados. Los resultados experimentales muestran que el método propuesto reduce el tiempo de transcodificación en un 72,14 %. Además, la integración de la propuesta en un marco de transcodificación VVC a LCEVC permite una reducción del 39,75 % en el tiempo de decodificación en la parte del cliente debido a la menor complejidad de LCEVC.**

Palabras clave— **Transcodificación de vídeo, VVC, LCEVC, reducción de resolución, transcodificación homogénea, UHD.**

I. Introducción

En los últimos años, la demanda global de contenido de vídeo ha crecido a un ritmo sin precedentes, impulsada por la expansión de las redes sociales, las plataformas de difusión de vídeo en tiempo real y los servicios de vídeo bajo demanda. Este auge no solo es cuantitativo, reflejado en un mayor volumen de vídeos compartidos, sino también cualitativo, ya que actualmente se ofrece más contenido en alta definición (HD) y ultra-alta definición (UHD). Esta tendencia se manifiesta en el planteamiento de los estándares de codificación más recientes, los cuales se diseñan orientados a comprimir eficientemente secuencias con resoluciones cada vez mayores y escenarios más complejos, junto con frecuencias de fotogramas cada vez más elevadas [1].

En este contexto, los estándares de compresión de vídeo han evolucionado significativamente desde H.264/*Advanced Video Coding* (AVC) [2] hasta H.265/*High-Efficiency Video Coding* (HEVC) [3], culminando recientemente en el estándar H.266/*Versatile Video Coding* (VVC) [4], el cual logra mejoras sustanciales en eficiencia de compresión, alcanzando reducciones de tasa de bits cercanas al 50 % respecto a HEVC para una calidad equivalente [1]. No obstante, estas mejoras se obtienen a costa de un incremento notable en la complejidad computacional, especialmente en tareas de codificación y transcodificación [5], [6].

En escenarios de distribución adaptativa (ABR) se codifican múltiples versiones de un mismo contenido con distintas resoluciones y tasas de bits para posibilitar su transmisión en redes y dispositivos con recursos limitados, surgiendo así la necesidad de generar múltiples representaciones del mismo contenido en distintas resoluciones y frecuencias de fotogramas dentro de aplicaciones de transmisión de vídeo adaptativa. Sin embargo, obtener representaciones de las secuencias a menor resolución implica una recodificación completa, y conlleva un coste computacional elevado. Una alternativa de creciente interés es la transcodificación heterogénea, en la que el estándar de salida difiere del de entrada, frente a la homogénea, en la que se mantiene el mismo estándar de codificación. En ambos casos, la información presente en el flujo comprimido puede aprovecharse para asistir determinadas etapas del proceso de transcodificación [7]. Diversos trabajos han explorado esta línea mediante la reutilización de información estructural y de particionamiento, mostrando reducciones significativas en tiempo de codificación [8], [9].

En este contexto, este trabajo propone un método de transcodificación homogénea dentro del estándar VVC para la reducción de la resolución espacial de secuencias UHD. La propuesta reutiliza la información de particionamiento de la secuencia codificada a resolución completa, aprovechando la estructura jerárquica basada en *Quadtree Plus Multi-Type Tree* (QT+MTT) definida en VVC [10], para generar una secuencia a menor resolución evitando un proceso completo de reparticionamiento. De este modo, se reduce significativamente la complejidad del proceso, lo que resulta de interés tanto en escenarios de transmisión adaptativa como en entornos donde la eficiencia computacional y el aprovechamiento de los recursos es clave, como es el caso de las granjas de transcodificación.

Además, esta propuesta posibilita la integración de la secuencia transcodificada como capa base dentro de un marco bajo el estándar *Low Complexity Enhancement Video Coding* (LCEVC) [11], el cual permite mejorar la calidad mediante capas de mejora de baja complejidad sobre una representación base de menor resolución [12]. Esta combinación permi-

[1]Pedro Jesús Martínez-Herrero, Gabriel Cebrián-Márquez y Pedro Cuenca pertenecen al grupo Redes y Arquitecturas de Altas Prestaciones (RAAP), Universidad de Castilla-La Mancha, Albacete, España.

[2]Antonio Jesús Díaz-Honrubia pertenece al grupo *Medical Data Analytics Laboratory* (MEDAL), Universidad Politécnica de Madrid, España.

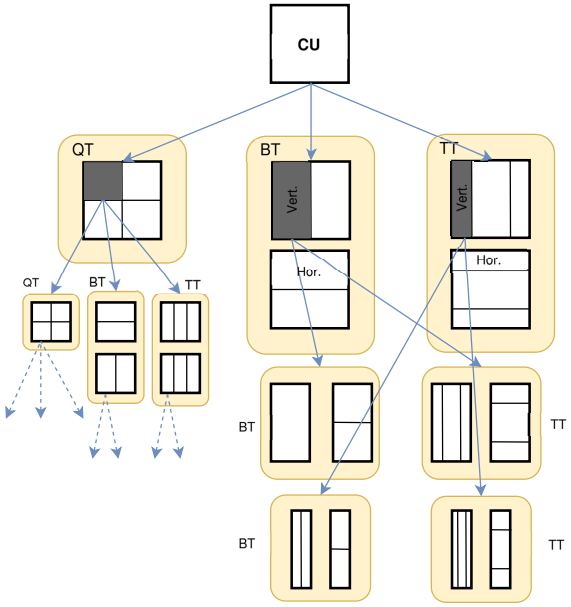


Fig. 1: Ejemplo de posibles casos del particionamiento QT+MTT para una CU en VVC. *Vert.* denota una partición en dirección vertical y *Hor.* denota una partición en dirección horizontal. Cada nodo sombreado en gris puede subdividirse adicionalmente mediante QT, BT o TT en función de la estructura actual del árbol y de determinadas restricciones del estándar.

te explotar las ventajas de una capa base eficiente junto con herramientas de mejora visual, facilitando soluciones escalables y computacionalmente eficientes para sistemas de distribución de vídeo.

II. Particionamiento QT+MTT en VVC

Para comprender cómo la información de particionamiento puede ser reutilizada o adaptada en un flujo de transcodificación homogénea en VVC, es necesario revisar la estrategia de particionamiento basada en bloques definida en este estándar, la cual constituye una etapa típica en la compresión de vídeo para adaptar la representación espacial de la señal a las características del contenido, incrementando así la eficiencia de codificación.

El esquema QT+MTT se propone como estrategia de particionamiento en el estándar y es uno de los principales responsables de la eficiencia de codificación en VVC. A diferencia de estándares anteriores como AVC o HEVC, que se basan principalmente en subdivisiones rectangulares sobre un árbol cuaternario (QT), VVC amplía esta estrategia permitiendo particiones binarias (BT) y ternarias (TT) en lo que se conoce como árbol multitipo (MTT) [4]. Acompañando a este incremento en la flexibilidad, el estándar aumenta el tamaño máximo de bloque hasta 128×128, para la codificación de contenidos en grandes resoluciones.

En este planteamiento, cada unidad de árbol de codificación (CTU) se subdivide inicialmente en cuatro bloques de igual tamaño y, posteriormente, cada nodo hoja puede dividirse adicionalmente mediante particiones binarias o ternarias, tanto en dirección horizontal como vertical, tal y como muestra la Fig. 1. Como resultado, las unidades de codificación (CUs) finales pueden adoptar formas rectangulares o cuadradas en función de la complejidad espacio-temporal del contenido. Además, a diferencia de HEVC, VVC integra los conceptos de unidades de predicción y transformación en la propia CU.

Asimismo, VVC introduce una estrategia de particionamiento paralela conocida como *Chroma Separate Tree* (CST), que permite aplicar esquemas de particionamiento independientes para los planos de luminancia y crominancia en fotogramas de tipo intra. Esta característica habilita una adaptación más precisa del particionamiento al contenido, permitiendo subdivisiones más finas en luminancia y bloques más grandes en crominancia, lo cual contribuye a maximizar la eficiencia de codificación [1].

A medida que aumenta la profundidad de la estructura en árbol, la complejidad del proceso de codificación incrementa de forma significativa. En particular, la incorporación de particiones binarias y ternarias extiende el espacio de búsqueda del codificador ampliamente, lo que implica evaluar un gran número de configuraciones posibles antes de seleccionar la más eficiente. Como consecuencia, el proceso planteado de particionamiento constituye uno de los factores responsables de la elevada complejidad de codificación en VVC [6].

Este incremento de complejidad es especialmente notable en comparación con esquemas de particionamiento más restringidos, como los utilizados en estándares anteriores. Aunque, al igual que en AVC y HEVC, VVC emplea estructuras jerárquicas en forma de árbol donde los nodos hoja representan las unidades finales de codificación, la posibilidad de generar particiones no cuadradas introduce un número mucho mayor de combinaciones posibles.

Diversos trabajos han estudiado el coste asociado a la aplicación de estos modos de partición. En [6], se muestra que la limitación de la profundidad máxima del esquema QT+MTT puede reducir de forma significativa la complejidad del codificador. En particular, restringir dicha profundidad de 6 a 4 niveles permite alcanzar reducciones de hasta el 58 % en el tiempo de codificación, mientras que la desactivación completa de las particiones MTT puede llegar a reducirlo en un 83 %, aunque a costa de una pérdida considerable en eficiencia de compresión.

En este contexto, el método propuesto en este trabajo se orienta a disminuir el impacto de esta complejidad mediante la reutilización de la información de particionamiento de la secuencia codificada bajo VVC a resolución completa. En lugar de explorar exhaustivamente el espacio de particiones en la secuencia de salida, se aprovechan decisiones previamente obtenidas, reduciendo así el coste computacional asociado a los niveles más profundos del árbol, y obteniendo una reducción notable en el tiempo de ejecución.

III. Algoritmo propuesto de transcodificación homogénea en VVC

En esta sección se describe el algoritmo propuesto de transcodificación homogénea en VVC, cuyo ob-

jetivo es transformar y reutilizar la información de particionamiento disponible en la secuencia previamente codificada en VVC para acelerar la generación de una nueva codificación a mitad de resolución. Al aprovechar decisiones de particionamiento ya determinadas, la propuesta evita una parte significativa del proceso convencional de optimización tasa-distorsión (RDO), reduciendo así el tiempo de codificación razonablemente.

A. Descripción general del algoritmo

Partiendo de una secuencia UHD codificada en VVC, el algoritmo propuesto vuelve a codificar el contenido a mitad de resolución (HD), reduciendo la complejidad de la transcodificación al reutilizar cierta información del flujo codificado. Para ello, se diferencian tres etapas en el procedimiento:

1. *Extracción de información de particionamiento:* Se emplea un decodificador VVC para reconstruir la secuencia de entrada codificada a resolución completa. Durante este proceso se extrae, para cada CTU, la información correspondiente al árbol de particionamiento QT+MTT, incluyendo el orden y el tipo de particiones aplicadas en cada nivel de profundidad. Además, se extrae la información relativa al tamaño y posición de las CUs, así como el particionamiento CST utilizado para diferenciar los planos de luminancia y crominancia. Estos elementos constituyen la información reutilizada durante la transcodificación.
2. *Submuestreo espacial y adaptación del particionamiento QT+MTT:*
   Con el fin de adaptar la estructura completa de particionamiento para una codificación a la mitad de la resolución en ambas dimensiones, se aplican dos transformaciones. En primer lugar, todas las CUs reducen sus posiciones y tamaños por un factor de 0,5, con el fin de que las coordenadas y dimensiones espaciales se alineen con la resolución objetivo. En segundo lugar, con el propósito de que el tamaño de CTU se mantenga en la secuencia codificada de salida, se introduce una nueva partición QT extra en la raíz del árbol de particionamiento. De este modo, cada una de las nuevas subparticiones hereda dimensiones equivalentes a la mitad del tamaño original del CTU. Por ejemplo, si en la secuencia de entrada el tamaño de CTU es de 128×128 píxeles, cada CTU original pasa a corresponderse con un bloque de 64×64 píxeles, asociado a una de las cuatro subparticiones generadas tras aplicar una división QT en el CTU de salida.
   La Fig. 2 ilustra este procedimiento de adaptación, en el que se añade una partición QT extra en el nivel superior del CTU para acomodar la estructura transformada.
3. *Segunda codificación con reutilización del particionamiento:* Una vez transformada la información de particionamiento, el codificador VVC recibe la secuencia a mitad de resolución junto con las decisiones extraídas de la secuencia codificada original. En cada nivel de profundidad, el codificador compara su secuencia de particiones propuesta por el codificador a mitad de resolución con la información reutilizada. Si la partición forzada resulta válida bajo las restricciones del estándar VVC, se aplica directamente, evitando así evaluaciones de modos y el proceso completo de RDO y la complejidad algorítmica asociada al proceso. En caso contrario, cuando la partición forzada no sea válida debido a las transformaciones derivadas del submuestreo, el codificador descarta dicha decisión y vuelve a su proceso convencional, preservando la conformidad con el estándar.

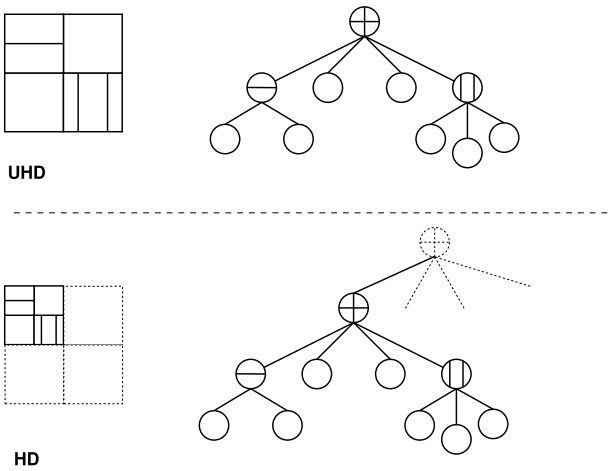


Fig. 2: Proceso de submuestreo espacial y adaptación del particionamiento QT+MTT. La parte superior muestra un CTU de la secuencia UHD y su estructura de particionamiento original. La parte inferior representa su adaptación a la secuencia HD, donde las coordenadas y tamaños de las CUs se reducen por un factor de 0,5 y el subárbol resultante se inserta bajo una partición QT adicional en la raíz del CTU de salida, representada con líneas discontinuas, cuyas dimensiones son la mitad de las del CTU original.

B. Consideraciones sobre la transformación

La reducción de la resolución espacial a la mitad en ambas dimensiones conlleva diversas consecuencias a la hora de transformar la resolución. Por ello, obliga a tratar ciertos casos particulares durante la transformación del árbol de particionamiento.

En primer lugar, para mantener el mismo tamaño de CTU en la secuencia transcodificada, todos los CTUs deben ser divididos mediante particionamiento QT con el fin de acomodar los bloques transformados de la secuencia original. Consecuentemente, en la secuencia codificada de salida no aparecen CUs del mayor tamaño permitido. Por ejemplo, para un tamaño de CTU de 128×128 píxeles, las CUs de dicho tamaño presentes en la secuencia original se transforman en CUs de 64×64 píxeles. Esta restricción puede afectar negativamente a la eficiencia de codificación, especialmente en regiones homogéneas con baja complejidad espacial, donde el codificador suele favorecer tamaños de partición más grandes.

Con el fin de mitigar esta limitación, la propuesta incorpora una heurística sencilla. Si ninguno de los hasta cuatro CTUs originales que se corresponden

con un único CTU de la secuencia reducida presenta particiones QT, binarias o ternarias, se permite al codificador realizar un proceso completo de RDO sobre la región correspondiente de la secuencia reducida. Si, como resultado de este proceso, el codificador decide no subdividir la región, dicha decisión se mantiene, de acuerdo con la observación de que en esa zona ya existía una tendencia a emplear un menor número de particiones. Por el contrario, si el codificador determina que resulta beneficioso seguir subdividiendo la región, la partición heredada no se impone directamente. En su lugar, el codificador retoma el proceso convencional de particionamiento basado en RDO. En estos casos, la baja complejidad inferida para la región permite suponer que la búsqueda posterior de particiones tendrá un coste computacional acotado. Los resultados preliminares muestran que esta heurística proporciona un mejor compromiso entre la reducción del tiempo de codificación y el incremento en tasa de bits, además de simplificar la estructura final de particionamiento en la codificación a mitad de resolución.

En segundo lugar, aparecen particiones no válidas según el estándar cuando las unidades mínimas de partición del árbol original, como por ejemplo bloques de 8×8 píxeles, dejan de cumplir las restricciones tras la reducción de resolución. En estos casos, dicha información de particionamiento se descarta y el codificador vuelve a su proceso de decisión por defecto. En particular, se le permite explorar particiones binarias y ternarias de acuerdo con las restricciones del estándar y con los parámetros de configuración establecidos, lo que le permite adaptarse de forma flexible a la nueva resolución sin generar tamaños de bloque no válidos. Esto otorga mayor libertad en los niveles más profundos del árbol para completar adecuadamente la estructura de CUs.

En tercer lugar, la reutilización de un árbol de particionamiento generado a resolución completa conserva implícitamente las decisiones tasa-distorsión (RD) asociadas al parámetro de cuantificación (QP) original. Tras reducir la resolución a la mitad en ambas dimensiones, la información de cada bloque es menor, la correlación espacial aumenta y por ello los valores RD entre resoluciones no son equivalentes. Como consecuencia, se utiliza una cuantización menos agresiva, para así mantener el rendimiento de la secuencia transcodificada con un particionamiento obtenido a partir de una secuencia con una mayor cantidad de información. Los experimentos iniciales demuestran que QPs entre 4 y 7 unidades menores al empleado en la secuencia original permiten mejorar la eficiencia de codificación frente al uso directo del mismo QP de entrada. No obstante, este desplazamiento suele estar condicionado en la práctica por mecanismos de control de tasa orientados a satisfacer restricciones de ancho de banda o almacenamiento. Es por ello que este trabajo propone ΔQP como un parámetro ajustable, cuyo valor óptimo se determina empíricamente en la sección de evaluación experimental.

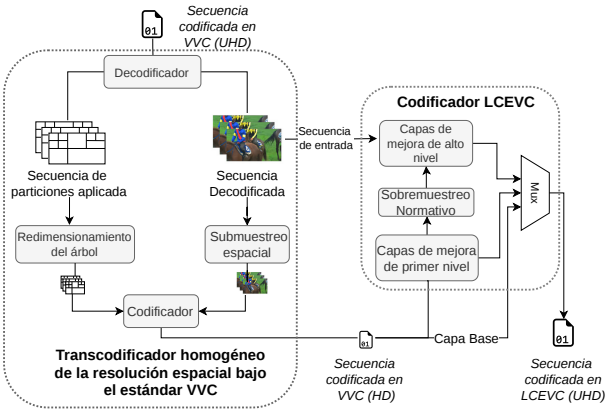


Fig. 3: Esquema general del flujo propuesto, en el que una secuencia codificada en VVC se transcodifica homogéneamente a mitad de resolución para generar la capa base en HD, que posteriormente se emplea dentro del codificador LCEVC para obtener una secuencia final en UHD.

C. Integración del flujo resultante con LCEVC

LCEVC mejora la calidad de vídeo mediante la codificación de una capa base a resolución reducida utilizando un códec de capa única, y reconstruye la resolución completa mediante hasta dos subcapas de mejora [12]. La primera subcapa codifica los residuos de predicción (diferencia entre la señal original y la señal predicha) antes del proceso de escalado, mientras que la segunda refina la salida añadiendo residuos tras el escalado. LCEVC no ha sido concebido para ser utilizado de forma independiente, sino que se diseña para mejorar códecs de vídeo existentes.

En este contexto, el proceso de transcodificación propuesto se integra dentro de este marco generando una capa base homogénea a resolución reducida utilizando un códec VVC, en coherencia con el escenario de transcodificación planteado. Mediante la generación de un flujo homogéneo a mitad de resolución, esta etapa de transcodificación produce una base adecuada para LCEVC y, además, acelera de forma significativa la etapa de codificación a dicha resolución utilizando el algoritmo propuesto. Además, al emplear una capa base de baja resolución, esta propuesta también afecta a la complejidad en el lado del cliente. En un flujo típico, el decodificador recompone una secuencia base a mitad de resolución junto con una o dos subcapas de mejora para reconstruir la secuencia a resolución completa. Como resultado, el coste total de decodificación resulta inferior al de la decodificación de un flujo VVC de resolución completa de una sola capa, tal como se analizará en la Sección IV.

De acuerdo con la Fig. 3, el flujo completo del algoritmo integra la reutilización de la información de particionamiento, el proceso de submuestreo espacial y su posterior uso dentro del esquema con LCEVC.

IV. Evaluación experimental

En esta sección se evalúa el rendimiento del método propuesto desde dos perspectivas complementarias. En primer lugar, se analiza su comportamiento como esquema de transcodificación homogénea VVC a VVC,

comparándolo con un transcodificador en cascada que no reutiliza información de particionamiento. En segundo lugar, se estudia la integración de la secuencia VVC transcodificada a mitad de resolución como capa base dentro de un esquema LCEVC, considerando tanto la eficiencia de codificación como la reducción de la complejidad de decodificación.

A. Condiciones experimentales

Los experimentos se realizaron sobre un procesador Intel® Xeon® CPU E5-2630L v3 a 1.80 GHz, con 32 GB de memoria y Ubuntu 20.04.4 LTS. Con el fin de garantizar la reproducibilidad de los resultados, la tecnología Turbo Boost fue desactivada y todas las codificaciones se ejecutaron en modo monohilo. No obstante, la propuesta no introduce dependencias que limiten la paralelización disponible en el codificador, pues el ámbito de actuación del algoritmo propuesto es a nivel de CTU. Consecuentemente, la propuesta podría compatibilizarse con las estrategias de paralelización de mayor granularidad que los codificadores puedan implementar, de modo que la aceleración del tiempo de ejecución mostrada en esta evaluación se debe interpretar como una reducción de complejidad complementaria al paralelismo de la codificación.

El sistema de transcodificación propuesto utiliza VVenC 1.13.1 [13], compilado con GCC 9.4.0 y soporte AVX2. Todas las pruebas de codificación se realizaron con la configuración *slow*, y con el modo *Random Access* (RA), fijando el parámetro `--threads` a 1 y desactivando la adaptación perceptual del QP (QPA) mediante `--qpa 0`, con el fin de preservar el comportamiento natural del particionamiento. Asimismo, para mantener la compatibilidad con el flujo de codificación de LCEVC, se forzó que el primer fotograma en orden de presentación fuese de tipo IDR.

Para la capa de mejora se empleó la implementación de referencia LTM 7.0.0 [14], basada en VTM 19.1 [15] como decodificador VVC. El submuestreo de la capa base a mitad de resolución se realizó empleando el algoritmo Lanczos3 sobre luminancias y crominancias.

La evaluación se ha centrado exclusivamente en secuencias UHD, concretamente en las clases A1 y A2 de las condiciones comunes de prueba definidas por JVET [16]. Todas las secuencias fueron codificadas en formato 10 bits y submuestreo cromático 4:2:0, utilizando los valores de QP 22, 27, 32 y 37.

B. Métricas

La evaluación se llevó a cabo empleando dos tipos de métricas: eficiencia de codificación y reducción de tiempo. La eficiencia de codificación se cuantificó mediante BD-rate [17], utilizando como métricas de calidad objetiva tanto PSNR como VMAF. Mientras que PSNR mide la similitud píxel a píxel, VMAF constituye una métrica perceptual más adecuada para escenarios donde intervienen procesos de escalado y realce visual, como ocurre en LCEVC [18], [19]. Por este motivo, aunque en el estudio se exponen ambos valores, el análisis se apoya principalmente en los resultados obtenidos con VMAF.

Tabla I: Eficiencia de codificación y ahorro de tiempo del algoritmo de transcodificación propuesto

Secuencia	ΔQP	BD-rate PSNR (%)	BD-rate VMAF (%)	TR_{enc} (%)
Tango2	4	5,84	6,03	68,73
Drums100	5	13,91	9,32	73,74
Campfire	6	10,96	10,65	76,45
ToddlerFountain2	6	1,93	7,26	77,22
CatRobot	4	17,59	13,44	70,10
TrafficFlow	5	9,87	10,08	71,01
DaylightRoad2	4	16,44	12,20	69,68
Rollercoaster2	4	10,45	8,66	70,23
Media	-	10,87	9,70	72,14

La reducción de tiempo se expresa mediante la métrica TR, definida como:

$$\mathrm{TR}(\,\%) = \frac{\mathrm{T_{base}} - \mathrm{T_{prueba}}}{\mathrm{T_{base}}} \cdot 100,$$

donde T_{base} representa el tiempo del método de referencia y T_{prueba} el correspondiente al método evaluado. En el caso de la transcodificación VVC, esta métrica se denota como $\mathrm{TR_{enc}}$, mientras que en el escenario LCEVC se emplea $\mathrm{TR_{dec}}$ para reflejar la reducción del tiempo de decodificación frente a la secuencia codificada en VVC original a resolución completa.

C. Selección del parámetro de cuantificación

Como se indicó en la Sección III-B, la reducción de resolución modifica los valores RD del contenido a codificar, de modo que el valor de QP más adecuado para la secuencia transcodificada depende, en mayor medida, de la naturaleza de la secuencia. En consecuencia, el parámetro de cuantificación de salida se define como un valor reducido del valor original, tal que

$$\mathrm{QP_{transcodificado}} = \mathrm{QP_{original}} - \Delta\mathrm{QP}.$$

Experimentos preliminares demostraron que valores bajos de ΔQP tendían a producir particionamientos excesivamente complejos y pérdidas apreciables de calidad, mientras que valores demasiado altos incrementaban notablemente la tasa de bits de la secuencia codificada resultante. Dentro de las secuencias UHD consideradas, los valores finalmente seleccionados se situaron entre 4 y 6, tal y como se recoge en la Tabla I. Esta selección se ha realizado buscando un equilibrio razonable entre la tasa de bits obtenida al transcodificar, la degradación de la calidad perceptual y la capacidad de reutilización efectiva del particionamiento original.

D. Resultados experimentales

La Tabla I recoge los resultados obtenidos para el escenario de transcodificación homogénea VVC. En general, el método propuesto alcanza una reducción del tiempo de transcodificación del 72,14 %, con una penalización media del 9,70 % en $\mathrm{BD\text{-}rate_{VMAF}}$. Estos resultados muestran que la reutilización de la información de particionamiento permite reducir de

Tabla II: Resultados del método propuesto tras su integración con LCEVC

Secuencia	BD-rate PSNR (%)	BD-rate VMAF (%)	TR_{dec} (%)
Tango2	6,41	5,18	44,93
Drums100	13,88	9,58	43,80
Campfire	7,96	8,82	34,64
ToddlerFountain2	4,07	3,86	43,35
CatRobot	17,35	14,17	34,65
TrafficFlow	9,71	8,91	29,76
DaylightRoad2	17,82	13,59	41,55
Rollercoaster2	10,41	8,03	45,34
Media	10,95	9,02	39,75

forma notable la complejidad computacional, manteniendo una degradación perceptual moderada. Entre las secuencias evaluadas, *Tango2* presenta el menor impacto en términos de BD-$\text{rate}_{\text{VMAF}}$ (6,03 %), mientras que *CatRobot* y *DaylightRoad2* muestran el mayor impacto en la compresión, aunque manteniendo ahorros en el tiempo de ejecución superiores al 69 %.

La Tabla II recoge los resultados obtenidos al integrar la secuencia transcodificada en VVC como capa base dentro del esquema LCEVC. En este caso, la penalización media observada es del 9,02 % en BD-$\text{rate}_{\text{VMAF}}$, valor muy próximo al obtenido en la transcodificación VVC a VVC, lo que indica que la capa de mejora de LCEVC preserva adecuadamente la calidad perceptual del sistema propuesto. En términos de complejidad, el tiempo de decodificación se reduce un 39,75 % de media respecto a la secuencia codificada bajo VVC original a resolución completa.

En general, los resultados obtenidos sobre las secuencias UHD consideradas confirman que el método propuesto constituye una solución válida para acelerar la transcodificación homogénea en VVC y, al mismo tiempo, generar una capa base para LCEVC con una reducción apreciable tanto en complejidad de codificación como de decodificación.

V. Conclusiones

En este trabajo se ha presentado un método de transcodificación homogénea para VVC basado en la reutilización de información de particionamiento procedente de una codificación previa. La propuesta parte de la idea de aprovechar la estructura espacial ya determinada en una secuencia codificada para acelerar el proceso de recodificación, reduciendo así el coste asociado a la exploración del espacio de particionamiento. De este modo, se plantea una estrategia orientada a disminuir la complejidad computacional del proceso sin comprometer de forma severa la eficiencia de compresión.

Los resultados experimentales obtenidos constatan que el método propuesto reduce de manera notable el tiempo de codificación, manteniendo una moderada penalización en términos de BD-rate. Esta relación entre ahorro computacional y pérdida de eficiencia sitúa a la propuesta como una alternativa válida para escenarios en los que la transcodificación debe realizarse con requisitos de escalabilidad o de limitaciones energéticas, como sucede en aplicaciones de adaptación de contenido y en sistemas orientados a ABR. Asimismo, la evaluación llevada a cabo en el contexto de integración con LCEVC muestra que la utilización de una capa base VVC generada mediante el método propuesto permite mejorar las prestaciones en el escenario analizado. En particular, la reducción de la complejidad de decodificación otorga interés práctico de la solución en entornos con recursos limitados.

Como líneas de trabajo futuro, resulta de interés profundizar en mecanismos adaptativos que permitan ajustar de forma más precisa la reutilización de la información de particionamiento en función de las características del contenido. Del mismo modo, sería conveniente extender la validación del método a otros escenarios de uso, así como analizar su comportamiento en infraestructuras de computación distribuida y en entornos de procesamiento a gran escala.

Agradecimientos

Esta publicación es parte de los proyectos de I+D+i PID2021-128167OA-I00, PID2021-123627OB-C52 y PID2022-142332OA-I00 financiados por MCIN/AEI/10.13039/501100011033/ y FEDER "una forma de hacer Europa", así como del proyecto SBPLY/21/180501/000195 financiado por la Junta de Comunidades de Castilla-La Mancha.

Esta publicación es parte de los proyectos de I+D+i PID2024-161538OB-I00 y PID2024-158682OB-C32 financiados por MICIU/AEI/10.13039/501100011033/ y FEDER/UE.

Referencias

[1] Benjamin Bross, Jianle Chen, Shan Liu, and Ye-Kui Wang, "Versatile Video Coding (H.266): Overview and Applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3736–3764, 2021.

[2] ISO/IEC and ITU-T, "Advanced Video Coding for Generic Audiovisual Services. ITU-T Recommendation H.264 and ISO/IEC 14496-10," May 2003.

[3] ISO/IEC and ITU-T, "High Efficiency Video Coding (HEVC). ITU-T Recommendation H.265 and ISO/IEC 23008-2," Apr. 2013.

[4] ISO/IEC and ITU-T, "Versatile Video Coding (VVC). ITU-T Recommendation H.266 and ISO/IEC 23090-3," Aug. 2020.

[5] Frank Bossen, Karsten Sühring, Adam Wieckowski, and Shan Liu, "VVC Complexity and Software Implementation Analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3765–3778, 2021.

[6] Farhad Pakdaman, Mohammad Ali Adelimanesh, Moncef Gabbouj, and Mahmoud Reza Hashemi, "Complexity Analysis Of Next-Generation VVC Encoding And Decoding," in *2020 IEEE International Conference on Image Processing (ICIP)*, 2020, pp. 3134–3138.

[7] Ishfaq Ahmad, Xiaohui Wei, Yu Sun, and Ya-Qin Zhang, "Video transcoding: An overview of various techniques and research issues," *IEEE Transactions on Multimedia*, vol. 7, no. 5, pp. 793–804, Oct. 2005.

[8] Xi Xie, Kai Zhang, Li Zhang, Meng Wang, Junru Li, and Shiqi Wang, "Low Complexity Transcoding from HEVC to VVC," in *2023 IEEE International Conference on Multimedia and Expo (ICME)*, 2023, pp. 2573–2578.

[9] David García-Lucas, Gabriel Cebrián-Márquez, Antonio Jesús Díaz-Honrubia, Thanuja Mallikarachchi, and Pedro Cuenca, "A fast full partitioning algorithm for HEVC-to-VVC video transcoding using Bayesian classifiers," *Journal of Visual Communication and Image Representation*, vol. 94, pp. 103829, 2023.

[10] Yu-Wen Huang, Jicheng An, Han Huang, Xiang Li, Shih-Ta Hsiang, Kai Zhang, Han Gao, Jackie Ma, and Olena Chubach, "Block Partitioning Structure in the VVC

Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3818–3833, 2021.
[11] ISO/IEC, "Low Complexity Enhancement Video Coding (LCEVC). ISO/IEC 23094-2," Nov. 2021.
[12] Stefano Battista, Guido Meardi, Simone Ferrara, Lorenzo Ciccarelli, Florian Maurer, Massimo Conti, and Simone Orcioni, "Overview of the Low Complexity Enhancement Video Coding (LCEVC) Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 11, pp. 7983–7995, 2022.
[13] Adam Wieckowski, Jens Brandenburg, Tobias Hinz, Christian Bartnik, Valeri George, Gabriel Hege, Christian Helmrich, Anastasia Henkel, Christian Lehmann, Christian Stoffers, Ivan Zupancic, Benjamin Bross, and Detlev Marpe, "VVenC: An Open And Optimized VVC Encoder Implementation," in *2021 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, 2021, pp. 66–67.
[14] MPEG, "Low complexity enhancement video coding (lcevc) test model (ltm), version 7.0.0," `https://git.mpeg.expert/MPEG/Video/LCEVC/LTM/-/tags/LTM_7_0`, GitLab tag, accessed July 20, 2025.
[15] Fraunhofer HHI and JVET, "Versatile Video Coding Test Model (VTM) Reference Software, Version 19.1," `https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/-/releases/VTM-19.1`, GitLab release, accessed July 20, 2025.
[16] Frank Bossen, "Common Test Conditions and Software Reference Configurations," Tech. Rep. JCTVC-L1100, JCT-VC, Jan. 2013.
[17] Gisle Bjøntegaard, "Calculation of average PSNR differences between RD-curves," 2001, Thirteenth Meeting, Austin, Texas, USA, 2-4 April, 2001, Document VCEG-M33.
[18] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara, "Toward a practical perceptual video quality metric," 2016, Accessed April 30, 2025.
[19] Reza Rassool, "VMAF reproducibility: Validating a perceptual practical video quality metric," in *2017 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, 2017, pp. 1–2.

Lenguajes, compiladores y herramientas de programación y ejecución paralela (2/2)

Detección de crisis epilépticas en un SoC RISC-V de ultra bajo consumo

David Díaz-Reyes[1], Javier Hormigo[1] y Angeles G. Navarro[1]

Resumen— **La monitorización ambulatoria continua de crisis epilépticas exige dispositivos ponibles (*wearables*) que combinen una alta precisión diagnóstica, explicabilidad clínica y un consumo energético ultrabajo. Para abordar este reto, este artículo presenta una solución arquitectónica y algorítmica orientada a la monitorización continua sobre el SoC RISC-V heterogéneo HEEPsilon. El sistema se basa en el co-diseño *hardware-software* que adapta y mapea el algoritmo de detección de ataques epilépticos PaFESD, que es explicable y está fundamentado en el emparejamiento morfológico de señales de ataque utilizando *Dynamic Time Warping* (DTW). Para resolver las restricciones de consumo energético, la arquitectura explota el dispositivo reconfigurable (CGRA) integrado en el SoC y que en nuestro caso se usa para acelerar la extracción de características elementales. Además, proponemos la instanciación de un nuevo acelerador ASIC dedicado a la computación de la DTW. A nivel algorítmico, se introduce una novedosa función de coste continua basada en *softmin* para el entrenamiento *offline*, mitigando el impacto de la severa pérdida de resolución espacial y logrando recuperar parte de la sensibilidad. Los resultados experimentales demuestran que la plataforma diseñada supera holgadamente las restricciones de tiempo real, alcanzando factores de mejora de hasta 12× cuando se incorpora el dispositivo DTW y 8× cuando se utiliza la CGRA, frente a la ejecución en el microcontrolador. Esta holgura computacional habilita una agresiva estrategia de reducción del consumo de energía (*race-to-sleep*) en un factor de 3.47×, consolidando la viabilidad del sistema para operar en regímenes *always-on*.**

Palabras clave— **Detección de crisis epilépticas, SoC RISC-V, CGRA, Acelerador *hardware*, *Dynamic Time Warping* (DTW), Ultrabajo consumo, *Edge computing*.**

I. Introducción

A. Contexto clínico y requisitos de monitorización ambulatoria

La epilepsia se define como una predisposición anormal y duradera a generar crisis imprevisibles [1]. Estas alteraciones cerebrales transitorias merman drásticamente la calidad de vida y conllevan riesgos vitales como la Muerte Súbita Inesperada (SUDEP). Por ello, su monitorización continua es indispensable para identificar episodios severos, especialmente las crisis generalizadas tónico-clónicas, con latencia mínima. Esta inmediatez permite habilitar protocolos de rescate inmediatos (reposicionamiento o medicación), exigiendo al hardware subyacente un estricto requerimiento de procesamiento en tiempo real.

Las crisis se clasifican anatómicamente en focales (típicamente temporales [2]) y generalizadas [3], condicionando el diseño del hardware. Puesto que ambas manifestaciones son detectables en la región temporal, esta zona resulta idónea para integrar sensores con alta relación señal-ruido por su menor grosor dérmico y ausencia de cabello. Consecuentemente, limitar la monitorización a canales laterales en dispositivos discretos (ej. unas gafas) resulta óptimo para conciliar la integridad de la señal con la usabilidad y la eliminación del estigma.

Desde una perspectiva de diseño de sistemas empotrados, el objetivo es garantizar una detección binaria robusta frente a los retos físicos de la subclasificación clínica. Las crisis con fuerte componente motor inducen severos artefactos electromiográficos (EMG) que saturan los canales de EEG de superficie, invalidando algoritmos basados únicamente en umbrales de energía. Simultáneamente, los episodios sin manifestaciones motoras evidentes presentan un reto de confianza; si el sistema alerta ante una descarga subclínica, el usuario puede percibirlo como un falso positivo. Ambas casuísticas exigen abandonar modelos de clasificación opacos e implementar algoritmos inherentemente explicables. Al basar la detección en el emparejamiento morfológico de la señal de ataque (*pattern matching*), se mitiga el impacto del ruido muscular y se proporciona al neurólogo una traza visual justificable de la anomalía eléctrica para un ajuste farmacológico de precisión.

Con 51.7 millones de afectados y más del 80 % de la morbimortalidad concentrada en países de bajos y medios ingresos (LMICs), depender de la nube (*Cloud Computing*) resulta inviable. Este escenario exige sistemas en el extremo (*edge*) accesibles y energéticamente autónomos. En consecuencia, es imperativo adoptar plataformas SoC de ultrabajo consumo basadas en arquitecturas abiertas (ej. RISC-V) que ejecuten la inferencia de forma estrictamente local, habilitando una monitorización ininterrumpida (*always-on*).

Finalmente, la arquitectura de detección debe responder a un presupuesto temporal crítico dictado por la asimetría entre el inicio electrográfico y el clínico. Para que una descarga sincrónica supere la atenuación del cráneo y sea perceptible en superficie, lo que requiere el reclutamiento de al menos 6 cm^2 de área cortical [4], existe una latencia inherente de entre escasos segundos y medio minuto antes de la manifestación de síntomas [5,6]. Si bien la predicción preictal en horizontes de 20 a 50 minutos es viable en entornos experimentales [7,8], su dependencia de registros invasivos o uso de canales de señal de alta densidad la aleja del uso ambulatorio. Por ello, la presente propuesta se centra en perfeccionar la detección reactiva dentro de este estrecho margen temporal, asumiendo las restricciones espaciales y de consumo que impone un dispositivo *wearable* de adopción realista.

[1]Dpto. de Arquitectura de Computadores, Universidad de Málaga, e-mail: {daviddr,fjhormigo,magonzalez}@uma.es.

B. Limitaciones de las soluciones actuales

Más allá de las limitaciones de las arquitecturas de propósito general, la tendencia hegemónica en la literatura actual se orienta hacia los modelos de *Deep Learning* [9]. Sin embargo, su despliegue en dispositivos *wearables* enfrenta barreras fundamentales: un coste computacional que excede holgadamente los presupuestos energéticos del *edge*, y una carencia de explicabilidad nativa que dificulta su adopción clínica. A esto se suma un sesgo de validación frecuente: gran parte de las soluciones se evalúan sobre conjuntos de datos idealizados o asumiendo una alta densidad de canales [10, 11]. Cuando estos modelos se trasladan a entornos ambulatorios reales y uso de canales de señal reducidos, sufren una caída drástica de precisión y un aumento de falsas alarmas [12–14], confirmando la desconexión entre la viabilidad algorítmica teórica y la realidad del paciente.

Por otro lado, los intentos de acelerar estos modelos mediante hardware dedicado (ASIC fijos o aceleradores de CNN) adolecen de una rigidez que impide la adaptación a nuevos biomarcadores o a la variabilidad inter-paciente [15, 16]. La personalización mediante reentrenamiento en el dispositivo resulta costosa en recursos y compleja de implementar [16]. Frente a esto, en este trabajo proponemos el diseño de arquitecturas heterogéneas que desacoplen la extracción de características mediante hardware reprogramable (CGRA) y empleen técnicas de emparejamiento morfológico (*pattern matching*) como DTW que se pueden acelerar mediante la integración de dispositivos específicos. Este enfoque híbrido no solo facilita la explicabilidad clínica, sino que permite una personalización inmediata (*one-shot*) con un consumo energético reducido, superando las limitaciones de autonomía y robustez de las propuestas actuales.

En concreto, las contribuciones principales de este trabajo se centran en tres aspectos clave. En primer lugar, a nivel de co-diseño *hardware-software*, se presenta la adaptación y mapeo del algoritmo explicable PaFESD sobre una arquitectura SoC RISC-V heterogénea. Esta implementación delega la extracción de características a un dispositivo reconfigurable (CGRA) y propone la instanciación de un acelerador ASIC dedicado para el emparejamiento morfológica basado en DTW. En segundo lugar, para sustentar este despliegue con garantías clínicas, se aborda el reto algorítmico derivado de la drástica reducción espacial usando solo canales laterales de la señal. Para ello, se formula una función de coste continua para el entrenamiento *offline* que, mediante la composición de funciones, logra satisfacer las propiedades de optimización deseadas para mitigar esta merma y recuperar la sensibilidad diagnóstica. Finalmente, la integración de estos avances demuestra empíricamente factores de aceleración masivos, hasta 12× en el dispositivo ASIC y 8× en la CGRA, frente a la ejecución secuencial en el microcontrolador. Esta holgura computacional es la que habilita una agresiva reducción del ciclo de trabajo (*race-to-sleep*), confirmando la viabilidad del sistema para operar de forma ininterrumpida (*always-on*) con un consumo energético ultrabajo.

II. Estado del arte

A. Algoritmos de detección de crisis epilépticas

Tradicionalmente, la detección de crisis epilépticas dependía de la inspección visual o de algoritmos lineales simples, pero en los últimos años el paradigma se ha desplazado masivamente hacia el Aprendizaje Profundo (*Deep Learning* o DL) y el Aprendizaje Automático (*Machine Learning* o ML). Revisiones sistemáticas recientes revelan que más del 70 % de las investigaciones actuales emplean arquitecturas de DL puro, tales como Redes Neuronales Convolucionales (CNN) y redes recurrentes como LSTM [9, 17]. Se han propuesto diversos modelos sofisticados, incluyendo CNNs híbridas 1D y arquitecturas basadas en *Transformers*, que alcanzan precisiones teóricas muy elevadas sobre conjuntos de datos públicos estándar [13, 18, 19].

No obstante, estos enfoques enfrentan barreras críticas para su adopción clínica en el mundo real. Principalmente, adolecen de una severa falta de interpretabilidad clínica (fenómeno de la «caja negra»), lo cual merma la confianza del especialista médico, y presentan una complejidad computacional que eleva el consumo energético al rango de los milivatios (50-100 mW), inviable para monitorización ininterrumpida. Además, existe un marcado sobreajuste (*overfitting*) hacia bases de datos clínicas altamente controladas y predominantemente pediátricas (como el clásico dataset CHB-MIT). Cuando estos modelos se exponen a escenarios ambulatorios reales en formato *wearable*, su rendimiento suele decaer drásticamente debido a la baja relación señal-ruido, los artefactos de movimiento y, especialmente, el ruido muscular electromiográfico (EMG) [12].

Para abordar el alto coste computacional de procesar la señal de forma continua, han surgido marcos jerárquicos de dos etapas (*Two-Stage Frameworks*). Estos sistemas emplean un predictor primario muy ligero que opera constantemente, activando un clasificador secundario más profundo y preciso únicamente cuando se detecta actividad sospechosa o preictal [20, 21]. Otras líneas de investigación proponen la fusión multimodal de sensores (como EEG, PPG y acelerometría) para compensar el ruido mediante la redundancia de datos fisiológicos [22], o incluso el uso de Espectroscopía Funcional de Infrarrojo Cercano (fNIRS) ensanchando la distancia fuente-detector [23].

B. Arquitecturas hardware para el edge

La traslación de los algoritmos descritos a entornos ambulatorios exige su despliegue en nodos de borde (*edge computing*), priorizando la ultra-baja potencia. La literatura reciente exhibe múltiples enfoques competitivos para este fin. En el ecosistema de los microcontroladores, el enfoque *TinyML* y el aprendizaje comprimido (*Compressed Learning*) intentan incrustar redes neuronales ligeras ajustando recursos de memoria. Buscando mayores picos de eficiencia, se ha

propuesto el desarrollo de ASIC a medida, aceleradores en plataformas RISC-V para inferencia de CNNs, y computación neuromórfica mediante Redes Neuronales de Pulsos (SNN) que explotan la dispersión espaciotemporal de la señal. En el extremo opuesto, los detectores de «complejidad ultra-baja» sustituyen las redes por umbrales estadísticos simples, mientras que arquitecturas *Zero-Shot* abordan el «arranque en frío» en nuevos pacientes.

C. Limitaciones del estado del arte y solución propuesta

Al contrastar las soluciones del estado del arte, se hace evidente una importante brecha entre la teórica viabilidad algorítmica y los estrictos requerimientos de un escenario clínico real. Por un lado, las propuestas de *Deep Learning* proporcionan una inmensa sensibilidad *in silico*, pero, como se ha expuesto, su inherente opacidad («cajas negras») y su ineficacia frente a los artefactos electromiográficos (EMG) limitan severamente su adopción ambulatoria [9, 12]. Inversamente, los diseños de hardware de complejidad ultra-baja logran la excelencia en consumos de nanovatios, pero a expensas de la robustez, produciendo tasas de falsas alarmas clínicamente inaceptables [14].

Nuestra propuesta busca converger estas dos vertientes mediante un diseño simbiótico. Inspirándonos en la aproximación algorítmica de Muñoz et al. [24], el paradigma propuesto plantea fusionar la extracción de características eficiente en recursos con el emparejamiento morfológico de la señal de ataque (*pattern matching*). A diferencia de las topologías de redes profundas opacas, fundamentar la detección continua en el uso de *Dynamic Time Warping* (DTW) pretende obtener un sistema inherentemente explicable. Esta aproximación está orientada a suministrar a los neurólogos una traza visual justificable del evento neurofisiológico, aspirando a una mayor tolerancia al ruido electromiográfico que demuestra [24] en comparación con modelos puramente espectrales. Al mapear este algoritmo sobre una arquitectura de *edge computing* modular (como la plataforma X-HEEP), la solución propuesta busca optimizar el compromiso entre latencia y autonomía.

Mediante el uso de aceleradores específicos para la extracción de características, se pretende implementar una estrategia de procesamiento intermitente eficiente (*duty cycling* agresivo). El objetivo es que el sistema mantenga una vigilancia continua (*always-on*) operando en estados de ultra-bajo consumo la mayor parte del tiempo, despertando los recursos de computación paralela únicamente para la validación morfológica mediante DTW cuando se detecten señales candidatas de crisis. Esta gestión inteligente de los estados de reposo busca reducir drásticamente la energía, aspirando a niveles de especificidad clínica que las soluciones de umbral estático no pueden alcanzar.

III. Adaptaciones algorítmicas para la implementación en un wearable

La literatura reciente evidencia que los algoritmos de detección de anomalías neurofisiológicas comparten un flujo de datos (*pipeline*) estandarizado. Tras la adquisición de la señal (típicamente entre 128 Hz y 1 kHz), la etapa de preprocesamiento busca maximizar la relación señal-ruido mediante la selección restrictiva de canales, el filtrado frecuencial (filtros IIR/FIR paso-banda) y la eliminación de artefactos empleando referenciado espacial o separación ciega de fuentes. Para la inferencia en nodos de borde (*edge*), la estrategia más eficiente energéticamente frente al procesamiento en crudo de redes profundas es la extracción explícita de características en múltiples dominios (temporal, frecuencial y espacial). Finalmente, tras aplicar técnicas de reducción de dimensionalidad, un clasificador evalúa estas métricas y una capa de postprocesamiento consolida las predicciones por ventana temporal para emitir alertas clínicas robustas y mitigar la fatiga por falsas alarmas.

Para materializar este flujo de procesamiento continuo bajo estrictas restricciones de recursos, este trabajo toma como base algorítmica PaFESD (*Patterns augmented by Features Epileptic Seizure Detection* [24]), un enfoque híbrido y explicable que sigue una estrategia diferente a la de los modelos opacos de *Deep Learning*. Su implementación original para procesadores de propósito general asume un uso de canales de señal denso (sistema 10-20 [4]). Para habilitar su despliegue en un dispositivo *wearable* discreto, forzamos una drástica reducción espacial a canales puramente laterales (ej. T7-FT9, F8-T8). Esta pérdida de resolución exige adaptar el preprocesamiento y explorar nuevas características para mantener el F_1-*score* original del 98.9 %.

A. Preprocesamiento y extracción de características

La primera etapa elimina artefactos no estacionarios mediante el *Riemannian Potato Field* (RPF), que caracteriza las ventanas temporales mediante matrices de covarianza y evalúa su distancia Riemanniana afín-invariante respecto a un centroide limpio. La restricción espacial obligó a reconfigurar la dimensionalidad de los detectores originales hacia subredes fronto-laterales y parieto-temporales. Su evaluación genera un vector binario (*Do-Not-Compute*, DNC) que excluye ventanas (épocas) de señal contaminadas del cómputo siguiente. La señal validada se filtra (1.5 a 36 Hz) y se segmenta en ventanas *hardware-friendly* de 4 segundos con un paso (*stride*) de 1 segundo.

A continuación, se extraen características de bajo coste computacional que actúan como filtro fisiológico temprano: amplitud *peak-to-peak*, energía (calculada como la varianza de la época) y la potencia integrada en tres bandas frecuenciales (PSD calculada vía Welch y Simpson). Para compensar la merma espacial lateral que impone nuestro wearable, se integra ahora la longitud de línea (*line-length*) como nueva característica. Esta métrica fractal se computa eficientemente sumando las diferencias absolutas entre mues-

tras consecutivas de la época ($L = \sum_{i=1}^{N-1} |x_{i+1} - x_i|$), enriqueciendo el espacio de decisión sin requerir multiplicaciones complejas. Si alguna de estas métricas cae fuera de los límites precalculados del paciente, la ventana se etiqueta en el vector DNC y se descarta.

B. Clasificación morfológica y función de optimización continua

Las ventanas que superan el filtro fisiológico se validan evaluando su similitud morfológica contra un conjunto (*batch*) de patrones de crisis almacenados mediante *Dynamic Time Warping* con ventana de Sakoe-Chiba (cDTW). Para optimizar el cómputo, la distancia euclídea que se usaba en el algoritmo original se sustituye ahora por el valor absoluto de la diferencia.

La determinación *offline* de los límites de las características y de los patrones óptimos emplea el optimizador de Lipschitz (LIPO). El algoritmo original dependía exclusivamente de maximizar el *Discrimination Ratio* (DR). Sin embargo, su rango no acotado $[0, \infty)$ y comportamiento abrupto generaban mesetas (*plateaus*) que estancaban la búsqueda.

Para acelerar la exploración global, proponemos ahora una nueva función de coste continua. Esta evalúa simultáneamente el DR y dos tasas probabilísticas: la retención armónica de crisis (H_{ret}) y el rechazo armónico interictal (H_{rej}). Estas métricas de rangos dispares se mapean a un espacio simétrico $[-1, 1]$, centrado en un objetivo de diseño (T):

$$N_{DR} = \tanh\left(S_{DR} \cdot (\ln(DR) - \ln(T_{DR}))\right) \quad (1)$$

$$N_H = \tanh\left(S_H \cdot \left(\ln\left(\frac{H}{1-H}\right) - \ln\left(\frac{T_H}{1-T_H}\right)\right)\right) \quad (2)$$

donde S actúa como un factor de nitidez (*sharpness*) que controla la pendiente de la transición alrededor del objetivo. Si la métrica evaluada alcanza el objetivo, la función devuelve 0. Para evitar que el optimizador priorice una métrica sobre el resto, las tres métricas normalizadas (N_{DR}, N_{ret}, N_{rej}) se combinan mediante una función *softmin* (Ec. 3), empujando a mejorar siempre el peor desempeño.

$$M_{raw} = -\frac{1}{S_{SM}} \ln\left(e^{-S_{SM} \cdot N_{DR}} + e^{-S_{SM} \cdot N_{ret}} + e^{-S_{SM} \cdot N_{rej}}\right) \quad (3)$$

Para garantizar la estabilidad numérica computacional se emplea el *Log-Sum-Exp trick*. Finalmente, el valor M_{raw} se proyecta mediante la ecuación 4, para ofrecer una métrica final (M_{final}) estrictamente acotada en el rango $[0, 1]$.

$$M_{final} = \frac{M_{raw} - \left(-1 - \frac{\ln(3)}{S_{SM}}\right)}{2} \quad (4)$$

Esta reformulación transforma el espacio de búsqueda en un paisaje suave y estrictamente acotado, permitiendo al optimizador LIPO explorar los umbrales sin degradarse a búsquedas aleatorias. Para casos extremos donde se filtren accidentalmente todas las ventanas de no-crisis, una heurística de respaldo selecciona el *batch* que minimice la distancia máxima a las ventanas de crisis restantes, garantizando la operatividad del dispositivo bajo condiciones límite.

IV. Arquitectura del Sistema (SoC HEEPsilon)

Para materializar el algoritmo de detección continua bajo estrictas restricciones de ultra-bajo consumo, este trabajo toma como punto de partida el SoC HEEPsilon [25]. Esta arquitectura emplea un paradigma de computación heterogénea que proponemos extender mediante la integración de un acelerador especializado para el cómputo de la cDTW. El objetivo es equilibrar eficientemente el flujo de control y el procesamiento intensivo de datos, maximizando la autonomía del dispositivo.

A. Plataforma Base X-HEEP y SoC HEEPsilon

El núcleo del sistema se fundamenta en HEEPsilon, una evolución de la plataforma de código abierto X-HEEP (*eXtendable Heterogeneous Energy-Efficient Platform* [25]) basada en la arquitectura RISC-V. En nuestro diseño, el procesador no actúa de forma exclusiva como orquestador, sino que asume un rol activo en el procesamiento. Además de gestionar el flujo de datos y tareas, configurar periféricos y programar los accesos a memoria mediante controladores de acceso directo (DMA), el núcleo RISC-V, programado en C, balancea la carga computacional colaborando con la arquitectura reconfigurable CGRA en las etapas iniciales del algoritmo durante la extracción de las características, y descargando la computación de la cDTW en el ASIC cuando las etapas de detección previas han detectado un potencial ataque.

B. Aceleración mediante CGRA

El cálculo de las características físicas de la señal requiere flexibilidad para adaptarse a futuros biomarcadores. HEEPsilon aporta esta versatilidad integrando de base una *Coarse-Grained Reconfigurable Array* (CGRA) de 4x4 *Processing Elements* (PE). A nivel de sistema, este acelerador se encuentra estrechamente acoplado a los bancos de memoria SRAM del SoC, lo que permite el acceso directo y concurrente a los datos minimizando la intervención del núcleo principal. A nivel físico, el tejido de la CGRA opera con aritmética entera de 32 bits (`int32`) e incorpora soporte nativo para operaciones en punto fijo Q16.15. Dado que el desarrollo de compiladores de C convencionales para este hardware es un trabajo en curso, la CGRA se programa mediante pseudoensamblador, definiendo macroinstrucciones compuestas por subinstrucciones extraídas del repertorio (ISA) de la propia arquitectura del dispositivo.

Para ilustrar este paradigma de computación espacial, la Figura 1 expone el grafo resultante de mapear el *kernel* de la amplitud *peak-to-peak* sobre la CGRA. En este esquema de dependencias, cada nodo circular representa una microinstrucción instanciada en un PE, mientras que la codificación por colores identifica la macroinstrucción a la que pertenecen. El flujo visualiza claramente las fases de ejecución: un bloque de inicialización (nodos azules) que alimenta a un *software pipeline* optimizado (nodos verdes y rojos). Dicho *pipeline* logra un intervalo de iniciación

(II) de 2 ciclos y explota un nivel de paralelismo de datos de 2×, procesando dos muestras del *epoch* simultáneamente en cada iteración (evidenciado por las dos instrucciones de carga de memoria con direccionamiento calculado `LWI`). Finalmente, el flujo converge en la etapa de finalización (nodo amarillo) para emitir el resultado.

La capacidad de aceleración de la CGRA se evidencia aún más en la extracción de métricas estadísticas. La implementación del *kernel* de la media logra un mapeo óptimo con un II=1 y un paralelismo masivo de 4×, saturando eficientemente el ancho de banda de lectura de la CGRA. Seguidamente, el cómputo de la energía de la señal se resuelve alcanzando un II=2 con un paralelismo sostenido de 4×.

Durante la fase de exploración del diseño, se evaluó la implementación del método estadístico *online* de Welford, una alternativa algorítmica clásica que permite computar la media y la varianza en una única iteración temporal. Si bien esta técnica es idónea para arquitecturas secuenciales (CPU) orientadas a minimizar los accesos a memoria, su traslación a un paradigma de computación espacial resultó deficiente. Las estrictas dependencias de datos inter-iteración inherentes a las ecuaciones de Welford limitaron el mapeo en la CGRA, con un cuello de botella de II=4 y con nulo paralelismo de datos (1×). Por consiguiente, nuestra arquitectura final consolidó el diseño en dos pasadas independientes (*two-pass*), demostrando que, la maximización del paralelismo espacial compensa holgadamente el coste de releer las ventanas de datos, filtrando la señal de forma de no ataque significativamente más eficiente.

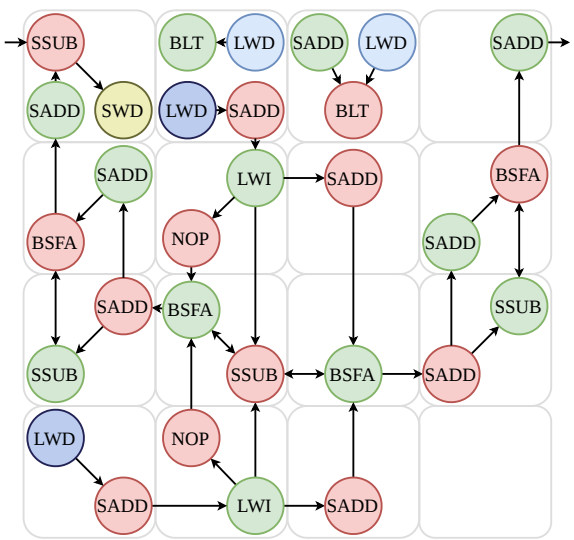


Fig. 1: Mapeo espacial del *kernel peak-to-peak* sobre la CGRA de 4x4 PEs. Los nodos (microinstrucciones) se agrupan por colores según su macroinstrucción: inicialización (azul), cuerpo del *pipeline* con paralelismo 2× e II=2 (verde y rojo), y finalización (amarillo).

C. Extensión con Acelerador ASIC para DTW

El núcleo de clasificación morfológica (cDTW) sobre ventanas de 1024 muestras es algorítmicamente ejecutable en tiempo real por el procesador del sistema, incluso cuando se compara contra un conjunto de hasta cuatro patrones simultáneamente. Sin embargo, esta tarea constituye el principal cuello de botella energético. Para resolverlo, proponemos extender el SoC HEEPsilon con un acelerador ASIC de función fija dedicado de forma exclusiva a la métrica cDTW, cuya microarquitectura se detalla en la Figura 2.

A nivel de diseño, el acelerador ha sido modelado utilizando Síntesis de Alto Nivel (HLS) en C++, lo que agiliza su exploración arquitectónica antes de la transición a silicio. La microarquitectura diseñada es fuertemente parametrizable en tiempo de compilación, permitiendo ajustar estáticamente variables clave como el tamaño de la época, la anchura de la banda de restricción de Sakoe-Chiba, y la cantidad de elementos de procesamiento. Esta flexibilidad paramétrica resulta vital para adaptar con precisión el hardware a las estrictas limitaciones de área y potencia del dispositivo *wearable* objetivo.

En el contexto de este SoC de ultrabajo consumo, el acelerador se instancia como una única Unidad de Cómputo (CU) que orquesta internamente una cadena de PEs que trabajan concurrentemente en la comparación de las señales entrantes que representan potenciales ataques con patrones de ataques previos. Para maximizar el rendimiento, la arquitectura abandona el enfoque sistólico tradicional donde múltiples PEs colaboran en una única matriz para el cómputo de la cDTW; en su lugar, cada PE opera de forma completamente independiente sobre distintas ventanas de la señal entrante y un conjunto de patrones conocidos. Asimismo, el *datapath* de cada elemento ha sido optimizado para evitar el almacenamiento completo de la matriz de costes intermedia, reteniendo en memoria local rotativa únicamente los valores de la banda activa de Sakoe-Chiba. Combinado con la sustitución de la distancia euclídea por el valor absoluto de la diferencia, el diseño elimina la necesidad de multiplicadores hardware y minimiza drásticamente el movimiento de datos interno, lo que minimiza el consumo dinámico y el área.

El objetivo fundamental de esta instanciación directa en silicio no es habilitar el tiempo real, que ya es factible por software en la CPU, sino acelerar drásticamente el cómputo. Al resolver la DTW en una fracción del tiempo original, el acelerador permite que el SoC pueda implementar una política de *race-to-sleep* (ejecución rápida para retorno temprano al reposo), con el consiguiente ahorro energético.

D. Estrategia de gestión de energía

Para materializar esta política de *race-to-sleep* y garantizar la viabilidad de la monitorización ininterrumpida (*always-on*), la arquitectura implementa un despertar progresivo (*wake-up*) guiado por eventos. Por defecto, el SoC permanece en modo de ultrabajo consumo (*sleep*). Cuando se captura una nueva ventana temporal (época), el DMA despierta al conjunto compuesto por el procesador y la CGRA para calcular las características físicas de forma colaborativa. Si

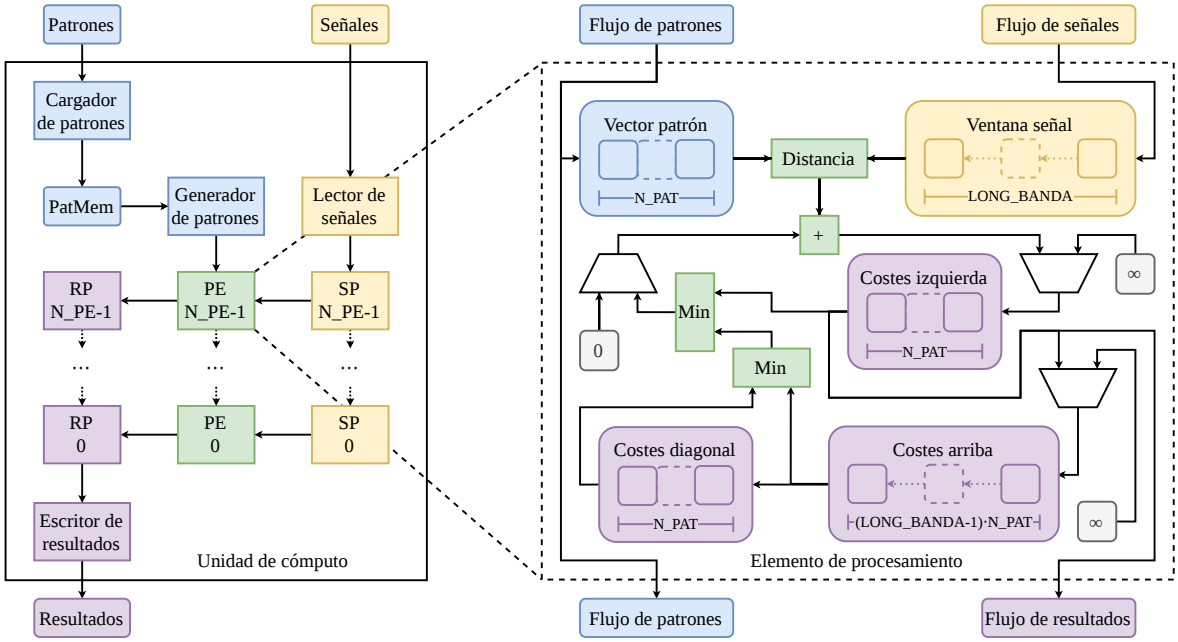


Fig. 2: Arquitectura del acelerador cDTW.

el resultado de estas métricas activa el vector DNC (filtrado de señal de no ataque), el procesamiento se aborta y el sistema retorna al reposo inmediatamente, ahorrando un tiempo de computación crítico. Únicamente si la ventana sobrevive a este filtro temprano, el sistema enciende el acelerador ASIC para resolver el emparejamiento morfológico basado en la computación de la cDTW que compara las señales entrantes de potenciales ataques con un conjunto de patrones de ataques previos. Esta orquestación jerárquica garantiza que el dispositivo de mayor consumo dinámico solo se alimente cuando es estrictamente necesario, maximizando la autonomía del dispositivo.

V. Evaluación experimental y resultados

Para evaluar la viabilidad de la arquitectura propuesta, se ha establecido una comparativa progresiva partiendo del algoritmo PaFESD original (diseñado para montajes clínicos de 18 canales de señal) hacia una versión *Restringida* (limitada a 7 canales laterales: T7-FT9, T7-P7, F7-T7, FT9-FT10, F8-T8, T8-P8 y FT10-T8) y, finalmente, a una versión *Modificada* que incorpora las optimizaciones matemáticas descritas en la Sección III.

A nivel de configuración experimental, el modelo modificado asume la reducción del tamaño de ventana (época) a 1024 muestras (4 seg.) para su alineación hardware. Asimismo, el conjunto de patrones de referencia para la cDTW se fijó estrictamente a un único patrón óptimo por paciente. Para compensar la pérdida de crisis de corta duración en este nuevo formato, el margen de descarte temporal en las zonas intericitales se ajustó de 10 a 8 segundos.

Respecto a la adaptación espacial del filtro RPF, el detector de parpadeo se transpuso a canales frontolaterales (F7-T7, F8-T8, T7-FT9), el occipital se mapeó a un montaje de canales de señal parieto-temporal (T7-P7, T8-P8) manteniendo la norma de la traza activa, y el detector global se ajustó a la subred de 7 canales laterales. Para recuperar la precisión diagnóstica bajo estas restricciones, la versión modificada amplió la banda de restricción de Sakoe-Chiba de la distancia cDTW de 16 muestras al 10 % del tamaño de la época (102 muestras) e incorporó la longitud de línea (*line-length*) como nueva característica. Los umbrales de este conjunto ampliado de características se ajustaron empleando la nueva función de coste *softmin* previamente detallada.

A. Rendimiento de clasificación

Los resultados experimentales (Figura 3) evidencian el impacto de las modificaciones propuestas. Si bien la drástica reducción a canales laterales impone una caída severa del rendimiento global respecto al modelo original ideal, la arquitectura modificada apoyada en el optimizador *softmin* logra mitigar esta degradación y elevar la media de detección. Más allá de esta mejora promedio, la verdadera fortaleza de la propuesta reside en su capacidad para extraer características discriminativas en escenarios de alta complejidad, estabilizando el sistema. Cuantitativamente, se logran incrementos sustanciales en pacientes que resultaban críticos para el modelo base; destaca especialmente el rescate del paciente chb24, que pasa de un rendimiento nulo ($F_1 = 0$) a un sólido 0.81, así como los pacientes chb01 y chb12, con mejoras relativas del 78 % (de 0.50 a 0.89) y del 138 % (de 0.31 a 0.74), respectivamente. Resulta especialmente relevante que esta superioridad se manifieste incluso operando sobre el mismo canal electroencefalográfico óptimo, como se observa en el paciente chb05, donde el F_1 se eleva de 0.66 a un 1.0 perfecto. Esto demuestra que la nueva función objetivo dota al modelo de una mayor robustez matemática intrínseca para caracterizar las crisis, reduciendo su dependencia de una ubicación espacial ideal.

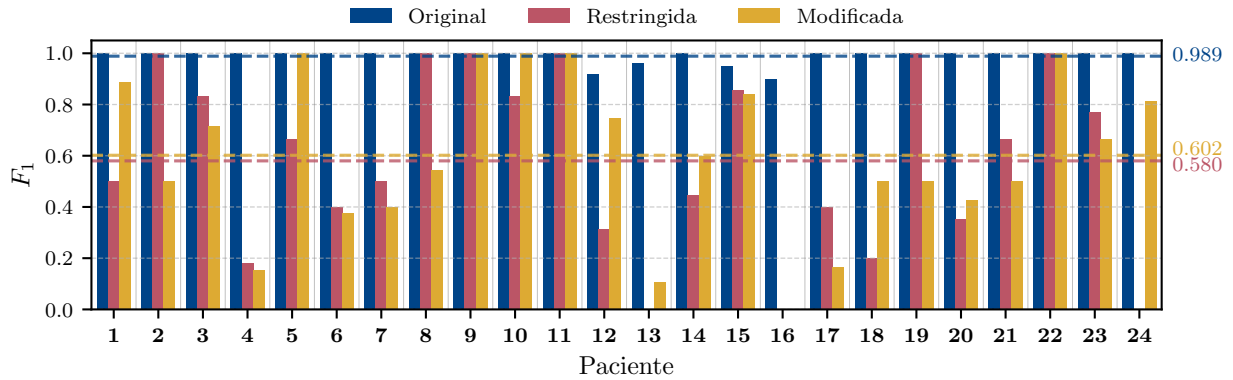


Fig. 3: Comparativa de la métrica F_1 entre los distintos entrenamientos para cada paciente. Las líneas horizontales discontinuas indican el F_1 medio global de cada aproximación.

No obstante, este aumento en la sensibilidad ante casos difíciles impone un claro compromiso de diseño, penalizando la estabilidad del modelo en pacientes con perfiles de crisis previamente triviales. En sujetos como chb02, chb08 y chb19, donde la arquitectura *Restringida* lograba una clasificación perfecta ($F_1 = 1$), la versión *Modificada* sufre una degradación severa, cayendo a valores en torno a 0.50 y 0.55. Este fenómeno sugiere que los nuevos mecanismos de la arquitectura, y en particular la penalización estricta impuesta por la función *softmin*, fuerzan al optimizador a buscar un compromiso global en el espacio de características. En pacientes con crisis focales simples y muy marcadas, este desplazamiento forzoso de los límites de decisión resulta excesivamente agresivo, induciendo al sistema a descartar ventanas válidas que antes superaban holgadamente los umbrales. Asimismo, la persistencia de fallos absolutos en casos como el paciente chb16 ($F_1 = 0$ en ambas versiones restringidas) confirma una limitación ineludible del análisis con baja densidad de canales; existen crisis cuya firma espacial exige estrictamente la captura de información topológica distribuida, marcando el límite de simplificación del hardware de monitorización lateral. Ambas limitaciones serán abordadas en el trabajo futuro.

B. *Carga computacional*

Para cuantificar el impacto en la eficiencia, se midió la latencia de inferencia por ventana (Figura 4). La evaluación del MCU y la CGRA se realizó mediante simulación RTL con Verilator. Los ciclos de reloj resultantes se transformaron a unidades temporales asumiendo una frecuencia de operación de 300 MHz para el núcleo RISC-V, representativo del rendimiento base de la arquitectura X-HEEP subyacente en tecnología TSMC de 65 nm [25]. Por su parte, los tiempos de ejecución del acelerador cDTW dedicado se obtuvieron mediante su despliegue físico en una plataforma ZedBoard (tecnología de 28 nm) operando a 142.8 MHz.

El mapeo de la extracción de características elementales (como la amplitud *peak-to-peak*, la energía y la media) en la CGRA demuestra una mejora drástica frente a la ejecución secuencial en el MCU. Como se

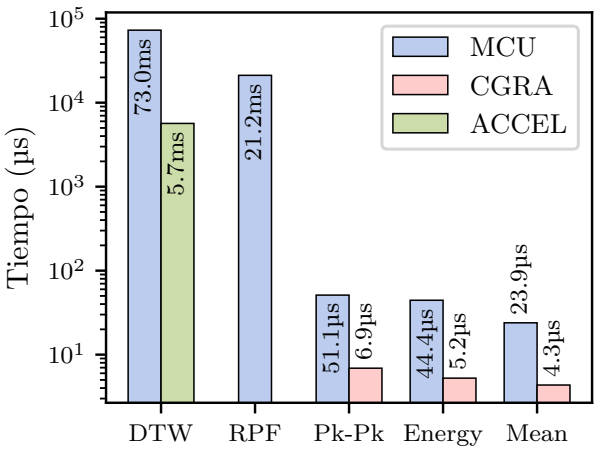


Fig. 4: Rendimiento computacional de las distintas etapas del algoritmo, representado en escala logarítmica.

aprecia en la Figura 4, la aceleración de estas métricas oscila sistemáticamente entre factores de 5× y más de 8×. Esta reducción de tiempo en el *front-end* del algoritmo es relevante para acortar el tiempo en activo del dispositivo, permitiendo que el dominio de potencia de la CGRA retorne rápidamente a estados de reposo (*sleep*) lo que minimiza el consumo de energía en las fases tempranas del *pipeline*.

En la etapa de emparejamiento morfológico, la ejecución por *software* de la cDTW en el núcleo RISC-V domina por completo el presupuesto de tiempo de inferencia. Al delegar esta tarea al acelerador ASIC, la latencia se desploma, logrando un factor de aceleración superior a un orden de magnitud ($> 12\times$). Resulta destacable que esta ganancia masiva de rendimiento se obtiene operando el silicio dedicado a menos de la mitad de la frecuencia de reloj del procesador principal, evidenciando la superioridad arquitectónica del acelerador para resolver la complejidad cuadrática de esta métrica.

Para traducir esta ganancia de latencia en métricas de eficiencia energética, se ha realizado una estimación teórica del consumo combinando los perfiles de potencia reportados para la arquitectura subyacente. Asumiendo un factor de actividad dinámico de 28 µW/MHz [26] para el núcleo operando a 300 MHz, la potencia activa del sistema asciende a unos 8.43 mW (cifra que incluye 29 µW de fugas estáticas). En

contraparte, cuando el dispositivo transiciona a su estado de reposo profundo (*deep sleep*), el consumo estático cae a 3 µW [25]. El modelo ejecutado íntegramente por *software* requiere mantener el sistema activo durante 94.32 ms cada segundo, lo que arroja un consumo de potencia medio de aproximadamente 798 µW (sumando la energía dinámica de cálculo y las pérdidas estáticas durante el reposo restante). Por el contrario, al delegar la carga a la CGRA y al acelerador dedicado, el tiempo en estado activo se desploma a 26.92 ms por segundo. Al permitir que el sistema retorne al estado de bajo consumo mucho antes, el gasto promedio cae a 230 µW. Cabe destacar que estas estimaciones representan el escenario de mayor carga computacional (peor caso), presuponiendo que la ventana de señal atraviesa todas las etapas de clasificación sin aplicar ninguna política de salida temprana (*early-exit*). Bajo estas condiciones de máxima exigencia, en términos absolutos, la aceleración hardware propuesta logra que el sistema sea 3.47× más eficiente energéticamente frente a la aproximación mononúcleo convencional.

Como conclusión arquitectónica, este trasvase masivo de carga computacional hacia el ASIC y la CGRA altera el cuello de botella del sistema. Aunque una integración *full-custom* del acelerador en el mismo silicio que el SoC afinaría las mediciones absolutas cruzadas, los resultados actuales evidencian que el filtro RPF, al ejecutarse íntegramente por *software* en el MCU, pasa a ser la etapa limitante del flujo de datos. En consecuencia, el trabajo futuro se enfocará en el mapeo de la estimación de matrices de covarianza del RPF hacia la CGRA o, en su defecto, en la instanciación de un nuevo coprocesador dedicado. Esto permitiría aplicar algoritmos de covarianza más sofisticados sin degradar la premisa de ultrabajo consumo del sistema *always-on*.

VI. Conclusiones y Trabajo Futuro

Este trabajo presenta el diseño y evaluación de un algoritmo de detección de crisis epilépticas en HEEPsilon, una arquitectura de procesamiento heterogénea orientada a la monitorización continua sobre *wearables*. La principal conclusión es que la estrategia de co-diseño *hardware-software* ha superado holgadamente la barrera del tiempo real. Al derivar la carga intensiva del emparejamiento morfológico (cDTW) a un acelerador ASIC dedicado, así como la extracción de características físicas a una CGRA, se han logrado factores de aceleración masivos de hasta 12× y 8×, respectivamente. Esto permite al sistema implementar una política agresiva de reducción del ciclo de trabajo (*race-to-sleep*), lo que consigue una mejora de la eficiencia energética en un factor 3.47×. De este modo, se demuestra que el desafío fundamental en los sistemas *always-on* actuales ya no es la latencia de inferencia, sino la minimización extrema del consumo energético.

A nivel algorítmico y clínico, se ha evidenciado que la restricción a canales laterales discretos impone un severo compromiso espacial. Aunque la introducción de una función objetivo continua (*softmin*) en la etapa de entrenamiento ha demostrado ser capaz de recuperar la sensibilidad en pacientes históricamente críticos para el modelo restringido, la inestabilidad inducida en casos previamente triviales confirma que compensar la pérdida de resolución topográfica sigue siendo un desafío abierto.

Al tratarse de una investigación en desarrollo, el trabajo futuro se bifurca en optimizaciones de arquitectura y refinamientos algorítmicos. En el plano físico, queda pendiente la implementación completa en el microcontrolador de las características basadas en la Densidad Espectral de Potencia (PSD). Se estima que la latencia de estas métricas se situará por debajo del filtro de artefactos RPF, pero significativamente por encima de las características temporales simples. Por consiguiente, el siguiente hito será mapear la transformada rápida de Fourier (FFT) y la integración de Simpson directamente sobre el tejido de la CGRA. Simultáneamente, se explorarán nuevos algoritmos para la estimación de matrices de covarianza que forman parte del detector de artefactos RPF.

En la vertiente algorítmica, se ampliará el espacio de decisión explorando métricas de complejidad no lineal, como la entropía aproximada o la entropía difusa. Para evitar que esta expansión colapse la capacidad de convergencia del optimizador LIPO, se aplicará un estudio estadístico previo de las distribuciones poblacionales de cada característica. En los casos donde la distribución de las ventanas de crisis se solape parcialmente pero mantenga una direccionalidad estricta respecto a las no-crisis (ej. valores consistentemente superiores), el optimizador fijará un único límite de búsqueda en lugar de dos. Esta técnica reducirá drásticamente la dimensionalidad del problema, permitiendo explorar soluciones de mayor calidad con el mismo presupuesto de iteraciones.

Finalmente, para incrementar la robustez clínica frente a artefactos sin incurrir en el coste estigmatizante de añadir más canales EEG, se plantea la integración de señales biológicas complementarias de bajo consumo, como la variabilidad de la frecuencia cardíaca o un canal de ECG simple. El objetivo último a largo plazo será la fabricación en silicio del SoC completo para validar las métricas de potencia física y la evaluación del sistema consolidado sobre bases de datos puramente ambulatorias.

Agradecimientos

Este trabajo ha sido financiado mediante los proyectos de investigación TED2021-131527B-I00, PID2022-136575OB-I00 y TSI-069100-2023-0013 del Ministerio de Ciencia, Innovación del Gobierno de España.

Referencias

[1] Robert S. Fisher et al., "Ilae official report: A practical clinical definition of epilepsy," *Epilepsia*, vol. 55, no. 4, pp. 475–482, 2014.

[2] Jose F. Téllez-Zenteno and Lizbeth Hernández-Ronquillo, "A review of the epidemiology of temporal lobe epilepsy," *Epilepsy Research and Treatment*, vol. 2012, no. 1, pp. 630853, 2012.

[3] Sándor Beniczky et al., "Updated classification of epileptic seizures: Position paper of the international league against epilepsy," *Epilepsia*, vol. 66, no. 6, pp. 1804–1823, 2025.

[4] Ray Cooper et al., "Comparison of subcortical, cortical and scalp activity using chronically indwelling electrodes in man," *Electroencephalography and Clinical Neurophysiology*, vol. 18, no. 3, pp. 217–228, 1965.

[5] Steven V. Pacia and John S. Ebersole, "Intracranial eeg substrates of scalp ictal patterns from temporal lobe foci," *Epilepsia*, vol. 38, no. 6, pp. 642–654, 1997.

[6] Martin E. Weinand et al., "Time from ictal subdural eeg seizure onset to clinical seizure onset: an electrocorticographic time factor associated with temporal lobe epileptogenicity," *Neurological Research*, vol. 29, no. 8, pp. 862–870, 2007, PMID: 17601365.

[7] Mark J Cook et al., "Prediction of seizure likelihood with a long-term, implanted seizure advisory system in patients with drug-resistant epilepsy: a first-in-man study," *The Lancet Neurology*, vol. 12, no. 6, pp. 563–571, 2013.

[8] Brian Litt et al., "Epileptic seizures may begin hours in advance of clinical onset: A report of five patients," *Neuron*, vol. 30, no. 1, pp. 51–64, 2001.

[9] Yitong Wu et al., "A review of machine learning and deep learning trends in eeg-based epileptic seizure prediction," *IEEE Access*, vol. 13, pp. 159812–159842, 2025.

[10] Julia Werner et al., "Energy-efficient seizure detection suitable for low-power applications," in *2024 International Joint Conference on Neural Networks (IJCNN)*. June 2024, p. 1–8, IEEE.

[11] Moez Hizem et al., "Sustainable e-health: Energy-efficient tiny ai for epileptic seizure detection via eeg," *Biomedical Engineering and Computational Biology*, vol. 16, pp. 11795972241283101, 2025.

[12] Miguel Bhagubai et al., "Towards automated seizure detection with wearable eeg – grand challenge," *IEEE Open Journal of Signal Processing*, vol. 5, pp. 717–724, 2024.

[13] Swathy Ravi and Ashalatha Radhakrishnan, "A hybrid 1d cnn-bilstm model for epileptic seizure detection using multichannel eeg feature fusion," *Biomedical Physics & Engineering Express*, vol. 10, no. 3, pp. 035040, 4 2024.

[14] Andrea Vittimberga, Giovanni Nicolini, and Giuseppe Scotti, "Real-time multi-channel epileptic seizure detection exploiting an ultra-low-complexity algorithm–hardware co-design approach," *Sensors*, vol. 25, no. 22, 2025.

[15] Shuenn-Yuh Lee et al., "Reconfigurable and scalable artificial intelligence acceleration hardware architecture with risc-v cnn coprocessor for real-time seizure detection," *IEEE Access*, vol. 13, pp. 31057–31068, 2025.

[16] Shuenn-Yuh Lee et al., "Rvdlaha: An risc-v dla hardware architecture for on-device real-time seizure detection and personalization in wearable applications," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 19, no. 1, pp. 40–54, 2025.

[17] Goldwyn Sudhakar Jebaraj and Konguvel Elango, "A comprehensive review of eeg-based seizure detection techniques," *IEEE Access*, vol. 13, pp. 103531–103564, 2025.

[18] Prabhat Kumar Upadhyay et al., "Seamless integration for enhanced seizure prediction using hybridconvmobilenet on typhoon hil," *Scientific Reports*, vol. 15, no. 1, pp. 45480, 11 2025.

[19] Sanjeev Kumar Varun and Tharun Kumar Reddy Bollu, "Wearneuronet: An interpretable light-weight deep learning approach for ictal–interictal classification for limited channel eeg wearables," *IEEE Sensors Letters*, vol. 8, no. 10, pp. 1–4, 2024.

[20] Siyuan Qiu et al., "A two-stage prediction + detection framework for real-time epileptic seizure monitoring," *IEEE Transactions on Instrumentation and Measurement*, vol. 74, pp. 1–14, 2025.

[21] Dhanalekshmi Prasad Yedurkar et al., "An iot-based novel hybrid seizure detection approach for epileptic monitoring," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 2, pp. 1420–1431, 2024.

[22] Thorir Mar Ingolfsson et al., "Brainfusenet: Enhancing wearable seizure detection through eeg-ppg-accelerometer sensor fusion and efficient edge deployment," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 18, no. 4, pp. 720–733, 2024.

[23] Netaniel Rein et al., "Wide-based illumination and detection in functional near-infrared spectroscopy for enhanced seizure detection in grey matter," *Sensors*, vol. 25, no. 12, 2025.

[24] Felipe Muñoz, Rafael Asenjo Plaza, and Angeles Navarro, "Pafesd: Patterns augmented by features epileptic seizure detection," *IEEE Transactions on Biomedical Engineering*, vol. 72, no. 1, pp. 137–151, 2025.

[25] Simone Machetti et al., "X-heep: An open-source, configurable and extendible risc-v platform for tinyai applications," in *2025 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2025, vol. 1, pp. 1–6.

[26] Pasquale Davide Schiavone et al., "X-heep: An open-source, configurable and extendible risc-v microcontroller," in *Proceedings of the 20th ACM International Conference on Computing Frontiers*, New York, NY, USA, 2023, CF '23, p. 379–380, Association for Computing Machinery.

Compile-Time Reordering for Instruction Fusion

Ravikiran Ravindranath Reddy[1], Sawan Singh[2], Arthur perias[3], Alberto Ros[1,4] y Alexandra Jimborean[1,5]

Resumen— **Hardware instruction fusion combines multiple instructions into a single operation to improve performance and free resources. While fusion usually targets consecutive instructions, fusing non-consecutive ones can offer greater gains but requires complex and costly hardware for prediction and validation.**

This work introduces CAIF (Compiler-Aware Instruction Fusion), a compiler-based technique that schedules instructions to make fusible non-consecutive memory operations appear consecutively.

Experiments show that for neural network workloads, hardware limited to consecutive fusion achieves 1.2 % performance improvement with a standard compiler and 19.6 % with CAIF. With non-consecutive fusion hardware (Helios), CAIF increases performance from 6.6 % to 20.3 %. For general-purpose applications (SPEC CPU 2017), CAIF improves performance from 2.4 % to 6.4 % and from 14.4 % to 17.7 % under the respective hardware setups.

Palabras clave— **Compiler optimization, instruction fusion, neural networks**

I. Introducción

BR inging AI to the edge, IoT, and wearable devices is a key goal in the field nowadays. Deep learning is executed directly on mobile phones, edge nodes, and even highly resource-constrained microcontrollers (MCUs) [22]. On one hand, enabling AI capabilities on these devices greatly enhances cost-efficiency, scalability, and user privacy by performing data processing locally, thus eliminating the need to constantly send sensitive information to the cloud. This not only reduces latency and energy spent sending data wirelessly, but also improves reliability and supports offline operation in environments with limited connectivity.

On the other hand, these devices often face severe resource constraints, lacking the full hardware acceleration or vectorization support necessary for high-performance AI computations. This gap poses a critical challenge: how to deliver efficient AI inference without increasing hardware complexity or power consumption.

Instruction fusion emerges as a compelling solution—a lightweight microarchitectural technique that combines multiple instructions into fused operations, boosting execution efficiency with limited additional hardware overhead.

[1]Dpto. de Informática, Universidad de Murcia, e-mail: ravikiran.r.r@um.es.
[2]AMD Cambridge UK, e-mail: singh.sawan@um.es.
[3]Univ. Grenoble Alpes, TIMA, Grenoble, France, e-mail: arthur.perais@univ-grenoble-alpes.fr.
[4]e-mail: aros@ditec.um.es.
[5]e-mail: alexandra.jimborean@um.es.

Instruction fusion (IF) is a hardware optimization technique used in many modern micro architectures [2, 3, 12]. Instruction fusion combines multiple architectural instructions into a single operation. Improves performance by freeing up slots in Reorder buffer/Store queue/Load Queue for example: (*cmp+br*).

More recently, RISCV Instruction is gaining a lot of traction [4, 8, 39, 40], These works emphasize that (i) fusing both consecutive and non-consecutive instructions is essential for fully exploiting fusion potential [40].

Problem: While consecutive fusion requires minimum hardware support [24], state of the art fusion of non-consecutive instructions entails complex hardware which is more costly [40, 46].

Insight: One requires a code layout in which fusible instructions are placed together, thus converting non-consecutive into consecutive pairs.

Solution: We propose compiler support to identify fusible pairs and reorder instructions to bring fusible non-consecutive memory operations together.

CAIF is implemented in the widely used LLVM compiler framework [18] and demonstrated on neural networks (NNs).

This work offers the following contributions.

- A compiler technique to identify non-consecutive fusible memory operations and their dependencies and place them consecutively through instruction reordering.
- Through simulation, we evaluated the benefits of compiler-aware instruction fusion on various hardware configurations, including Helios [40], a state-of-the-art microarchitecture that enables hardware fusion, and demonstrated performance improvements.

Our experiments on NNs demonstrate that CAIF boosts the fusion of consecutive memory operations from 1.94 % (without compile-time reordering) to 61.67 %. Compared to a no-fusion baseline, CAIF achieves a 19.6 % performance improvement by fusing only consecutive instructions non-speculatively, outperforming the baseline compiler's 1.2 % improvement and surpassing advanced hardware supporting non-consecutive fusion (Helios), which achieves a 6.6 % gain. Furthermore, combining CAIF with Helios further boosts performance, achieving on average 20.3 % reduction in execution time over the no-fusion baseline. On SPEC, CAIF increases the fraction of consecutively fused memory operations from 20 % (without compile-time reordering) to 30 %. By fu-

sing only consecutive instructions non-speculatively, CAIF delivers a 6.6 % performance improvement, surpassing the baseline compiler's 2.6 % gain.

II. Background

This section describes the hardware support for instruction fusion and instruction scheduling in a mainstream compiler. We will leverage the compiler support to enforce an instruction schedule that favors simple hardware instruction fusion.

A. Fusion Mechanisms for Memory Instructions

Multiple hardware fusion approaches exist, each balancing complexity and performance [37, 41]. Typically, fusing memory operations reduces latency only when both accesses map to the same cache line.

A.1 Microarchitectural-Level Fusion

At the decode stage, hardware can pair memory instructions that have the same base register and suitable offsets and emit a fused μ-op to optimize throughput [37].

A.2 Speculative Fusion

Predictive microarchitectures like Helios [40] fuse non-consecutive instructions based on runtime patterns, boosting fusion rates (e.g., from 1.94 % to 27.12 % in NNs). However, they incur significant hardware costs, including predictors, history buffers (9KB), and rename/issue validation to recover from misspeculation, avoid deadlocks, and ensure memory consistency.

B. Compile-Time Instruction Scheduling

Instruction scheduling is a back-end compiler optimization that reorders instructions to avoid pipeline stalls and maximize parallelism [17]. LLVM's modular design allows easy customization of scheduling strategies tailored for fusion.

III. Motivation

In this section, we characterize a hardware fusion approach for non-consecutive instructions, taking Helios as a state-of-the-art implementation and employing a Gracemont-like processor [11].

A. Characterization of Fused Pairs

Current solutions such as helios [40] employ hardware prediction mechanisms to fuse both non same-base-register (SBR) and nonconsecutive memory operations. Figure 1 illustrates the characterization of fused instructions in Helios, applied on the most common neural network operations (see section V). For each application, the percentage of memory instructions fused at run-time is displayed on the y-axis. We categorize fusion pairs into one of the following four groups, listed by increasing hardware complexity, as detailed in section II:

- *Consecutive-SBR-contiguous* simplest hardware fusion mechanism (Non speculative). Both instructions are consecutive, they use the SBR and access contiguous memory location.
- *Consecutive-SBR-64bytes* The previous but non-contiguous access within a cacheline (64B) is allowed and non-speculative.
- *Consecutive-DBR* The previous but with different base register (DBR). It is a speculative approach.
- *Non-consecutive* SBR/DBR instruction pairs that are non-consecutive in the instruction stream.

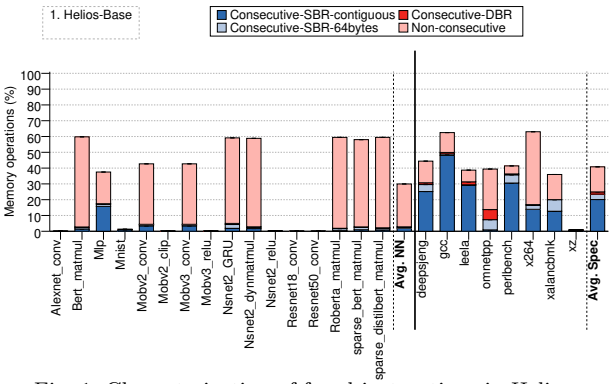


Fig. 1: Characterization of fused instructions in Helios.

In NNs, Non-consecutive fusion has the potential that the benefits outweigh the (high) hardware costs. As shown in Figure 1, on average, 27.12 % of the fused instructions are non-consecutive, while only 1.94 % fall in the Consecutive-SBR-contiguous category which requires minimal hardware support. For SPEC, on average, 15.86 % of instructions are fused non-consecutively, while 20 % are fused as consecutive SBR-contiguous instructions.

Opportunity. A fusion-aware compiler can reschedule memory operations to convert non-consecutive fusible instructions into consecutive ones, reducing reliance on Helios' complex hardware while maintaining fusion and performance.

B. The Limitation of Prediction Mechanisms

By arranging fusible instructions consecutively in the generated code, the compiler eliminates the need for hardware support for non-consecutive fusion. In addition to reducing hardware complexity, this approach avoids performance penalties caused by mispredictions and the delays introduced to enforce correctness (such as to commit).

IV. Compiler

We propose a compiler-based approach that (i) identifies fusible pairs, (ii) detects dependences, and (iii) reorders instructions to place the fusible instructions consecutively, thereby eliminating the need for a complex hardware. In short, we shift the complexity from hardware to software.

Figure 3 showing the original sequence of instructions, and highlighting how the non-consecutive load instructions (left) are identified as fusible and placed consecutively by our compiler (right).

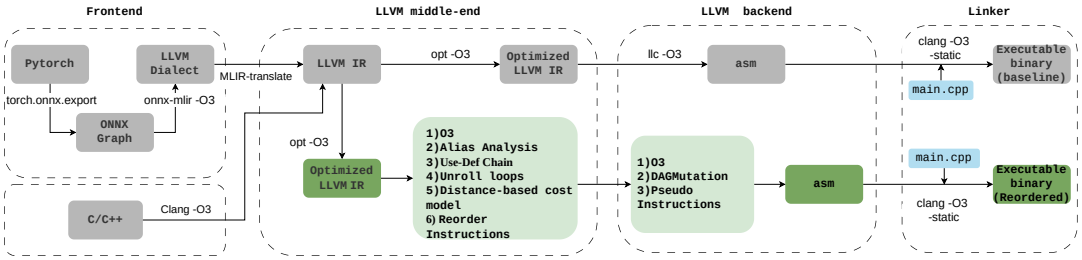


Fig. 2: Compilation flow: CAIF passes (shown in light green) are executed after the standard O3 optimization passes at each compilation stage.

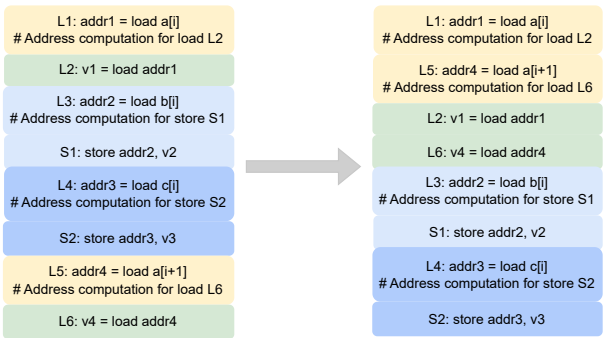


Fig. 3: Reordering instructions at LLVM IR requires complex *alias analyses* between memory instructions (*head*, *tail*, and *catalyst*) to determine the validity of the transformation and identifying instruction dependences (the *use-def chain*) to hoist the address computation together with the tail load.

Following the compilation chain described in Figure 2 and detailed in section V, we expose the compiler's intermediate representation (IR). Our compile transformations are implemented in LLVM [17] and span the compiler middle-end and back-end, as shown in Figure 2. To preserve fusion pair across backend optimizations, we extend the back-end to ensure that the fusible memory instructions remain paired in the final generated code (subsection IV-B).

A. Middle-End Analyses and Transformations

A.1 Identifying Fusible Memory Operations

In LLVM IR, we iterate each basic block's instructions — when a head load or store is found, we scan ahead for a corresponding tail in the same block. Next, we check if the head and tail operations are fusible, i.e. they access memory locations within a 64-byte range. For this we use Scalar evolution (SCEV), SCEV helps us identify how a pointer evolves in a loop, by using this information we find if two pointers within a loop, access within a 64 byte range. If the selected memory operations cannot be fused, we reset the tail and continue searching for tail till end of basic block.

We repeat this process with each load (store) that has not been marked as fusible, in the role of the head.

If the head load (store) and tail load (store) are fusible, then we keep the instructions between the two fusible memory instructions in a queue named *Catalyst*. Next, we build Use-Def chains to find instructions from the Catalyst that the tail Instruction depend on(subsubsection IV-A.2).

Finally, using alias analysis, we identify memory dependence between tail and catalyst, use-def chain of tail and catalyst.

These analyses ensure safe and correct instruction reordering. The analysis steps are summarized below:

1. Identify fusible loads/stores pair.
2. Collect intermediate instructions in the Catalyst.
3. Build the use-def chain for the tail instruction.
4. If the use-def chain of the Tail includes the Head, no reordering is performed.
5. Check memory dependences between the tail instruction and its use-def chain with the Catalyst, as follows:

a) If tail is a load:
- Check dependences with all the stores from the Catalyst.
- If its use-def chain contains any loads, check dependences between each load and all the stores from the Catalyst.

b) If tail is a store (Figure 4):
- Check dependences with all the loads and stores from the Catalyst.
- If its use-def chain contains any loads, check dependences between each load and all the stores from the Catalyst and the head store.

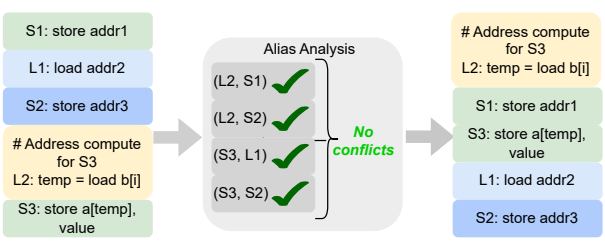


Fig. 4: Memory dependency to reorder a Tail store. The fusible instructions are S1 and S3. The use-def chain (target address computation) of L3 contains L2. Our compiler verifies for potential conflicts between the loads (L2) from the use-def chain and other stores (S2) from the catalyst that proceed the use-def chain instructions and any conflicts with the Head store. Also, it verifies the Tail store (S3) does not conflict with any load (L1) or store (S2) from the catalyst it is reordered with or with the Head store. The checks are shown in the gray box. If no conflicts exist, reordering can proceed (right).

To broaden our analysis, we integrated advanced alias analyses, combining Andersen analysis [20], flow-sensitive methods [5] from SVF [44] with LLVM's built-in analysis. For interprocedural analysis, we used GraphAA [23], enabling memory opera-

tion reordering across function calls.

A.2 Identifying Required Instructions for Hoisting

Along with the tail operation, the compiler must hoist the necessary instructions that the tail depends on, before the Head operation to preserve correctness. To identify these instructions from the Catalyst, we use the use-def chain from LLVM.

A.3 Increasing the Compiler's Reach

Dynamic loop unrolling (DLU) unrolls loops with statically unknown trip counts by guarding the unrolled iterations with conditionals. DLU may increase live variables and cause register spilling. Thus we apply it only to innermost loops not already unrolled by O3, with less than 50 instructions and at least one memory operation. The unroll factor is fixed at 2.

B. Backend Analyses and Transformations

To preserve a fusion-friendly memory access order in the backend, we employ Pseudo-Instructions.

B.1 Pseudo Instructions

Pseudo-instructions are a powerful mechanism to enforce clustering of memory operations. Through pseudo-instructions, two or more scalar memory operations are packed into one, preventing the default scheduler and register allocator from breaking the desired order of the selected memory operations.

After instruction scheduling and register allocation, pseudo-instructions are expanded back into their original scalar instructions, ensuring that the intended sequence of consecutive memory accesses is preserved. Note that additional memory operations (such as stack pushes and pops) are generated in later back-end phases and cannot be packed, because pseudo-instructions operate before register allocation. These late memory operations are instead co-scheduled using the DAGMutation [1] mechanism in CAIF.

V. Methodology

A. Workloads and Compilation Flow

To evaluate our proposal, we selected twelve neural network models: Alexnet [32], Resnet18 [35], Resnet50 [36], MobileNet v2 (Mobv2) [33], MobileNet v3 (Mobv3) [34], Mlp [30], Mnist [31], Nsnet2 [7, 25], Google Bert [6], Roberta [38], Sparse Bert [10, 13] and Sparse DistilBert [14, 48] pre-trained models.

The steps to generate LLVM IR from a neural network implemented with PyTorch are shown in Figure 2. In a nutshell, we convert PyTorch code to an ONNX graph [28], and translate the ONNX graph to the LLVM dialect of MLIR [19]. Finally, we use clang to produce an executable optimized for the RISCV with RISCV64G extensions. All compilation steps use the O3 optimization level.

We also evaluate SPEC CPU 2017 [43] (int, speed) by selecting a region of interest (ROI) based on a thorough application analysis to delimit the amount of work rather than a fixed number of instructions, thus providing comparable results across compile-time transformed code.

We conduct all experiments with two compilation flows: Base –the baseline compilation with standard O3 optimization level and no reordering– and CAIF.

B. Simulation Model and Configurations

The produced binaries are executed on top of the RISC-V Spike Simulator [9] in full-system mode running a Linux kernel [45] and trace is generated. Our cycle-level in-house simulator uses the GEMS memory model with a three-level cache hierarchy and a detailed out-of-order core that resembles an Intel Gracemont architecture[11]. We use CACTI [21] to model the energy consumption of the LQ, SQ and L1D. The LQ and the SQ are modeled as a CAM and use the high-performance (hp) model.

Our out-of-order core model has been extended to implement both consecutive and non-consecutive instruction fusion as described in Helios. In particular, we are able to run several configurations of the Helios hardware with different complexity requirements (see subsection III-A for the description of the fusion types):

- *NoFusion*: Fusion is disabled in the hardware.
- *ConsSBR*: This hardware is able to fuse only Consecutive-SBR-contiguous instruction pairs. It is a non-speculative fusion implementation.
- *ConsSBR64*: This hardware is able to fuse both Consecutive-SBR-contiguous and Consecutive-SBR-64bytes instruction pairs. It is also a non-speculative implementation.
- *Cons*: This hardware is able to fuse all consecutive instruction pairs. It requires speculative support.
- *Helios*: This hardware enables all kinds of fusion, including non-consecutive pairs. It is also a speculative solution.

VI. Evaluation

This section evaluates the impact of compiler assistance on instruction fusion, and investigate (1) the degree to which CAIF shifts fusion from non-consecutive (highest hardware complexity) to consecutive (minimum hardware complexity); (2) the overall performance with varying hardware support for fusion; (3) energy savings.

A. Characterization of Fused Pairs

To evaluate the efficiency of the compile-time reordering, we analyzed the percentage of fused memory operations across different hardware configurations of increasing complexity, as detailed in subsection V-B and with two compiler optimizations (Base and CAIF), leading to the eight combinations shown in Figure 5. Thus, Figure 5 extends Figure 1 (which corresponds to the bar labeled Helios-base) with all the configurations tested. It shows the breakdown of fusion per category: Consecutive-SBR-contiguous, Consecutive-SBR-64bytes, Consecutive-

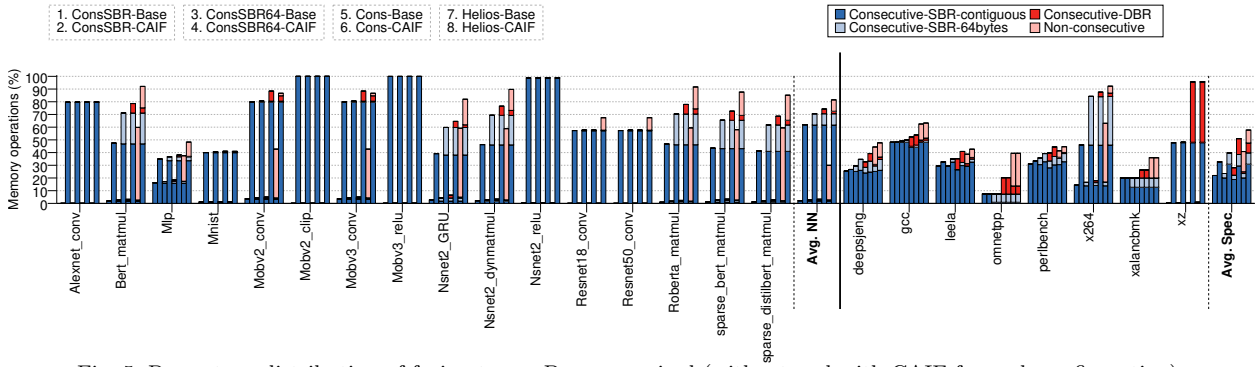


Fig. 5: Percentage distribution of fusion types. Bars are paired (without and with CAIF for each configuration).

DBR, and Non-consecutive. The Oy axis shows the percentage of fused instructions.

Focusing on the last two bars (average bar), we can observe that for Helios, CAIF converted Non-consecutive fusion to consecutive fusion hence on average CAIF managed to decrease Non-consecutive fusion from 27.12 % to 8.96 % on NNs. On SPEC, with CAIF Non-consecutive fusion reduced from 15 % to 10 %. In addition, on average for NNs, CAIF increases Consecutive-SBR-contiguous from 1.94 % to 61.67 %. Fusing consecutive memory operations requires relatively simple hardware compared to Helios. CAIF delivers high performance particularly on layers such as Matmul, GRU, and Relu.

More importantly, when focusing (Avg. NN) on a ConsSBR64 hardware, we can observe that the number of fused instructions with CAIF ConsSBR64-CAIF (4th bar, 70.61 %) increases by more than 40 % compared to Helios running the code generated by the base compiler (7th bar, 29.93 %). For SPEC (Avg. Spec), CAIF with ConsSBR64 (4th bar), we see that fused instructions have increased from 23.51 % to 39.8 % when compared to ConSBR64-Base (3rd bar). ConsSBR64 CAIF (4th bar, 39.8 %) is on par with Helios-Base (7th bar) which achieves 40.69 %. This shows that CAIF on NNs, even with simple non-speculative fusion hardware, can outperform the more complex speculative Helios hardware while achieving comparable performance on SPEC.

B. Execution Time

Figure 6 shows the impact of CAIF versus `Base` (-O3) for the different fusion configurations on execution time (y-axis) and for the neural network kernels explored in this work (x-axis). The execution time is normalized against our compiler baseline (-O3) with hardware fusion disabled (`NoFusion-Base`). Since Consecutive-DBR instructions represent a negligible (0.12 %) fraction of the fused instructions (see Figure 1), here we omit the Cons-DBR hardware configuration for clarity. We discuss the execution time by pair, with and without CAIF.

Applying our instruction reordering techniques without fusion (NoFusion-CAIF) improves execution time (by 5 % on average in NNs), thanks to dynamic loop unrolling and instruction reordering, which can help increase the scope of optimizations and enhance memory level parallelism [47]. Clustering memory instructions enhances both memory-level and instruction-level parallelism by enabling simultaneous memory requests and parallel execution of independent loads. This reduces latency, improves bandwidth utilization, and keeps the pipeline full. NoFusion-CAIF outperforms NoFusion-Base by leveraging higher ILP and MLP, demonstrating that CAIF enables a more efficient instruction schedule. Through careful engineering to reduce overheads (see section IV, our compiler introduces no register pressure: CAIF constrains DLU as described in subsubsection IV-A.3; only reorders within a basic block, which imposes a natural boundary and limits the distance of reordering; and performs backend optimizations to eliminate redundant operations, prior to packing mem-ops in pseudo-instructions.

When employing the simplest fusion hardware (ConsSBR) on NNs, CAIF reduces the average execution time by 19.1 % with respect to NoFusion-Base, while the baseline compiler achieves a 1.1 % reduction. CAIF also excels when both non-speculative fusion techniques are leveraged (ConsSBR64). In this case, CAIF reduces execution time by approximately 19.6 % with respect to NoFusion-Base, while execution time reduces by an average of 1.2 % with the baseline compiler with respect to NoFusion-Base. ConsSBR64-CAIF further reduces execution time on SPEC, improving from 2.6 % with ConsSBR64-Base to 6.6 % relative to NoFusion-Base.

The overall reduction in execution time suggests that instruction reordering indeed enhances the potential of consecutive instruction fusion even on simple hardware.

Helios on NNs, fusing all possible pairs including non-consecutive, shows an execution time reduction of 6.6 % when using the baseline compiler. However, combining Helios with instruction reordering (Helios-CAIF) yields the most significant improvement, reflecting a 20.3 % reduction in execution time compared to NoFusion-Base.

While matrix multiplication (MatMul) shows consistent performance gains, convolution workloads benefit less. This is primarily due to a higher number of cache misses in convolution, which causes processor stalls waiting for memory.

When executing on non-speculative fusion hardware, CAIF-generated binaries surpass the performance of baseline binaries leveraging more intricate hardware with speculative fusion techniques. These results underline the potential of simple fusion

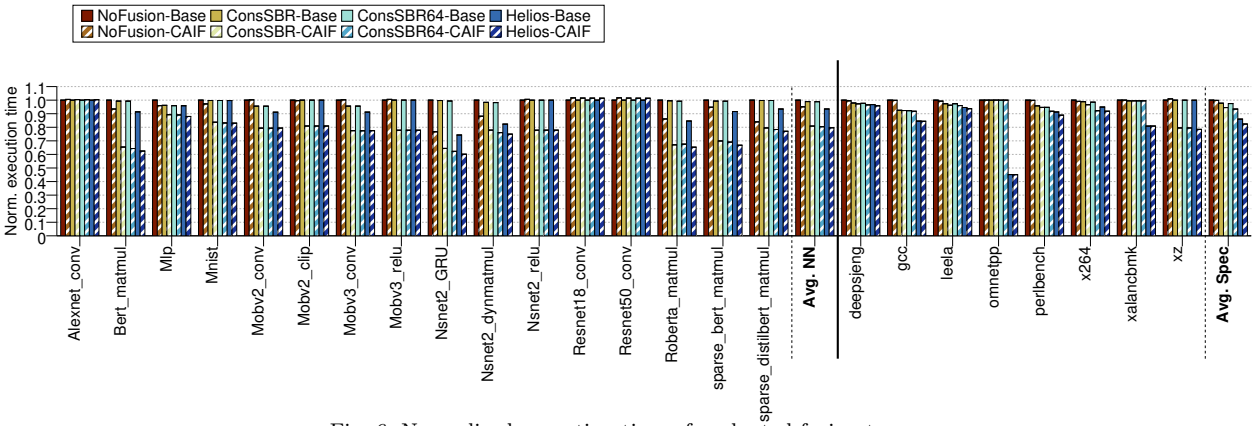


Fig. 6: Normalized execution time of evaluated fusion types.

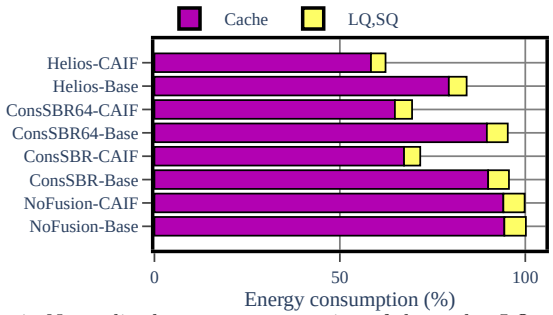


Fig. 7: Normalized energy consumption of the cache, LQ, and SQ.

mechanisms in hardware, particularly when complemented by tailored compiler instruction reordering.

C. Energy Savings

CAIF reduces energy consumption in both the L1D cache, by minimizing accesses, and the processing core, by combining two instructions into one. Figure 7 highlights its impact on the energy usage of the L1D cache, as well as the LQ and SQ—key structures affected by fusion. CAIF proves highly effective in lowering L1 cache accesses, with ConsSBR64-CAIF achieving a 31.28 % reduction and Helios-CAIF reaching 38.17 %, compared to NoFusion-Base.

Comparing Helios-Base and ConsSBR64-CAIF, which deliver comparable performance, ConsSBR64-CAIF stands out by using 18.2 % less L1D energy and 3.54 % less LQ/SQ energy which makes CAIF a more energy-efficient.

VII. Related Work

A. Software Instruction Fusion

Modern processors like ARMv7/v8 offer instructions such as Load-Pair (LDP) and Load Multiple Increment After (LDMIA), which can load multiple registers from consecutive memory locations. Those can be cracked to simplify hardware but increase latency, or executed as a unit to minimize latency at hardware cost [8] (e.g., two register file write ports to perform LDP as a unit). Fusion idioms are limited to what the architectural instructions provide.

Six et al. [42] provide a certified instruction scheduler for fusing memory operations on AArch64 platforms (*load pair* and *store pair*). However, the technique is limited to contiguous memory accesses only, as required by the AArch64 instruction semantics. Moreover, as it reorders instructions post register allocation, it is more restricted than CAIF by register-introduced dependences.

This work focuses on memory instruction fusion as they have been shown to provide the highest potential [8, 40], leveraging advanced memory alias analysis and balancing the soft constraints of DAGMutations with the hard constraints imposed through pseudo-instructions, to enable O3 back-end optimizations, but also guarantee the desired order in the final generated code.

B. Hardware Fusion

Kim and Lipasti propose Macro-op scheduling [16] pairs two dependent instructions in hardware and schedules them as a unit. This is a limited form of hardware fusion that saves scheduler entries but still performs the two instructions distinctly, i.e., the latency is the sum of both and two functional units are required over two cycles. It is orthogonal to our focus on fusing independent instructions. Singh et al. propose non-consecutive load and store fusion in hardware with the Helios microarchitecture, which relies on speculation and several points of validation to confirm that two instructions could indeed be fused. CAIF targets some of the potential of Helios without incurring hardware cost over consecutive fusion.

C. Vectorization

CAIF differs from the LLVM Loop Vectorizer [27, 26, 15], which relies on predictable loop patterns and widens instructions [29], and from SLP vectorization, which bundles isomorphic operations, instead of reordering. This lightweight mechanism for preserving instruction order from middle-end till binary generation, currently absent in LLVM and in the literature, is key to software-hardware co-design contracts.

VIII. Conclusion

Compiler reordering improves fusion by scheduling fusible instructions consecutively. Using static analysis, the compiler identifies such instructions, checks dependencies, and validates transformations—eliminating the need for complex and costly runtime hardware mechanisms.

For neural network workloads, CAIF reduces execution time by 19.6 % on hardware limited to consecutive fusion, outperforming Helios's 6.6 % (which supports both consecutive and non-consecutive fusion). When combined with Helios, CAIF further improves performance to 20.3 %. Since most instruction pairs can be fused with minimal hardware changes using CAIF, the added complexity of predictors offers limited benefit.

For SPEC workloads, CAIF achieves a 6.6 % reduction in execution time versus 2.6 % with a standard compiler on consecutive-fusion hardware, demonstrating improved performance while keeping hardware simple.

Acknowledgement

This work has been funded by the EU's Horizon 2021 research and innovation program (CONVOLVE, g.a. no. 101070374 under HORIZON-CL4-2021-DIGITAL-EMERGING-01), the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (ECHO, g.a. no 819134), the MCIN/AEI/10.13039/501100011033/ and the "ERDF A way of making Europe", EU (DAMAS, grant PID2022-136315OB-I00), and the Ramón y Cajal Research Contract RYC2018-025200-I.

Referencias

[1] Javed Absar y Florian Hahn. *Writing Great Machine Schedulers*. Accessed: 2025-12-03. URL: https://www.youtube.com/watch?v=brpomKUynEA.

[2] Advanced RISC Machines. *Arm® Neoverse™-N2 Core Software Optimization Guide*. [Online; accessed June.-2024], pág. 88.

[3] *Arm® Cortex™-A77 Core Software Optimization Guide*. [Online; accessed Nov.-2024]. Advanced RISC Machines. Cap. Section 4.13, pág. 68. URL: https://developer.arm.com/documentation/swog011050/latest/.

[4] Karthikeyan Kalyanasundaram Balasubramanian et al. "Designing RISC-V Instruction Set Extensions for Artificial Neural Networks: An LLVM Compiler-Driven Perspective". En: *IEEE Access* 12 (2024), págs. 55925-55944.

[5] Mohamad Barbar, Yulei Sui y Shiping Chen. "Object versioning for flow-sensitive pointer analysis". En: *Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization*. CGO '21. Virtual Event, Republic of Korea: IEEE Press, 2021, págs. 222-235. ISBN: 9781728186139. DOI: 10.1109/CGO51591.2021.9370334. URL: https://doi.org/10.1109/CGO51591.2021.9370334.

[6] *Google Bert*. 2020. URL: https://huggingface.co/textattack/bert-base-uncased-yelp-polarity.

[7] Sebastian Braun e Ivan Tashev. "Data Augmentation and Loss Normalization for Deep Noise Suppression". En: *Speech and Computer*. Ed. por Alexey Karpov y Rodmonga Potapova. Cham: Springer International Publishing, 2020, págs. 79-86. ISBN: 978-3-030-60276-5.

[8] Christopher Celio et al. "The Renewed Case for the Reduced Instruction Set Computer: Avoiding ISA Bloat with Macro-Op Fusion for RISC-V". En: (jul. de 2016).

[9] RISC-V Software Development. *RISC-V ISA Simulator*. https://github.com/riscv-software-src/riscv-isa-sim. Accessed: 2024-06-25. 2024.

[10] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". En: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[11] *Gracemont Microarchitecture (online)*. 2021. URL: https://www.anandtech.com/show/16881/a-deep-dive-into-intels-alder-lake-microarchitectures/4.

[12] Intel. *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. [Online; accessed June 2024]. 2024, págs. 3-12. URL: https://www.intel.com/content/www/us/en/content-details/671488/intel-64-and-ia-32-architectures-optimization-reference-manual-volume-1.html.

[13] Intel. *Sparse bert*. https://huggingface.co/Intel/bert-base-uncased-sparse-70-unstructured. Hugging Face. 2025. URL: https://huggingface.co/Intel/bert-base-uncased-sparse-70-unstructured.

[14] Intel. *Sparse Distilbert*. https://huggingface.co/Intel/distilbert-base-uncased-sparse-90-unstructured-pruneofa. Hugging Face. 2025. URL: https://huggingface.co/Intel/distilbert-base-uncased-sparse-90-unstructured-pruneofa.

[15] Ralf Karrenberg y Sebastian Hack. "Whole-function vectorization". En: *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*. CGO '11. USA: IEEE Computer Society, 2011, págs. 141-150. ISBN: 9781612843568.

[16] Ilhyun Kim y M Lipasti. "Macro-op scheduling: Relaxing scheduling loop constraints". En: *36th Int'l Symp. on Microarchitecture (MICRO)*. IEEE. 2003, págs. 277-288.

[17] C. Lattner y V. Adve. "LLVM: a compilation framework for lifelong program analysis transformation". En: *Int. Symp. Code Gener. Optim. (CGO '04)*. 2004.

[18] Chris Lattner y Vikram S. Adve. "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation". En: *2nd Int'l Symp. on Code Generation and Optimization (CGO)*. Mar. de 2004, págs. 75-88.

[19]Chris Lattner et al. “MLIR: scaling compiler infrastructure for domain specific computation”. En: *Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization*. CGO ’21. IEEE Press, 2021, págs. 2-14. ISBN: 9781728186139.

[20]Yuxiang Lei y Yulei Sui. “Fast and Precise Handling of Positive Weight Cycles for Field-Sensitive Pointer Analysis”. En: *Static Analysis: 26th International Symposium, SAS 2019, Porto, Portugal, October 8–11, 2019, Proceedings*. Porto, Portugal: Springer-Verlag, 2019, págs. 27-47. ISBN: 978-3-030-32303-5. DOI: 10.1007/978-3-030-32304-2_3. URL: https://doi.org/10.1007/978-3-030-32304-2_3.

[21]Sheng Li et al. “CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques”. En: *2011 Int’l Conf. on Computer-Aided Design (ICCAD)*. Nov. de 2011, págs. 694-701.

[22]Edgar Liberis y Nicholas D. Lane. “Neural networks on microcontrollers: saving memory at inference via operator reordering”. En: *arXiv e-prints*, arXiv:1910.05110 (oct. de 2019), arXiv:1910.05110. DOI: 10.48550/arXiv.1910.05110. arXiv: 1910.05110 [cs.LG].

[23]Dinghao Liu et al. “Improving {Indirect-Call} Analysis in {LLVM} with Type and {Data-Flow}{Co-Analysis}”. En: *33rd USENIX Security Symposium (USENIX Security 24)*. 2024, págs. 5895-5912.

[24]Yaojie Lu y Sotirios G Ziavras. “Instruction Fusion for Multiscalar and Many-Core Processors”. En: *International Journal of Parallel Programming* 45 (2017), págs. 67-78.

[25]Riccardo Miccini et al. “Dynamic nsNET2: Efficient Deep Noise Suppression with Early Exiting”. en. En: *2023 IEEE 33rd International Workshop on Machine Learning for Signal Processing (MLSP)*. Rome, Italy: IEEE, sep. de 2023, págs. 1-6. ISBN: 9798350324112. (Visitado 24-01-2024).

[26]Dorit Nuzman y Richard Henderson. “Multi-platform Auto-vectorization”. En: *Proceedings of the International Symposium on Code Generation and Optimization*. CGO ’06. USA: IEEE Computer Society, 2006, págs. 281-294. ISBN: 0769524990. DOI: 10.1109/CGO.2006.25. URL: https://doi.org/10.1109/CGO.2006.25.

[27]Dorit Nuzman, Ira Rosen y Ayal Zaks. “Auto-vectorization of interleaved data for SIMD”. En: *Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI ’06. Ottawa, Ontario, Canada: Association for Computing Machinery, 2006, págs. 132-143. ISBN: 1595933204. DOI: 10.1145/1133981.1133997. URL: https://doi.org/10.1145/1133981.1133997.

[28]*ONNX GitHub Repository (online)*. 2016. URL: https://github.com/onnx/onnx.

[29]LLVM Project. *LoopVectorize.cpp*. https://github.com/llvm/llvm-project/blob/ae11c5c2c4d7ae4cba4a8e05f0c7d85b316a2cf0/llvm/lib/Transforms/Vectorize/LoopVectorize.cpp. Commit ae11c5c. 2024.

[30]*PyTorch MLP (online)*. 2022. URL: https://github.com/christianversloot/machine-learning-articles/blob/main/creating-a-multilayer-perceptron-with-pytorch-and-lightning.md.

[31]*PyTorch MNIST (online)*. 2020. URL: https://github.com/onnx/onnx-mlir/blob/main/docs/mnist_example/mnist.onnx.

[32]*AlexNet Pretrained Model (online)*. 2016. URL: https://pytorch.org/hub/pytorch_vision_alexnet.

[33]*PyTorch Vision MobileNet V2 (online)*. 2016. URL: https://pytorch.org/hub/pytorch_vision_mobilenet_v2.

[34]*PyTorch Vision MobileNet V2 (online)*. 2016. URL: https://pytorch.org/hub/pytorch_vision_mobilenet_v2.

[35]*PyTorch RESNET18 (online)*. 2016. URL: https://pytorch.org/hub/pytorch_vision_resnet/.

[36]*PyTorch RESNET50 (online)*. 2016. URL: https://pytorch.org/hub/nvidia_deeplearningexamples_resnet50/.

[37]Chiara Riatti. “Design and Implementation of Instruction Fusion Techniques in a RISC-V Out-Of-Order Core”. Open access, UPC’s institutional repository. Treball Final de Grau (Bachelor’s Thesis). Barcelona Supercomputing Center, sep. de 2024. URL: https://hdl.handle.net/2117/425653.

[38]*Roberta*. 2016. URL: https://huggingface.co/docs/transformers/en/model_doc/roberta.

[39]Jian-Yu Shen y Shih-Wei Liao. “Evaluating and Enhancing Performance through Macro-Op Fusion Optimization with RISC-V”. En: *Workshop Proceedings of the 53rd International Conference on Parallel Processing*. ICPP Workshops’24. New York, NY, USA: Association for Computing Machinery, 2024, págs. 33-37.

[40]Sawan Singh et al. “Exploring Instruction Fusion Opportunities in General Purpose Processors”. En: *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2022, págs. 199-212.

[41]Sawan Singh et al. “Exploring Instruction Fusion Opportunities in General Purpose Processors”. En: *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2022, págs. 199-212.

[42]Cyril Six et al. “Formally verified superblock scheduling”. En: *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 2022, págs. 40-54.

[43]Standard Performance Evaluation Corporation. *SPEC CPU2017*. 2017. URL: http://www.spec.org/cpu2017.

[44]SVF-tools. *SVF: Static Value-Flow Analysis Framework for Source Code*. `https://github.com/SVF-tools/SVF`. GitHub repository. Latest release: (2025-09-08). 2025. URL: `https://github.com/SVF-tools/SVF`.

[45]Sycuricon. *riscv-spike-sdk*. `https://github.com/sycuricon/riscv-spike-sdk`. Accessed: 2024-06-25. 2024.

[46]Harsh Thakker et al. *Combining load or store instructions*. US Patent 11,593,117. Feb. de 2023.

[47]Kim-Anh Tran et al. "Static Instruction Scheduling for High Performance on Limited Hardware". En: *IEEE Transactions on Computers* 67.4 (2018), págs. 513-527.

[48]Ofir Zafrir et al. "Prune Once for All: Sparse Pre-Trained Language Models". En: *arXiv preprint arXiv:2111.05754* (2021).

Explotación de la Concurrencia para Mejorar el Rendimiento de ORB-SLAM3

Marcos Ilarraza Sarto, Jorge Ferrer Beired, Rubén Gran Tejero, Alejandro Valero y Darío Suárez Gracia[1]

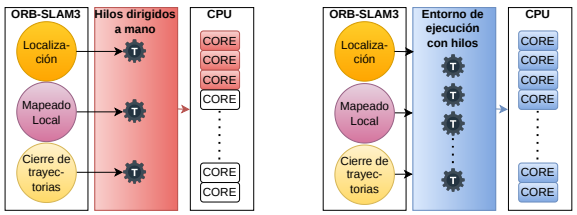
Resumen— **Localización y Mapeo Simultáneo (SLAM) es un algoritmo clave en robótica porque permite a vehículos, sondas, etcétera, crear un mapa de su entorno y localizarse en él. Por ello, muchos trabajos han intentado resolver este problema priorizando alcanzar un rendimiento mínimo (30 FPS) con una eficiencia energética aceptable en entornos con fuentes de energía limitadas. Entre estos algoritmos, ORB-SLAM3 destaca por ser el sistema de SLAM visual con una robustez equiparable a los mejores sistemas disponibles en la literatura actual y capaz de obtener una mejor precisión tanto en sus mapas como en su localización en los mismos. Sin embargo, ORB-SLAM3 está limitado a trabajar con 3 núcleos de CPU, limitando su escalabilidad en CPUs modernas, las cuales suelen disponer de al menos 6 núcleos. Este trabajo presenta una mejora de ORB-SLAM3 capaz de aprovechar todos los núcleos de CPU gracias a la exposición del paralelismo latente en el diseño original. Nuestra propuesta es capaz de ofrecer un rendimiento $1,6\times$ mayor que el de ORB-SLAM3 original consumiendo un $54\,\%$ menos de energía.**

Palabras clave— **ORB-SLAM3, pipeline, TBB.**

I. Introducción

La computación paralela se ha convertido en un punto clave para los sistemas de computación modernos. Los multiprocesadores conforman el segmento de alto rendimiento de casi todos los sectores del mercado, desde supercomputadores hasta dispositivos comerciales y sistemas embebidos. A diferencia de las arquitecturas secuenciales, los sistemas con multiprocesadores pueden ofrecer un mayor rendimiento a frecuencias de procesamiento idénticas y/o mantener el mismo rendimiento trabajando a una menor frecuencia para ahorrar energía. Esta dualidad establece un compromiso: un mayor rendimiento eleva el consumo energético, mientras que el ajuste del mismo puede mejorar la eficiencia energética. Consecuentemente, el objetivo para los sistemas modernos ha pasado de rendimiento puro a maximizar el rendimiento bajo restricciones de consumo energético. Debido a esto, la computación paralela es soportada nativamente en casi cualquier plataforma actual.

La robótica es un campo que se puede beneficiar del multiprocesado. Los robots necesitan realizar múltiples actividades en paralelo para ofrecer un servicio en un tiempo limitado, como la percepción, localización, planificación o control [1], [2]. Algunas de estas actividades son caras computacionalmente y se pueden beneficiar del paralelismo para incrementar su rendimiento [3]. Al mismo tiempo, los robots operan bajo una fuente limitada de energía, especialmente en escenarios móviles y autónomos, lo que define a la eficiencia energética como un factor crítico de diseño. Consecuentemente, los sistemas robóticos no deberían solo alcanzar el rendimiento mínimo de 30 FPS, sino hacerlo de una forma energéticamente eficiente.



(a) ORB-SLAM3: asignación estática de actividades a hilos

(b) Propuesta: las actividades generan tareas dinámicamente que se ejecutan en un *thread-pool*.

Fig. 1: Diagrama destacando como ORB-SLAM3 y nuestra propuesta mapean las actividades a núcleos de CPU.

La Localización y Mapeo Simultáneo (SLAM) es un algoritmo fundamental en robots móviles. SLAM construye un modelo del entorno (llamado mapa) mientras estima la posición del robot cuando se desplaza en él. SLAM visual es una sub-categoría que usa cámaras como sensor principal para extraer información del entorno. Entre los algoritmos dispersos de SLAM visual, ORB-SLAM3 [4] es el estado del arte, alcanzando un rendimiento robusto en un tiempo razonable con un bajo error en el cálculo de la trayectoria.

OBR-SLAM3 aprovecha el paralelismo funcional distribuyendo sus actividades principales –Localización, Mapeado Local y Cierre de Trayectorias– en tres hilos de ejecución independientes [4]. De entre estas actividades, la Localización representa el principal cuello de botella debido al alto coste computacional de la extracción de características ORB de las imágenes. En consecuencia, este trabajo introduce una arquitectura concurrente para la actividad de Localización basada en el paradigma *pipeline* con cambios mínimos en el código original.

Descomponiendo la secuencia de Localización en etapas paralelas, nuestra propuesta permite explotar los multiprocesadores modernos al máximo al desacoplar el número de operaciones del número de núcleos, incrementando tanto el rendimiento como la eficiencia energética al incrementar el uso del *hardware* disponible. Adicionalmente, en lugar de mapear estáticamente cada actividad a un hilo de ejecución, las implementamos como un conjunto de tareas que son gestionadas por un entorno de ejecución, que permite su mapeo dinámico al *hardware* disponible.

[1]Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza - Instituto de Investigación en Ingeniería de Aragón (I3A), e-mail: {milarraza, jferrer, rgran, alvabre, dario}@unizar.es.

La Figura 1 ilustra el mapeo de actividades al *hardware* de la aplicación ORB-SLAM3 original (Figura 1a) comparado con cómo lo hace nuestra propuesta (Figura 1b) sobre todos los núcleos disponibles de la CPU. Debido a que ORB-SLAM3 está limitado a tres hilos concurrentes creados en su arranque, no es capaz de aprovechar todos los núcleos disponibles para incrementar su rendimiento. En cambio, nuestra propuesta permite la ingesta de nuevas imágenes en núcleos adicionales mientras las imágenes previas se siguen procesando, paralelizando el procesado de la *pipeline*.

Como resultado alcanzamos un rendimiento equivalente a la referencia aprovechando una mayor cantidad de núcleos, lo que permite reducir la frecuencia de operación de cada núcleo. Conseguimos esto generando tareas en base a cada actividad y disponiendo de un entorno de ejecución que se auto-gestiona para asignarlas a hilos disponibles.

Las principales aportaciones de este trabajo son las siguientes:

- Una caracterización del rendimiento, consumo energético y concurrencia de ORB-SLAM3 original (que usaremos como base).
- La propuesta de una implementación utilizando el paradigma de *pipeline* para explotar mejor el paralelismo de sistemas SLAM.
- La aceleración de bucles internos compatibles con la paralelización a nivel de iteración.
- Una comparación cuantitativa del rendimiento y consumo energético de nuestra propuesta y la base en un computador de escritorio AMD6, demostrando un incremento en el rendimiento de hasta $2,5\times$ y hasta un $54\,\%$ de reducción del consumo energético comparado con el sistema base.

El resto del artículo se organiza de la siguiente manera. La Sección II proporciona el conocimiento mínimo necesario sobre ORB-SLAM3 y el paradigma de *pipeline*. En la Sección III introducimos trabajos relacionados con el incremento del rendimiento, así como la eficiencia energética de ORB-SLAM3 o en el módulo de extracción de características ORB. En la Sección IV se describe nuestra propuesta para conseguir que ORB-SLAM3 escale con el número de núcleos disponibles. Las características de la plataforma utilizada y la metodología de evaluación se discuten en la Sección V. La Sección VI presenta los resultados de nuestra propuesta y su comparación con el sistema base. Por último, en la Sección VII se exponen las conclusiones de este trabajo.

II. Marco Teórico

Esta sección presenta ORB-SLAM3 enfocándose en su estructura, actividades y terminología, seguido por una explicación de los beneficios de incrementar el paralelismo, así como la introducción del paradigma *pipeline* y los entornos de ejecución.

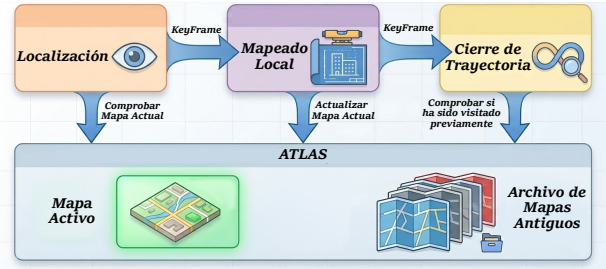


Fig. 2: Esquema simplificado de la estructura de ORB-SLAM3.

A. La Aplicación ORB-SLAM3

Los métodos de SLAM visual se dividen comúnmente en dos grupos: i) los métodos densos explotan casi toda la información contenida en las imágenes para construir un mapa de nube de puntos [5], [6], [7], [8] y ii) los métodos dispersos se centran únicamente en los puntos más informativos (puntos de referencia) en la imagen [4], [9], [10], [11]. Debido a su alto coste computacional, los métodos densos suelen emplear técnicas basadas en aprendizaje automático, como redes neuronales, para la extracción de características, mientras que los métodos dispersos emplean algoritmos basados en descriptores para extraer los puntos de interés.

ORB-SLAM3 [4], el estado del arte de SLAM visuales dispersos, es una biblioteca de código abierto que da soporte a SLAM visual, visual-inercial y multi-mapa utilizando cámaras monoculares, estéreo y RGB-D, con y sin medidas inerciales. La Figura 2 muestra que ORB-SLAM3 opera con 3 actividades principales en tres hilos independientes: Localización, Mapeado Local y Cierre de Trayectorias. Adicionalmente, ORB-SLAM3 utiliza una representación multimapa de mapas desconectados referida como Atlas. El Atlas mantiene un mapa activo y los mapas antiguos en desuso, permitiendo al sistema reconocer lugares visitados previamente que no están registrados en el mapa activo.

La actividad de Localización extrae las características ORB para definir *KeyPoints* (puntos de interés). Un conjunto de *KeyPoints* dentro de una misma imagen se llama *frame*. La Localización calcula la posición del sistema emparejando los *KeyPoints* con el mapa activo. Si el número de emparejamientos exitosos cae por debajo de un umbral específico, el *frame* se designa como *KeyFrame*. Los *KeyFrames* son *frames* que la Localización pasa a la actividad de Mapeado Local, que incluye sus *KeyPoints* como *MapPoints* en el mapa activo para servir de referencia a futuros *frames*.

Por último, los nuevos *KeyFrames* son analizados por la actividad de Cierre de Trayectorias, que comprueba si el *KeyFrame* coincide con algún escenario que se haya visitado previamente. Si encuentra una coincidencia, actualiza el mapa activo. Si la coincidencia surge con un mapa antiguo, los dos mapas son fusionados.

Si la actividad de Localización falla en localizar varios *frames* con el mapa activo, el mapa activo se archiva como un mapa viejo y se inicializa un nuevo

mapa activo.

B. El Paralelismo Pipeline

El paradigma *pipeline* [12] es un patrón de programación paralela en el que un flujo de datos pasa por una serie de etapas de filtrado y transformación. La salida de la última etapa es un flujo de datos procesados. Cada etapa es un objeto autónomo que opera de manera concurrente y asíncrona, recibiendo y enviando mensajes a otras etapas y actualizando su estado local. Es posible definir etapas que reciben y envían más de un mensaje, requiriendo la sincronización de los mensajes de entrada para iniciar la etapa.

Al implementar una solución con el paradigma *pipeline*, la carga computacional es dividida entre más núcleos. Esta reducción en la carga de trabajo por núcleo hace que sea posible reducir la frecuencia de cada núcleo manteniendo el rendimiento de la aplicación. Debido a que la disipación de potencia escala cúbicamente con la frecuencia de operación [13], la paralelización también proporciona una solución más eficiente energéticamente.

Existen múltiples entornos que dan soporte al paradigma *pipeline* con un entorno de ejecución (véase la Figura 1b), incluyendo TaskFlow [14], StarPU [15], OmpSs [16] y Threading Building Blocks (TBB) [17]. Estos entornos trabajan como entornos de ejecución que se encargan de mapear las tareas al *hardware* disponible de forma dinámica y gestionan la ejecución con cambios de contexto ligeros, que son más eficientes que los cambios de contexto realizados por el sistema operativo. Además, suelen ofrecer una interfaz para asignar prioridades a cada tarea, permitiendo dedicar más tiempo en CPU a las tareas más críticas.

TBB es un entorno comercial para la programación de código C++ escalable. Ofrece una planificación robusta, gestión de memoria y un amplio conjunto de patrones predefinidos (como el de *pipeline* o el de bucles paralelos). Sus implementaciones se basan en los cimientos de C++ moderno. Por todas estas razones hemos elegido TBB.

En TBB nombran como "*token*" a cada elemento independiente del flujo de datos y puede limitar la cantidad de tokens que procesa la *pipeline* al mismo tiempo. Esto es importante porque también permite definir etapas paralelas, una etapa que puede ser aplicada a distintos tokens en paralelo. Cuando un token se asigna a una etapa, TBB lanza una tarea para ejecutar la etapa con el mensaje recibido como entrada.

III. Trabajo Relacionado

Entre las operaciones de ORB-SLAM3, la operación que más tiempo consume es la extracción de características ORB y su emparejamiento. Como resultado, numerosos trabajos han intentado mejorar su rendimiento y eficiencia. El trabajo previo se puede clasificar en trabajos que han desarrollado *hardware* dedicado a acelerar esta operación y trabajos que se han centrado en adaptar la estructura de ORB-SLAM3 para mejorar su rendimiento.

A. Propuestas Basadas en Hardware

La extracción de características ORB es ampliamente usada en el entorno de SLAM Visual, siendo implementada en otros algoritmos aparte de ORB-SLAM3. Como resultado, muchos trabajos han desarrollado *hardware* dedicado a mejorar el rendimiento [18], [19], su eficiencia energética [20] o ambas características del módulo ORB [21], [22], [23]. Todas ellas implementan la misma *pipeline* de extracción de características ORB: extraer los puntos de interés mediante FAST y/o el detector de esquinas Harris-Stephens, así como la generación de descriptores usando BRIEF o una variación del mismo. Debido a que estas mejoras se centran en el módulo ORB, estos trabajos serían aplicables para mejorar el rendimiento de ORB-SLAM3.

Fang *et al.* [24] proponen un acelerador ORB basado en una FPGA con una implementación de SLAM visual basada en ORB-SLAM [25] y MSKCF [26] para incrementar tanto su rendimiento como su eficiencia energética manteniendo un error de trayectoria bajo.

Vibhakar y Demng [27] proponen un acelerador basado en una FPGA con estructura de *pipeline* que incrementa el rendimiento y la eficiencia energética de ORB-SLAM3. Su propuesta puede procesar la extracción de características ORB para la siguiente imagen mientras localiza su posición en función del *frame* actual.

Xue *et al.* [28] presentan un acelerador del extractor ORB que implementa paralelismo a nivel de dato y a nivel de instrucción, alcanzando una aceleración de $39,8\times$ con respecto a otras FPGAs ejecutando ORB-SLAM2 aunque su solución consume el doble de energía que la versión original.

Taranco *et al.* [29] proponen Locator, un acelerador ORB de alto rendimiento y alta eficiencia energética para vehículos autónomos que ejecutan ORB-SLAM [25]. Alcanza una aceleración de hasta $8\times$ la versión original con una reducción de $14597\times$ veces el consumo energético de una CPU de alto rendimiento. Internamente, su arquitectura implementa una *pipeline* que paraleliza la extracción de características dentro de una misma imagen.

B. Propuestas Basadas en Software

Aparte de la aceleración por *hardware*, otros trabajos se centran en mejorar la estructura de ORB-SLAM3. SOLO-SLAM mezcla el filtrado tradicional y un método basado en *deep learning* para manejar objetos dinámicos, mejorando la precisión de ORB-SLAM3 y manteniendo el mismo rendimiento [30]. Esto se consigue al procesar la segmentación de las imágenes y la estimación de la posición del sistema en paralelo.

OASIS [31] calcula el tiempo requerido para acabar de procesar cada *frame* y dinámicamente genera máscaras sobre qué secciones de las próximas imágenes se deben trabajar en la etapa de extracción de características, permitiendo mantener un rendimiento en tiempo-real más robusto en plataformas con limitaciones computacionales como sistemas embebidos.

Finalmente, a nuestro mejor saber y entender, Peng

et al. [32] son los únicos autores que evalúan el consumo energético y rendimiento de ORB-SLAM2 trabajando únicamente con una CPU o trabajando con una configuración CPU-GPU. Además, muestran que deshabilitar todos los núcleos de la CPU, salvo tres, no tiene ningún impacto en el rendimiento de la aplicación pero reduce su consumo energético. Esta observación indica que los núcleos adicionales de la CPU no se aprovechan en ORB-SLAM3.

Ninguno de estos trabajos ni el propio ORB-SLAM3 se enfocan en la escalabilidad del sistema con respecto al número de núcleos de CPU en procesadores modernos. En contraste, este trabajo expone paralelismo en ORB-SLAM3 implementando una versión con *pipeline* de la actividad de Localización y paralelizando el procesado de bucles, permitiendo a todos los núcleos disponibles contribuir a la ejecución.

IV. Propuesta

Este trabajo tiene como objetivo exponer oportunidades de concurrencia en ORB-SLAM3. Para ello, realizamos un análisis de dependencias entre tareas para identificar oportunidades de ejecución en paralelo. En base a este análisis, proponemos explotar el paralelismo entre tareas proponiendo una implementación con *pipeline* de la actividad de Localización (véase la Figura 2). Además, investigamos el potencial de paralelismo entre datos identificando regiones en el código con bucles computacionalmente intensos compatibles con el paralelismo de datos.

A. Versión Pipeline de la Actividad de Localización

SLAM visual procesa un flujo continuo de imágenes capturadas por una cámara. Una observación clave es que las etapas iniciales de la actividad de Localización están dominadas por operaciones computacionalmente intensas que pueden ser ejecutadas independientemente para cada imagen debido a que no dependen de la imagen anterior ni de su *frame*. En particular, la extracción de características y su emparejamiento no requieren de información de ningún dato procesado previamente. Esta característica permite el procesado concurrente de imágenes sucesivas por medio de una *pipeline*, consiguiendo exponer paralelismo entre imágenes sin violar dependencias de datos. Para implementar la versión con *pipeline* de la actividad de Localización, esta se divide en tres etapas:

- **Carga**: responsable de cargar imágenes desde la cámara al sistema (una o dos si es monocular o estéreo, respectivamente). Esta etapa no depende de información de imágenes/*frames* previos.
- **Procesado**: responsable del pre-procesado de las imágenes entrantes, extraer sus características ORB y realizar el emparejamiento estéreo. Esta etapa tampoco depende de información de imágenes/*frames* previos.
- **Localización**: responsable de estimar la posición del sistema basado en la posición del *frame* actual, el *frame* anterior y el mapa activo. También determina si el *frame* actual debería ser incorporado al mapa activo promoviéndolo a *KeyFrame*.

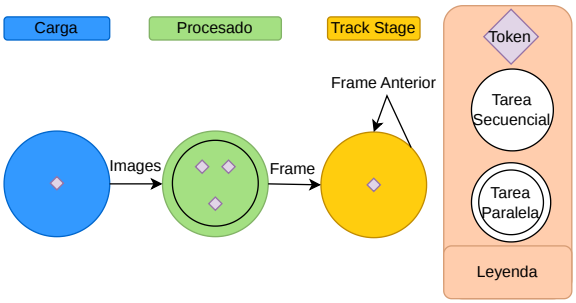


Fig. 3: Diagrama la versión escalable de 3 etapas de la actividad de Localización.

Si utilizamos una cámara que captura imágenes más rápido de lo que el sistema tarda en integrar una imagen en la trayectoria, esto implica que el sistema es incapaz de alcanzar el ritmo de la cámara. Por ejemplo, si trabajásemos con una cámara que captura una nueva imagen cada 15 ms, la implementación base no sería capaz de seguirle el ritmo a la cámara. Por contra, al permitir iniciar el procesamiento de nuevos *frames* cuando están disponibles dentro de nuestra *pipeline*, nuestra propuesta permite procesar varias imágenes en paralelo, mejorando el rendimiento. Este diseño se debe aplicar con cuidado porque puede introducir latencia adicional debido a la necesidad de gestionar las colas de tareas y sobrecarga de sincronización de las etapas de la *pipeline*. Sin embargo, nuestros resultados indican que el sobrecoste es mínimo y eclipsado por las ganancias.

La etapa de Procesado es la más demandante en términos de coste computacional y puede ser ejecutada en paralelo por varias imágenes debido a que no tiene dependencias entre imágenes. En consecuencia, el sistema es capaz de procesar más de una imagen a la vez, permitiéndole adaptarse a mayores FPS (*Frames* Por Segundo) que la versión base. En cambio, la etapa de Localización introduce una dependencia temporal entre los *frames* que procesa. Si dos instancias de la actividad de Localización llegan a esta etapa a la vez, el *frame* más reciente deberá esperar a que el *frame* más antiguo acabe su ejecución. Igualmente, el tiempo medio de ejecución de la etapa de Localización es significativamente más corto (alrededor de 4,7 ms) que el de la etapa de Procesado (alrededor de 13,8 ms), limitando en consecuencia su impacto en el rendimiento del sistema.

A raíz de la división anterior, la Figura 3 ilustra un diagrama con el diseño de nuestra propuesta para la implementación de la actividad de Localización con *pipeline*. Los nodos circulares representan tareas secuenciales, los nodos con doble círculo representan tareas que pueden ser procesadas en paralelo para varios tokens a la vez, mientras que los diamantes representan un ejemplo de la cantidad de tokens que pueden estar procesando. Los nodos secuenciales siempre procesarán como mucho un token a la vez, mientras que los nodos paralelos pueden procesar uno o más tokens a la vez. Las flechas representan la información enviada entre nodos, apuntando a la tarea

que recibe la información.

Esta propuesta explota dos tipos de paralelismo: el paralelismo de *pipeline* y el paralelismo de datos. El paralelismo de *pipeline* permite que distintas etapas procesen distintos datos de forma concurrente, manteniendo la dependencia entre etapas. El paralelismo de datos permite que la misma etapa sea ejecutada para distintos tokens concurrentemente.

Tras definir la *pipeline* nos centramos en el nuevo cuello de botella de la actividad de Localización (i.e., la etapa de Localización), así como en las actividades de Mapeado Local y Cierre de Trayectorias. En estas actividades encontramos tres bucles que son susceptibles de incrementar su rendimiento mediante su implementación como bucles paralelos debido a su gran número de iteraciones:

- **Predicción de posición**: dentro de la etapa de Localización existe un bucle que itera sobre todos los *MapPoints* para encontrar coincidencias con el mapa actual. Ejecuta un número de iteraciones del orden de 1000 y toma unos $0,64 \pm 0,25$ ms por *frame*.
- ***MapPoints* visibles**: dentro de la actividad de Mapeo Local, itera sobre los *MapPoints* que deberían ser visibles en el nuevo *KeyFrame* para comprobar si son visibles o no. Típicamente ejecuta entre 1000 y 5000 iteraciones y suele tardar unos $1,24 \pm 0{,}78$ ms.
- **Emparejado de *MapPoints***: Tras el bucle anterior, la actividad de Mapeo Local itera sobre los *MapPoints* visibles para emparejarlos con los puntos de interés del nuevo *KeyFrame*. Para este bucle, el número de iteraciones es de alrededor de 1000 y suele tardar $0{,}93 \pm 0{,}54$ ms.

B. Implementación con TBB

TBB ofrece una plantilla para implementar grafos de flujo que también se puede usar para implementar *pipelines*. Requiere de la definición explícita de un nodo limitador para limitar el número de tokens que puede procesar a la vez y de un nodo secuenciador para garantizar el procesado en orden de los tokens. Sin embargo, permite la definición de etapas ramificadas y su unión posterior, facilitando que múltiples nodos procesen el mismo token en paralelo, lo cual resulta útil para el procesado de imágenes estéreo.

Además de la implementación de la *pipeline*, los tres bucles que mencionamos arriba se re-implementaron utilizando la plantilla *parallel_for* de TBB, que divide la carga de trabajo en trozos que son procesados por tareas paralelas.

Finalmente, integramos el paralelismo de nuestra propuesta con el paralelismo existente en ORB-SLAM3 original: las actividades de Mapeado Local y de Cierre de Trayectorias son lanzadas con la plantilla *task_group* de TBB. Al lanzarlas de esta manera permitimos al entorno de ejecución administrarlas junto con la actividad de Localización, permitiendo que sea consciente de todo lo que se esta ejecutando en la aplicación. Además, permite asignarles una mayor prioridad que a la actividad de Localización.

Tabla I: Descripción de las principales características del sistema de prueba.

	hoGAZa
OS	Ubuntu 22.04.4 LTS
Kernel	Linux 5.15.0-117-generic
CPU	Intel Core i5-8400
# núcleos	6
# hilos	6
Frec. Máx. (GHz)	4
TDP (W)	65
Contadores HW energía	Sí

Esta decisión de diseño provoca que se priorice las actualizaciones del mapa activo por encima de su localización en el mismo. Esto no afecta negativamente al tiempo de localización debido a que el número de *KeyFrames* suele ser significativamente menor al número total de *frames* de una trayectoria.

V. Entorno Experimental

Hemos evaluado nuestra propuesta en un equipo de sobremesa al que denominamos *hoGAZa* (véase la Tabla I). El procesador es un Intel Core i5-4000 y es capaz de ejecutar 6 hilos concurrentemente. Esta CPU ofrece contadores internos (*hardware counters*) para medir el rendimiento y la energía de forma precisa.

Caracterizamos nuestra propuesta con dos perspectivas complementarias: rendimiento y consumo energético. Ambos aspectos son críticos para las aplicaciones de SLAM debido a que los despliegues en el mundo real requieren operar en un tiempo razonable (e.g., 30 FPS) con restricciones energéticas. Durante nuestros experimentos, medimos las siguientes métricas: i) tiempo total de ejecución de la aplicación (en s) ii) la Raíz Cuadrada del Error Medio de la Trayectoria (RMS ATE en metros) y iii) el consumo energético total del nodo -el *package* de la CPU y la RAM- al ejecutar la aplicación (en J).

Realizamos estos experimentos utilizando las 5 trayectorias Machine Hall (MH) del *dataset* EuRoC MAV [33]. Las secuencias de MH consisten en diferentes vuelos a través de un entorno industrial con tuberías y otros elementos capturadas con una trayectoria de referencia de precisión milimétrica, obtenida con un rastreador láser Leica MS50. Todas las secuencias constan de imágenes estéreo con una resolución de 752×480 píxeles.

Realizamos dos grupos de experimentos: *experimentos de rendimiento* y ii) *experimentos de consumo energético* enfocados en evaluar las ganancias de nuestra propuesta y en analizar el impacto del ajuste de la frecuencia de operación de la CPU. Ambos grupos de experimentos se evaluaron utilizando una cámara ideal, que provee una imagen cuando se le solicita (FPS ilimitados). Ejecutamos cada configuración tres veces y registramos los valores medios.

Adicionalmente, en los experimentos de rendimiento evaluamos el rendimiento en las 5 trayectorias Machine Hall con respecto al sistema base. Fijamos la frecuencia de operación de la CPU a 4 GHz para

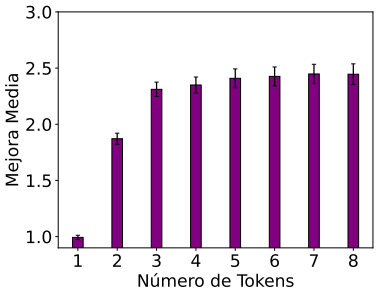


Fig. 4: Mejora de rendimiento de nuestra propuesta respecto a ORB-SLAM3 base. Cada barra representa la mejora media a través de los cinco datasets EuRoC Machine Hall con su desviación estándar. Los experimentos han sido ejecutados en hoGAZa con una frecuencia fija de 4 GHz.

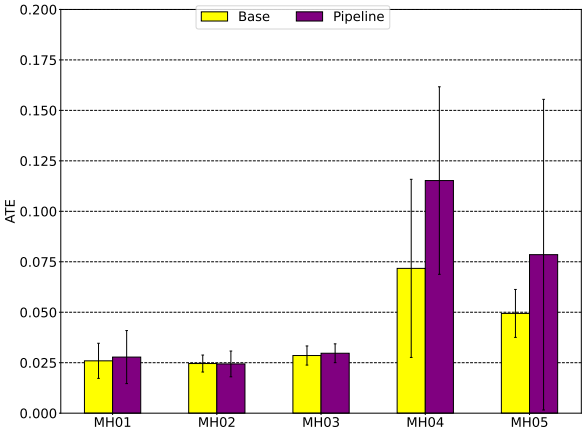


Fig. 5: Error de Trayectoria Medio (ATE) de ORB-SLAM3 y nuestra propuesta en cada una de las trayectorias. Los experimentos han sido ejecutados en hoGAZa con una frecuencia fija de 4 GHz. Cada barra ha sido generada con 24 experimentos.

garantizar que ambas versiones utilizan las máximas capacidades de la CPU. Evaluamos el rendimiento en términos de mejora de rendimiento y RMS ATE.

Con respecto a los experimentos de consumo energético, evaluamos cinco frecuencias de operación fija para analizar su efecto en el rendimiento y el consumo energético. Para estos experimentos nos centramos en la trayectoria MH01 debido a que las tendencias de rendimiento fueron consistentes a lo largo de todas las secuencias. Las métricas que evaluamos fueron el consumo energético y los FPS.

VI. Resultados Experimentales

Esta sección analiza en detalle el impacto de nuestra propuesta, agrupando los resultados en términos de rendimiento, energía y métricas de escalabilidad.

A. Rendimiento

En primer lugar, con respecto al rendimiento, o en otras palabras, con respecto al número de imágenes procesadas por segundo, nuestra propuesta alcanza una aceleración de más del doble, $2,5\times$, del rendimiento de ORB-SLAM3 original, sin comprometer la calidad de la trayectoria.

La Figura 4 ilustra que la mejora de rendimiento respecto al sistema base varía con el número de tokens; es decir, el número máximo de imágenes que están en vuelo en la *pipeline*. El rendimiento incrementa sustancialmente entre uno y tres tokens, alcanzando una aceleración de aproximadamente $2,25\times$. A partir de este punto, el incremento en la mejora se reduce debido a que la implementación empieza a utilizar todos los núcleos disponibles, estabilizándose con cinco o más tokens.

B. ATE

La Figura 5 ilustra el ATE entre el enfoque base (amarillo) y nuestra propuesta (morado) en todas las trayectorias MH. El ATE permanece consistente para ambas versiones; sin embargo, nuestra propuesta exhibe un ATE ligeramente mayor en las trayectorias MH04 y MH05. Estos casos específicos pertenecen a la categoría "difícil" de las trayectorias y muestran una mayor desviación estándar en ambas versiones. Por ello, concluimos que nuestra propuesta no afecta negativamente a la calidad de la trayectoria.

	0,8	1,6	2,4	3,2	4,0
Base	2199	1689	1716	**2045**	**2715**
1 Tkn	2287	1804	1930	**2378**	**3238**
2 Tkn	1501	**1292**	**1440**	**1804**	**2466**
3 Tkn	1316	**1163**	**1293**	**1640**	**2224**
4 Tkn	1280	**1146**	**1289**	**1621**	**2228**
5 Tkn	1262	**1131**	**1276**	**1615**	**2184**
6 Tkn	1237	**1119**	**1262**	**1613**	**2187**
7 Tkn	1246	**1117**	**1257**	**1611**	**2198**
8 Tkn	1240	**1110**	**1269**	**1609**	**2169**

Tabla II: Consumo total de energía (en J) con la trayectoria MH01 utilizando ORB-SLAM3 base y nuestra propuesta con distintas cantidades de tokens (Tkn). Las columnas se refieren a frecuencias de operación en GHz. En negrita se resaltan aquellas configuraciones que alcanzan al menos 30 FPS.

C. Análisis Energético

Como se ha mencionado anteriormente, hemos evaluado el consumo energético en el dataset MH01 fijando la frecuencia de operación de la CPU. Cuando ambas versiones se ejecutan a la misma frecuencia, nuestra propuesta consume más cuando trabaja con un token; esto es de esperar debido a la sobrecarga de una gestión de tareas más compleja con el mismo número de hilos activos. Aun así, la Tabla II muestra que, con dos o más tokens, nuestra propuesta consume consistentemente menos energía que la referencia; en esta marcamos en negrita las configuraciones que alcanzan el rendimiento mínimo de 30 FPS, observando que el enfoque base necesita trabajar al menos a 3,2 GHz, mientras que nuestra propuesta lo consigue trabajando a 1,6 GHz; es decir, podemos ofrecer el rendimiento mínimo con un menor coste energético.

Finalmente, hemos analizado que configuración frecuencia-número de tokens equilibra mejor el rendimiento y el consumo energético, asignándoles a ambos el mismo peso. La Tabla III muestra el rendimiento de cada configuración. Esta vez solo hemos marcado el máximo rendimiento que ofrece el sistema base (44 FPS) y la configuración de nuestra propuesta que mejor equilibra la mejora de rendimiento y el consumo energético: trabajar a 2,4 GHz con 6 *tokens* (72 FPS). A esta frecuencia alcanzamos una aceleración de $1,6\times$ en rendimiento respecto a base y consumiendo un 54 % menos de energía.

	0,8	1,6	2,4	3,2	4,0
Base	9,6	19,0	28,4	37,4	**44,0**
1 Tkn	9,7	19,2	28,8	37,7	44,3
2 Tkn	18,2	36,3	54,2	71,4	82,9
3 Tkn	22,2	44,7	67,3	86,6	100,7
4 Tkn	23,2	45,9	68,2	89,2	101,5
5 Tkn	23,8	46,9	70,1	90,2	105,3
6 Tkn	24,2	47,8	71,0	91,0	105,0
7 Tkn	24,1	47,9	**71,8**	91,1	104,7
8 Tkn	24,3	48,4	70,7	91,6	105,6

Tabla III: FPS alcanzados con la trayectoria MH01 utilizando ORB-SLAM3 base y nuestra propuesta con distintas cantidades de tokens. Las columnas se refieren a frecuencias de operación en GHz. En negrita se resalta el consumo del máximo rendimiento de base y nuestra configuración que mejor equilibra eficiencia y consumo energético.

VII. Conclusiones

Este trabajo ha presentado una versión paralelizada mediante paralelismo de *pipeline* de ORB-SLAM3 que escala con el número de núcleos de CPU disponibles. Nuestra propuesta mejora el rendimiento, alcanzando mejoras de rendimiento (*throughput*) de hasta $2,5\times$ en un computador de escritorio.

También hemos presentado una caracterización exhaustiva en términos de rendimiento y consumo energético basada en ajustar la frecuencia de operación de la CPU. Nuestra propuesta es capaz de alcanzar el mismo rendimiento que ORB-SLAM3 original consumiendo un 58 % menos de energía. Además, el mejor equilibrio entre rendimiento y consumo energético se obtiene trabajando con nuestra propuesta a 2,4 GHz, donde se alcanza una mejora de hasta $1,6\times$, consumiendo un 54 % menos que el sistema original ejecutándose a 4 GHz.

Nuestro trabajo futuro incluirá la extensión de la propuesta a más modelos de cámaras debido a que la implementación actual solo da soporte a cámaras estéreo. También planeamos explotar el paralelismo en otras secciones del algoritmo inexploradas en este trabajo.

Agradecimientos

Todos los autores reconocen el apoyo de las subvenciones (1) PID2022-136454NB-C22/AEI/10.13039/501100011033 de la Agencia Estatal de Investigación (AEI), (2) el grupo de investigación gaZ: T58_23R del Departamento de Ciencia, Universidad y Sociedad del Conocimiento del Gobierno de Aragón y (3) del Instituto de Investigación e Ingeniería de Aragón (I3A). Los financiadores no tuvieron ningún papel en el diseño del estudio, la recopilación y el análisis de datos, la decisión de publicar ni en la preparación del manuscrito.

Referencias

[1] Hongxing Wei, Zhenzhou Shao, Zhen Huang, Renhai Chen, Yong Guan, Jindong Tan, and Zili Shao, "RT-ROS: A real-time ROS architecture on multi-core processors," *Future Generation Computer Systems*, vol. 56, pp. 171–178, 2016.

[2] Wesley A. Powell, Michael A. Johnson, Jonathan Wilmot, Raphael Some, Kim P. Gostelow, Glenn Reeves, and Richard J. Doyle, "Enabling future robotic missions with multicore processors," Tech. Rep., NASA Technical Reports Server, 2011.

[3] Jeffrey Ichnowski and Ron Alterovitz, "Scalable multicore motion planning using lock-free concurrency," *IEEE Transactions on Robotics*, vol. 30, no. 5, pp. 1123–1136, 2014.

[4] Carlos Campos, Richard Elvira, Juan J. Gómez, José M. M. Montiel, and Juan D. Tardós, "ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.

[5] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Proceedings of the 10th IEEE International Symposium on Mixed and Augmented Reality*, 2011, pp. 127–136.

[6] Thomas Whelan, Renato F. Salas-Moreno, Ben Glocker, Andrew J. Davison, and Stefan Leutenegger, "ElasticFusion," *International Journal of Robotics Research*, vol. 35, no. 14, pp. 1697–1716, 2016.

[7] Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt, "BundleFusion: Real-Time Globally Consistent 3D Reconstruction Using On-the-Fly Surface Reintegration," *ACM Transactions on Graphics*, vol. 36, no. 3, 2017.

[8] Olaf Kähler, Victor Adrian Prisacariu, Carl Yuheng Ren, Xin Sun, Philip H. S. Torr, and David Murray, "Very high frame rate volumetric integration of depth images on mobile devices," *IEEE Transactions on Visualization and Computer Graphics*, vol. 21, no. 11, pp. 1241–1250, 2015.

[9] Georg Klein and David Murray, "Parallel tracking and mapping for small ar workspaces," in *Proceedings of the 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2007, pp. 225–234.

[10] Christian Forster, Matia Pizzoli, and Davide Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2014, pp. 15–22.

[11] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale, "Keyframe-based visual-inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2014.

[12] David Padua, "Patterns of actor programming," in *Encyclopedia of Parallel Computing*, David Padua, Ed. Springer, 2011.

[13] Trevor Mudge, "Power: A firstclass architectural design constraint," *Computer*, vol. 34, no. 4, pp. 52–58, 2001.

[14] Tsung-Wei Huang, Dian-Lun Lin, Chun-Xun Lin, and Yibo Lin, "Taskflow: A lightweight parallel and heterogeneous task graph computing system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1303–1320, 2022.

[15] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier, "Starpu: A unified platform for task scheduling on heterogeneous multicore architectures," in *Euro-Par 2009 Parallel Processing*, 2009, pp. 863–874.

[16] Alejandro Duran, Eduard Ayguade, Rosa M. Badia, Jesus Labarta, Luis Martinell, Xavier Martorell, and Judit Planas, "Ompss: A proposal for programming heterogeneous multi-core architectures," *Parallel Processing Letters*, vol. 21, no. 02, pp. 173–193, 2011.

[17] Michael J. Voss and James R. Reinders, *Today's TBB: C++ Parallel Programming with Threading Building Blocks*, Apress, Berkeley, CA, 2 edition, 2025.

[18] Josh Weberruss, Lindsay Kleeman, and Tom Drummond, "Orb feature extraction and matching in hardware," in *Proceedings of the Australasian Conference on Robotics and Automation*, 2015.

[19] Nadir Mohamed Belmessaoud, Youcef Bentoutou, and Miloud Chikr El-Mezouar, "Fpga implementation of feature detection and matching using orb," *Microprocessors and Microsystems*, vol. 94, pp. 104666, 2022.

[20] Phong Tran, Thinh Hung Pham, Siew-Kei Lam, Meiqing Wu, and Bhavan A. Jasani, "Stream-based orb feature extractor with dynamic power optimization," in *Proceedings of the International Conference on Field Programmable Technology*, 2018.

[21] Runze Liu, Jianlei Yang, Yiran Chen, and Weisheng Zhao, "eSLAM: An Energy-Efficient Accelerator for Real-Time

ORB-SLAM on FPGA Platform," in *Proceedings of the 56th Annual Design Automation Conference*, 2019.

[22] Josh Weberruss, Lindsay Kleeman, David Boland, and Tom Drummond, "Fpga acceleration of multilevel orb feature extraction for computer vision," in *Proceedings of the 27th International Conference on Field Programmable Logic and Applications*, 2017, pp. 1–8.

[23] Andre Costa, Pedro Tomás, Nuno Roma, and Nuno Neves, "Real-time orb accelerator with ros integration for embedded fpga socs," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1–5, 2025.

[24] Weikang Fang, Yanjun Zhang, Bo Yu, and Shaoshan Liu, "FPGA-based ORB feature extraction for real-time visual SLAM," in *Proceedings of the International Conference on Field Programmable Technology*, 2017, pp. 275–278.

[25] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[26] Anastasios I. Mourikis and Stergios I. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 3565–3572.

[27] Vibhakar Vemulapati and Deming Chen, "FSLAM: an Efficient and Accurate SLAM Accelerator on SoC FPGAs," in *Proceedings of the 21st International Conference on Field-Programmable Technology*, 2022.

[28] Rui Xue, Wenming Li, Haibin Wu, Cheng Guo, Yi Li, Xiaochun Ye, and Dongrui Fan, "A High-Performance Dataflow-Based ORB Extractor Accelerator for SLAM," in *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design*, 2025, pp. 1–7.

[29] Raúl Taranco, José-María Arnau, and Antonio González, "LOCATOR: Low-power ORB accelerator for autonomous cars," *Journal of Parallel and Distributed Computing*, vol. 174, pp. 32–45, 2023.

[30] Liuxin Sun, Junyu Wei, Shaojing Su, and Peng Wu, "SOLO-SLAM: A parallel semantic slam algorithm for dynamic scenes," *Sensors*, vol. 22, no. 18, pp. 6977, 2022.

[31] Alles Rebel, Nikil Dutt, and Bryan Donyanavard, "Oasis: Optimized adaptive system for intelligent slam," *ACM Transactions on Embedded Computing Systems*, vol. 24, no. 5s, 2025.

[32] Tao Peng, Dingnan Zhang, Ruixu Liu, Vijayan K. Asari, and John S. Loomis, "Evaluating the power efficiency of visual slam on embedded gpu systems," in *Proceedings of the IEEE National Aerospace and Electronics Conference*, 2019, pp. 117–121.

[33] Michael Burri, Janosch Nikolić, Pascal Gohl, Thomas Schneider, Jörn Rehder, Sammy Omari, Markus Achtelik, and Roland Siegwart, "The EuRoC micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

Porting the Striped Smith-Waterman Library to RISC-V via LLM-Driven Translation

Alejandro Fernández Camello, Manuel Prieto Matías, and Carlos García Sánchez[1]

Abstract— **Manually porting SIMD-optimized libraries across instruction set architectures is costly and error-prone, particularly when optimized kernels rely on architecture-specific idioms. We introduce a fully automated, two-phase LLM-driven pipeline for translating x86 SSE code to native RISC-V Vector (RVV) intrinsics. The first phase produces a functionally correct translation based on `sse2rvv` [1], while the second rewrites that translation to exploit RVV's scalable vector-length model. Both phases use an iterative compile-fix loop in which the LLM relies exclusively on compiler and simulator feedback to detect and repair errors. We evaluate the pipeline on the Striped Smith-Waterman (SSW) library, a production-grade genomics kernel. Relative to a manually written naive Smith-Waterman baseline, Phase 1 achieves up to 21.7× speedup, showing that a compatibility-layer-guided translation strategy can serve as an effective foundation for cross-ISA porting. Spike [2] simulation confirms that the Phase 2 transformation remains functionally correct across vector lengths from 128 to 4096 bits. On our target platform with a 256-bit vector length, Phase 2 delivers a further 1.30× speedup. Overall, the fully automated pipeline achieves performance and correctness comparable to those of a manually optimized implementation developed by a human expert with AI assistance.**

Keywords— **RISC-V, Vector Extensions, SIMD Translation, Smith-Waterman, LLM, SSE2, sse2rvv.**

I. Introduction

HIGH-PERFORMANCE software often relies on hand-optimised vector code to accelerate its most expensive kernels. In bioinformatics, for example, libraries and tools such as Minimap2 [3], Parasail [4], and SSW [5] use Intel SSE2 and AVX2 intrinsics to achieve substantial speedups in sequence alignment. While effective, this approach tightly couples performance-critical code to a particular ISA, making portability increasingly difficult. This is especially relevant for RISC-V, whose Vector extension (RVV) [6] introduces a vector-length-agnostic programming model that differs significantly from the fixed-width SIMD model used by conventional ISA extensions.

Manually porting SIMD kernels is costly and requires more than replacing individual intrinsics: developers must often reformulate loop structures, memory-access patterns, and data layouts to exploit variable-length vectors efficiently. As a result, porting becomes a complex process that demands substantial expertise in the low-level details of both the source and target architectures.

In this work, we investigate the porting of the SSW library, a production-grade genomics kernel, to RVV as a case study in translating fixed-width SIMD code to the RISC-V Vector extension. SSW is a particularly challenging target because its implementation of Farrar's striped query-profile technique [7] is closely tied to x86 SSE2's fixed-width 128-bit SIMD model.

Compatibility layers offer a partial solution. Projects such as sse2neon [8] and sse2rvv [1] provide drop-in headers that emulate SSE2 semantics on ARM NEON and RVV, respectively, enabling rapid functional porting while limiting exploitation of vector lengths beyond the original 128-bit width. However, for intrinsic-heavy libraries with complex data-flow patterns – such as Farrar's striped Smith-Waterman implementation – these headers often require manual intervention, as not all SSE2 idioms have direct equivalents in the target ISA. Higher-level wrappers such as SIMD Everywhere (SIMDe) [9] and Highway [10] abstract across ISAs, but require non-trivial code refactoring and may not cover all intrinsic patterns used in legacy codebases.

Recent work has explored LLM-based approaches to code translation [11], including SIMD intrinsic migration. IntrinTrans [12] uses fine-tuned models to translate individual intrinsic calls from x86 to RVV, while Lin et al. [13] address the rewriting of Arm SVE intrinsics to RVV. These efforts demonstrate the feasibility of LLM-assisted intrinsic translation, but stop short of end-to-end library porting with integrated correctness validation.

In this paper, we present an *automated two-phase pipeline* that translates x86 SSE2 libraries to native, vector-length-agnostic RVV code using a lightweight LLM in a compiler-feedback loop. Phase 1 produces a functionally correct binary via the `sse2rvv.h` compatibility header; Phase 2 replaces the compatibility calls with native RVV intrinsics written in a vector-length-agnostic style to exploit the target architecture's full vector length. Both phases are library-agnostic: the LLM resolves compilation and runtime errors using only compiler diagnostics and simulator feedback, without library-specific transformation rules.

We evaluate the pipeline on the SSW library and make the following contributions:

- A two-phase, LLM-driven translation method that decomposes the SSE2-to-RVV porting problem into two tractable sub-tasks, producing the same sequence-alignment output as the original across vector lengths from 128 to 4096 bits.
- An empirical evaluation showing that the Phase 1 translation delivers up to 21.7× speedup

[1]Facultad de Informática, Universidad Complutense de Madrid, Spain e-mail: {alejfe31,mpmatias,garsanca}@ucm.es.

over a naive baseline, while Phase 2 adds a further 1.30× on our 256-bit target platform.
- Evidence that the fully automated pipeline generates native RVV code of comparable quality to that produced by a human expert assisted by AI tools.

The complete translation pipeline and all resulting code are publicly available at `https://github.com/alexfdez1010/riscv-translator`.

The remainder of the paper is organised as follows. Section II reviews the Smith-Waterman algorithm, the RISC-V Vector extension, and related work on SIMD portability. Section III describes the translation method. Section IV presents the experimental evaluation. Section V concludes and outlines directions for future work.

II. Background

A. The Smith-Waterman algorithm

The Smith-Waterman (SW) algorithm [14] computes the optimal local alignment between two biological sequences. Given a query of length m and a reference of length n, it fills an $m \times n$ scoring matrix H using the recurrence

$$H_{i,j} = \max \begin{pmatrix} H_{i-1,\,j-1} + s_{i,j}, \\ H_{i-1,\,j} - d, \\ H_{i,\,j-1} - d \end{pmatrix} \tag{1}$$

where $s_{i,j} = s(q_i, r_j)$ is the substitution score and d is the gap penalty. The algorithm guarantees an optimal local alignment but has $O(mn)$ time and space complexity, making SIMD acceleration essential for practical use in genomics pipelines [3].

Farrar [7] introduced the *striped* query-profile technique, which reorders the scoring matrix so that consecutive elements in a SIMD register correspond to residues separated by a fixed stride. This layout eliminates the data dependency along the diagonal that limits naive anti-diagonal vectorization, achieving a 6× speedup over earlier SIMD implementations. The SSW library [5] builds on Farrar's approach with SSE intrinsics and is deployed in production tools such as MOSAIK and TANGRAM.

B. RISC-V vector extensions

The RISC-V Vector Extension (RVV), ratified as version 1.0 in 2021 [6], adopts a *vector-length-agnostic* (VLA) execution model. Unlike fixed-width packed-SIMD ISAs such as x86 SSE (128 bits) and AVX2 (256 bits), RVV does not encode a vector width in the binary interface. Instead, software determines the active vector length dynamically. As a result, a single RVV binary can execute correctly across processors with different vector lengths, provided that the code is written in a truly VLA style.

Key architectural features of RVV that distinguish it from SSE include:

- **Vector-length-agnostic execution:** RVV code can execute correctly across implementations with different hardware vector lengths.
- **Predicated execution:** RVV provides mask-controlled operations, enabling efficient handling of conditional computation and loop tails.
- **Decoupled vector configuration:** Element width, active vector length, and register grouping are configured dynamically rather than being fixed by the ISA.
- **Flexible vector memory operations:** RVV supports unit-stride, strided, indexed, and segmented accesses.

Several groups have evaluated early RVV hardware for HPC workloads. Lee et al. [15] benchmarked the first commercially available RVV 1.0 boards, while Brown et al. [16] assessed the 64-core Sophon SG2042 for general-purpose computing. Vizcaíno et al. [17] studied the impact of long vector lengths on HPC kernels, confirming that VLEN-agnostic code scales effectively across different implementations.

C. SIMD portability approaches

Three main strategies exist for porting SIMD code across ISAs:

Drop-in compatibility headers.

sse2neon [8] maps SSE intrinsics to ARM NEON through a single header inclusion, and sse2rvv [1] does the same for RVV. These compatibility layers provide the shortest path to a working binary, but limit performance to the source ISA's fixed 128-bit width, preventing effective use of wider target vectors. In addition, intrinsic-heavy codebases with complex data-flow patterns – especially those relying on data-layout optimizations tied to a specific SIMD width, such as the striped Smith-Waterman kernel – often require manual fixes to compile successfully through such layers.

Abstraction libraries.

Google's Highway [10] and SIMD Everywhere (SIMDe) [9] provide portable APIs that map to native code on each platform. Highway targets VLA architectures explicitly and can generate efficient RVV code, but adopting it requires rewriting the source to use the library's API – a non-trivial effort for large, intrinsic-heavy codebases.

Source-level rewriting. Lin et al. [13] propose an automated framework for rewriting Arm SVE intrinsics to RVV, addressing translation between two vector-length-agnostic ISAs. IntrinTrans [12] uses fine-tuned LLMs to translate individual x86 intrinsic calls to RVV equivalents. Our work differs in two key respects: (1) we target end-to-end library translation rather than isolated intrinsic substitution, and (2) we use a general-purpose LLM without fine-tuning, relying instead on compiler feedback to guide convergence through iterative refinement until the translation compiles and runs correctly.

D. LLMs for code translation

Large language models have shown strong performance on code translation tasks across high-level languages, but the low-level, ISA-specific nature of SIMD intrinsics poses additional challenges.

Pan et al. [11] systematically studied bugs introduced by LLMs during code translation, identifying semantic mismatches and silent correctness errors as the primary failure modes — precisely the risks that motivate our integrated validation pipeline.

Our approach mitigates these risks by treating the LLM as a *patch proposer* within a closed-loop system: every change is compiled, executed, and validated against a reference before acceptance. The LLM never operates in isolation; it receives concrete error messages and must resolve them to proceed. This design ensures that the final output is correct by construction, regardless of intermediate translation errors.

III. Translation method

Porting the SSW library [5] from x86 to RISC-V poses a significant challenge: the core Smith-Waterman kernel is tightly coupled to SSE intrinsics, using operations such as saturating arithmetic (`_mm_adds_epu8`, `_mm_subs_epu8`), packed comparisons (`_mm_cmpgt_epi8`), and byte-level shuffles (`_mm_srli_si128`) that have no direct equivalents in RVV. A manual rewrite would require deep expertise in both the SSE and RISC-V Vector (RVV) instruction sets, together with detailed knowledge of the algorithm itself—Farrar's striped query profile technique [7] organizes score vectors specifically to exploit 128-bit SIMD lanes.

We address this with a two-phase, LLM-driven pipeline that decomposes the problem into two tractable sub-tasks: (1) a semantics-preserving translation to a compatibility layer, and (2) an optimization pass that replaces the compatibility layer with native RVV intrinsics. Each phase is implemented as a *ReAct* agent [18], a framework that interleaves reasoning and acting in an iterative loop. In the *Reason* step, the LLM analyses the feedback from the previous iteration –compiler diagnostics, simulator traces, or output mismatches – and proposes a minimal code patch. In the *Act* step, the patch is applied and the resulting code is run through a two-stage validation chain (detailed in Section III-A) that ends with output comparison against the original Intel SSE reference. The pipeline is fully generic: the agent resolves errors using feedback alone, without any hardcoded, library-specific transformation rules.

A. Phase 1: SSE to sse2rvv.h

The goal of Phase 1 is to produce a functionally correct RISC-V binary that preserves the original 128-bit SSE semantics. It relies on `sse2rvv.h` [1], a drop-in compatibility header that maps SSE intrinsic functions to equivalent RVV operations at a fixed 128-bit vector width.

The translation proceeds in two steps: a deterministic pre-processing step followed by the ReAct loop.

Pre-processing.

All SSE header inclusions (`emmintrin.h`, `xmmintrin.h`) are systematically replaced with `#include "sse2rvv.h"`. This transformation is entirely deterministic and requires no LLM involvement.

A critical constraint enforced throughout Phase 1 is the *16-byte rule*. Even when the underlying hardware provides vector registers wider than 128 bits, all `sse2rvv.h` operations must preserve SSE semantics and therefore process exactly 16 bytes. Without this constraint, the translated SSW library produced correct results on 128-bit vector implementations, but silently corrupted the scoring matrix on wider RVV implementations.

ReAct loop. After pre-processing, the ReAct agent takes over. In each iteration the agent first *reasons* about the feedback from the previous iteration (or the initial pre-processed source on the first iteration), then *acts* by proposing a minimal search-and-replace patch. Each proposed patch is syntactically validated before application; malformed patches are rejected with explanatory feedback that feeds the next Reason step. Once a valid patch is applied, the modified code is run through the two-stage validation chain that constitutes the Act step:

1. *Simulator validation.* The source is compiled for the `rv64gcv` target inside a Docker container equipped with the RISC-V GNU toolchain (`riscv64-unknown-elf-gcc` with `-O2`) and the resulting binary is executed under the Spike RISC-V ISA simulator [2] with the `pk64` proxy kernel. Compilation errors or runtime failures (segfaults, assertion failures, etc.) are fed back to the agent as the observation for the next Reason step.
2. *Hardware validation.* The source is compiled natively on the Banana Pi BPI-F3 [19], a RISC-V development board based on the SpacemiT K1 SoC [20] featuring an octa-core in-order processor with RVV 1.0 support and 256-bit vector units, and executed on the board. The output is compared record-by-record against a reference run of the original SSE code on an x86 host: for each aligned sequence pair, the target name, query name, optimal alignment score, strand orientation, target end position, and query end position are checked for exact equality. If compilation fails, execution fails, the number of output records differs, or any field in any record does not match, the specific diagnostics or mismatches become the observation for the next Reason step.

The two stages execute strictly in order; the chain stops at the first failure and the diagnostic output is returned to the agent. A stage that succeeded in a previous iteration may fail after a new patch, and in that case the agent receives the new failure feedback regardless of prior success. The loop terminates only when both stages pass in a single iteration, at which point Phase 1 is considered complete. A fixed iteration limit prevents unbounded execution.

B. Phase 2: Native RVV rewriting

Phase 1 produces functionally correct code, but each vector instruction remains effectively constrained to 16 bytes regardless of the target architecture. On processors with 256-bit or wider vector registers, this leaves vector capacity underutilized. Phase 2 rewrites the Phase 1 output into vector-length-agnostic native RVV code that scales with the hardware vector length.

Why a separate phase?

Early experiments showed that asking the LLM to simultaneously map SSE intrinsics and generalize the translation to wider vector configurations introduced too many degrees of freedom. The model failed to produce reliable translations, confusing SSE 128-bit lane semantics with full-width RVV operations and thus generating code that compiled but produced incorrect alignment scores. By constraining Phase 1 to preserve exact SSE semantics, we reduced the task to a tractable problem with clear success criteria. Phase 2 then starts from known-correct code and focuses on a single, well-defined objective: generalizing the translation to vector-length-agnostic RVV execution.

Vector-length-agnostic rewriting pipeline. Phase 2 uses the same ReAct agent architecture, but with a different prompt and objective. The agent incrementally replaces `sse2rvv.h` wrapper calls with native `<riscv_vector.h>` intrinsics, processing the code in multiple passes. Each pass targets a subset of the remaining SSE calls and constitutes one ReAct iteration: the agent reasons about the current code, proposes a patch, and the patch is validated through the same two-stage Act chain as Phase 1—simulator validation (cross-compilation and Spike execution) followed by hardware validation (native compilation, execution, and record-by-record output comparison against the Intel SSE reference). As in Phase 1, if any stage in the chain fails, its diagnostic output feeds the next Reason step and the Act chain restarts from the first stage. The key difference from Phase 1 is the termination condition: the agent must explicitly signal that the rewriting is complete by emitting a sentinel token (`ALL_WIDENED`), indicating that no `sse2rvv.h` intrinsics remain in the source. The loop terminates only when the agent emits this token *and* both validation stages pass in the same iteration.

The resulting code is vector-length agnostic: it adapts to the hardware vector length at runtime and processes proportionally more data elements per instruction. On a machine with 256-bit vector registers, the rewritten code processes twice as many score-profile entries per iteration of the inner Smith-Waterman loop as the corresponding 128-bit version.

Figure 1 illustrates the complete ReAct pipeline with the two-stage validation chain used in both phases.

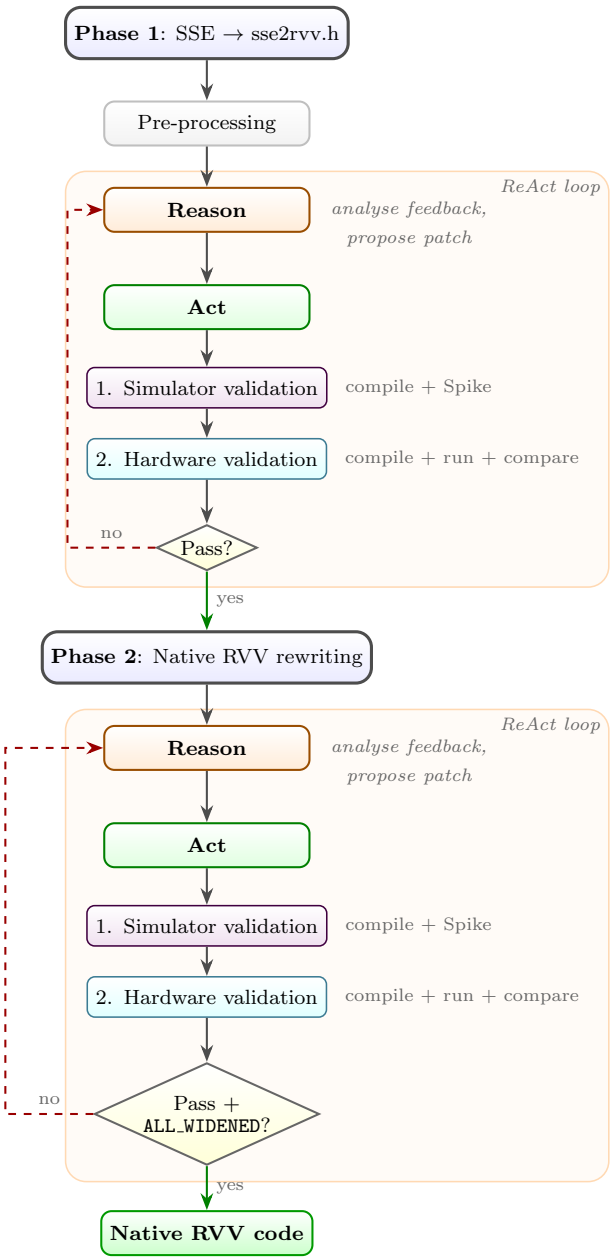


Fig. 1: Overview of the two-phase ReAct pipeline. Each phase iterates a Reason–Act cycle with a two-stage validation chain: (1) simulator validation (cross-compilation and Spike execution) and (2) hardware validation (native compilation, execution, and output comparison against the x86 SSE reference). Phase 1 terminates when both stages pass; Phase 2 additionally requires the agent to signal that all compatibility calls have been replaced.

C. LLM configuration and cost

The ReAct agent is instantiated with the MiniMax-M2.7 model [21] accessed through the OpenRouter API. The sampling temperature is set to 0 to reduce output variability. All code changes are proposed exclusively through search-and-replace blocks, which are parsed by a dedicated module robust to whitespace variations, markdown fences, and other common LLM formatting artefacts.

Each phase typically requires 3–8 ReAct iterations. At approximately $0.05 USD per phase, the total cost of a complete two-phase translation remains under $0.10 USD, substantially lower than that of a fully manual porting effort.

D. Translation variants

The pipeline produces four distinct code variants, summarized in Table I.

Table I: Code variants produced by the translation pipeline.

Variant	Phase	Description
Naive	—	RVV baseline using wavefront parallelism
SSE→RVV	1	Uses sse2rvv.h at fixed 128-bit width
Native RVV (manual)	2	AI-assisted expert native RVV rewriting
Native RVV (auto)	2	Fully automated native RVV rewriting via the LLM pipeline

The naive variant serves as a lower bound: rather than translating the SSE code, it reimplements the Smith-Waterman algorithm from scratch using native RVV intrinsics with wavefront (anti-diagonal) parallelism, the simplest vectorization strategy for dynamic-programming kernels. It serves exclusively as a performance baseline for comparison.

The native RVV (manual) variant was produced by a human expert assisted by AI tools, starting from the Phase 1 output (SSE→RVV via sse2rvv.h) and tasked with replacing the compatibility layer with native RVV intrinsics. The expert had full access to the repository (source files, build system, test harness, and validation tools) and iteratively verified correctness at every step using the same infrastructure described in Section III-A. A comprehensive RVV intrinsics reference document was used to provide detailed documentation of the target instruction set. Comparing this variant against the fully automated Phase 2 output allows us to assess whether the lightweight pipeline matches the quality of expert-guided rewriting.

IV. Experimental evaluation

A. Case study: the SSW library

The Striped Smith-Waterman (SSW) library [5] implements the optimal local sequence alignment algorithm using SSE SIMD intrinsics. It employs Farrar's striped query profile technique [7], which reorders the substitution scoring matrix to maximise data-level parallelism within fixed-width SIMD registers. The library is used in production genomics pipelines including the read mapper MOSAIK [22] and the structural variant caller TANGRAM [23], making it a representative target for SIMD translation.

The test harness aligns a fixed set of 100 short reads (54 bp each) against a single reference sequence of variable length, reporting the optimal alignment score, strand orientation, and target/query endpoint positions for each of the 100 alignments.

B. Experimental setup

Platform. All RISC-V benchmarks are executed on a Banana Pi BPI-F3 development board [19] based on the RVA22 standard and the first commodity processor to support the RVV 1.0 vector extension and 256-bit vector registers, which integrates the SpacemiT K1 SoC [20], an octa-core X60 processor with an eight-stage dual-issue in-order pipeline. Its memory hierarchy is notably lean, lacking an L3 cache and relying on 32KB L1 caches per core with a shared 1MB L2 cache (divided into two 512KB banks). Equipped with up to 16GB of LPDDR4-2666, the system provides a peak bandwidth of 10.6 GB/s. Despite its modest throughput, the platform is distinguished by its high energy efficiency, operating with a TDP of 3-5W.

The benchmarks are compiled for the `rv64gcv` target ISA. Code is compiled with Clang for the `riscv64-linux-gnu` target at `-O2`. The x86 reference runs are compiled with GCC at `-O2` and executed on an x86_64 host running Debian 12 with Linux kernel 6.1.0. Simulator-based validation uses Spike with configurable vector lengths ranging from 128 to 4096 bits.

Datasets. We use five reference sequences of increasing length (Table II) to exercise the translation under varying computational loads. Each reference file contains a single chromosomal region, ranging from approximately 1 000 to 10 000 000 base pairs, sourced from the SSW library's own test suite. The query set is a separate file of 100 reads of 54 bp each; performance therefore scales with reference length while the query workload remains constant. The variation analysis across all code variants is performed using the `10k.fa` dataset as the standard workload, as it provides sufficient computational load to validate the correctness of the translation and it is quick enough to run for the validation needed.

Table II: Reference sequences used for benchmarking. Each file contains a single chromosomal region; the 100-read query set is shared across all experiments.

Reference	Role in evaluation
`1k.fa`	Startup overhead, small-scale
`10k.fa`	Standard workload, correctness
`100k.fa`	Medium-scale performance
`1M.fa`	Large-scale sustained throughput
`10M.fa`	Maximum-scale stress test

Methodology. Each combination of code variant (Section III-D) and dataset is executed 10 times on the RISC-V hardware. *Cell updates per second* (CUPS) is a commonly used performance measure in the Smith-Waterman scenario, because it removes the dependency on the particular sequences used for testing. A single cell update comprises the complete computation of one cell in matrix H, including all memory operations and the corresponding computation of the values in the E and F arrays. The GCUPS metric (billions of CUPS) is calculated as

$$\mathrm{GCUPS} = \frac{\mathrm{m} \times \mathrm{n}}{\mathrm{t} \times 10^9}, \tag{2}$$

where m and n are the lengths of the query and reference sequences, respectively, and t is the exe-

cution time of the alignment phase only. The timing is taken from the SSW test harness's own internal measurement, so FASTA parsing and other file I/O are excluded; this isolates the cost of the sequence alignment kernel itself, which is the part affected by the SSE→RVV translation. We report mean, median, and standard deviation of GCUPS for each configuration. After every run, the output is validated against the x86 SSE reference using the same record-by-record comparison described in Section III-A: alignment scores, strand, target and query names, and endpoint positions must match exactly. Only experiments that pass correctness validation are recorded.

C. Speedup over the naive baseline

Figure 2 shows the speedup of the Phase 1 translation (SSE→RVV via `sse2rvv.h`) over the naive baseline. The SSE→RVV translation consistently outperforms the naive variant across all dataset sizes, from 17.2× on the smallest dataset (1k.fa) to **21.8**× on 100k.fa.

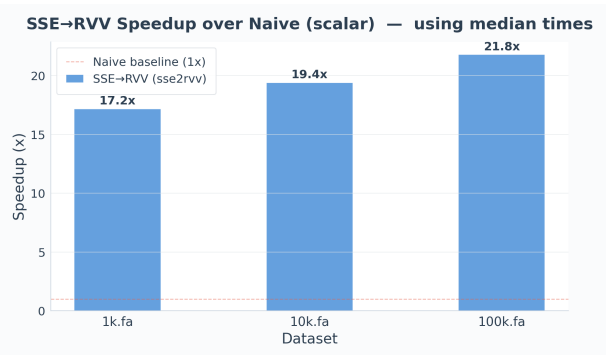


Fig. 2: Speedup of the SSE→RVV translation over the naive baseline, computed from median GCUPS. The `sse2rvv.h` compatibility layer delivers up to 21.8× improvement.

Even on the smallest dataset (`1k.fa`), the translation achieves a 17.2× speedup, indicating that Farrar's striped query profile technique exposes substantial data-level parallelism even at small scales. The advantage grows further with dataset size: on `100k.fa`, the SSE→RVV translation reaches a median of 0.169 GCUPS, compared with 0.008 GCUPS for the naive variant. This confirms that even a 128-bit-constrained translation based on `sse2rvv.h` captures the essential data-level parallelism of the original SSE implementation.

D. Phase 2 gains

Figure 3 compares the two RVV native variants against the Phase 1 baseline.

The benefits of the Phase 2 transformation are already visible on the smallest dataset (`1k.fa`), yielding speedups of 1.25× for the manual version and 1.23× for the automated version, and grow consistently with dataset size:

- On `100k.fa`, the manual Phase 2 version reaches **1.28**× and the automated version **1.27**×.

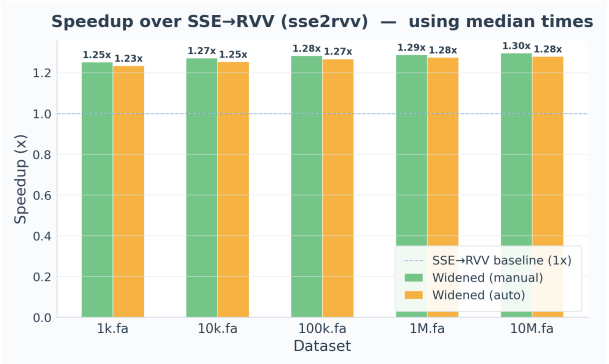


Fig. 3: Speedup of the native RVV variants over the SSE→RVV (`sse2rvv.h`) baseline, computed from median GCUPS. Both manual and automated rewriting achieve similar gains, reaching 1.30× on the 10M dataset.

- On `1M.fa`, the manual Phase 2 version reaches **1.29**× and the automated version **1.28**×.
- On `10M.fa`, the manual Phase 2 version reaches **1.30**× and the automated version **1.28**×, showing that the advantage of the Phase 2 transformation is sustained at larger input sizes.

Crucially, the automated Phase 2 output matches the performance of the expert-produced manual version to within 2%. This demonstrates that the lightweight automated LLM pipeline can produce native RVV code comparable to that of a human expert assisted by AI tools.

E. GCUPS throughput

Figure 4 presents the mean GCUPS throughput across all translated variants and datasets.

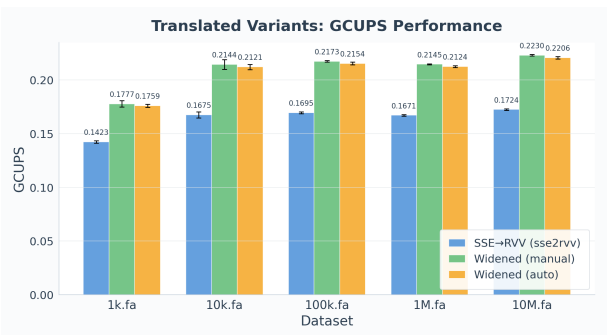


Fig. 4: Mean GCUPS throughput, with standard deviation error bars, for the three translated variants across all datasets. The native RVV variants consistently outperform the 128-bit baseline, especially on larger inputs.

On the `1M.fa` dataset, the Phase 1 translation achieves a mean throughput of 0.167 GCUPS, whereas the Phase 2 variants reach 0.215 GCUPS (manual) and 0.212 GCUPS (auto). On the largest dataset (`10M.fa`), the Phase 1 baseline reaches 0.172 GCUPS, compared with 0.223 GCUPS (manual) and 0.221 GCUPS (auto) for the Phase 2 variants. While relative standard deviations indicate stable and reproducible performance, these results remain significantly below the throughput observed in x86-based architectures. This performance gap is primarily attributed to the memory-intensive nature of the Smith-Waterman algorithm, which is bottle-

necked by the SpacemiT K1's constrained memory hierarchy. Specifically, the limited L1/L2 cache capacities and a modest peak main memory bandwidth of 10.6 GB/s hinder the SoC's ability to sustain the high data-feed rates required for optimal vector execution [24].

F. GCUPS distribution

Figure 5 shows boxplots of the GCUPS distributions for the three translated variants, with individual measurements overlaid.

Several observations emerge from these distributions:

- On the `1k.fa` dataset, all three variants achieve similar throughput values (0.14–0.18 GCUPS), indicating that fixed overheads dominate at small scale.
- From `10k.fa` onward, the Phase 2 variants clearly separate from the SSE→RVV baseline, with the gap stabilizing at approximately 0.05 GCUPS.
- On `10M.fa`, the manual Phase 2 variant achieves a median of 0.223 GCUPS, compared with 0.172 GCUPS for the SSE→RVV baseline, corresponding to a 30% throughput improvement.
- From `100k.fa` onward, the manual Phase 2 variant exhibits the tightest distributions, indicating more consistent performance once fixed overheads are amortized. By contrast, on `1k.fa`, the SSE→RVV baseline shows the tightest distribution, again reflecting the dominance of fixed setup costs at small scales.

G. Scaling behavior

Figure 6 shows how GCUPS throughput scales with dataset size for all four code variants, including the naive baseline.

The naive variant achieves approximately 0.008 GCUPS regardless of dataset size, suggesting that its unoptimized scalar loop is limited by computation rather than memory bandwidth. The vectorized variants achieve higher throughput as dataset size increases: the SSE→RVV translation improves from 0.142 GCUPS on `1k.fa` to 0.172 GCUPS on `10M.fa` as the fixed overhead of alignment setup is amortized over a larger amount of computation. The Phase 2 variants follow the same trend at a higher throughput level, reaching 0.223 GCUPS (manual) and 0.221 GCUPS (auto) on `10M.fa`. This shows that the Phase 2 transformation improves not only absolute throughput but also performance at larger input sizes.

H. Correctness across vector lengths

To verify that the Phase 2 code is truly vector-length agnostic, we executed it under the Spike simulator at every power-of-two vector width from 128 to 4096 bits. At each configuration, the sequence-alignment output was validated against the original Intel SSE implementation running natively on x86 using the same field-level comparison described in Section III-A. All six configurations (128, 256, 512, 1024, 2048, and 4096 bits) produced identical alignment results, confirming that the translation correctly handles width-dependent loop tiling, remainder elements, and scalable vector memory access patterns.

I. Translation cost and effort

The complete two-phase translation of the SSW library, from x86 SSE source to vector-length-agnostic native RVV code, required under $0.10 USD in LLM API costs and completed within minutes. Phase 1 converged in 5 iterations, while Phase 2 required 7 iterations. By comparison, the manual Phase 2 implementation was produced by a human expert assisted by AI tools, requiring substantially more time and effort while yielding nearly identical performance (see Section IV-D).

V. Conclusions

Porting SIMD-optimized libraries across instruction set architectures has historically required either expensive manual rewriting by experts familiar with both ISAs or the use of compatibility layers that sacrifice performance by constraining execution to the source register width. This paper shows that LLMs, guided only by compiler and simulator feedback, can bridge this gap by translating a production genomics kernel from x86 SSE to native, vector-length-agnostic RISC-V Vector code while preserving the sequence alignment results of the original implementation.

The key insight is that decomposing the translation into two phases –first achieving functional correctness through a compatibility header, then generalizing the result to exploit the full hardware vector width – reduces the overall problem into subtasks that a general-purpose LLM can handle reliably within a closed-loop system. The Phase 1 translation already captures the essential data-level parallelism of the original SSE implementation, delivering up to 21.7× speedup over a naive baseline. Phase 2 adds a further 1.30× by removing the 128-bit bottleneck and producing code that remains correct across vector lengths from 128 to 4096 bits.

Perhaps most significantly, the fully automated pipeline achieves performance and correctness comparable to those of a manual Phase 2 implementation produced by a human expert assisted by AI tools. This suggests that, for structured translation tasks with clear correctness criteria, the combination of a lightweight LLM and a robust validation loop can match the effectiveness of expert-guided alternatives.

It is important to note, however, that the absolute performance gains observed on the SpacemiT K1 are partially bottlenecked by the SoC's hardware constraints, specifically its lean memory hierarchy and modest 10.6 GB/s main memory bandwidth. In memory-intensive kernels such as Smith-Waterman, these limitations prevent the vector units from reaching their peak theoretical throughput. Nevertheless,

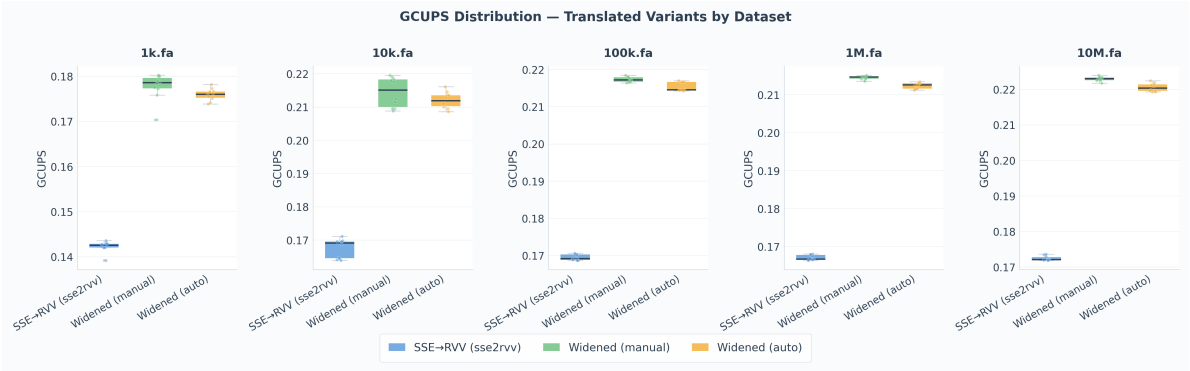


Fig. 5: GCUPS distributions for the three translated variants across all five datasets, with individual measurements shown as scatter points. The native RVV variants consistently achieve higher throughput, with performance differences becoming more apparent on the larger datasets.

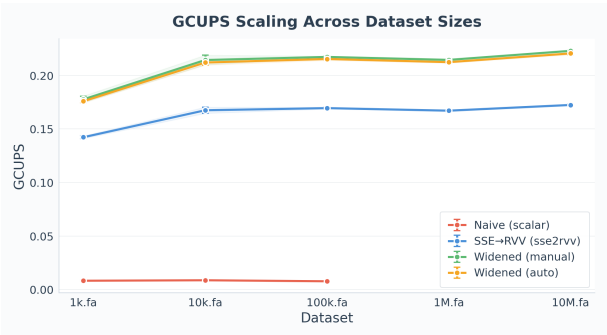


Fig. 6: GCUPS scaling across dataset sizes for all four variants.

the significance of these results transcends the current hardware: by producing vector-length-agnostic code that scales seamlessly, this pipeline establishes a viable path for migrating mission-critical HPC libraries from legacy x86 codebases to the next generation of high-performance RISC-V processors. As upcoming RISC-V designs incorporate wider memory buses and deeper cache hierarchies, the LLM-generated code will be positioned to exploit those resources without further manual intervention.

More broadly, LLM-driven translation occupies a practical middle ground that was previously unavailable. Traditional translation of low-level SIMD code has largely been limited to pattern-matching approaches that cannot handle the semantic complexity of operations such as Farrar's striped query profile, where algorithmic structure is tightly coupled to the source ISA's register geometry. Fully manual porting, while capable of producing highly optimized code, demands scarce expertise and substantial engineering effort—particularly as the RISC-V ecosystem grows and the volume of x86 code requiring migration increases. The approach presented here shows that LLMs can bridge the semantic gap between fixed-width and vector-length-agnostic SIMD models, handling the restructuring of loop tiling, memory access patterns, and lane-dependent idioms that defeat simpler translation strategies.

Several directions remain open for future work. First, the success of the LLM-driven agentic pipeline suggests it could be extended to explore alternative vectorization schemes beyond Farrar's striped approach. Within the context of the Smith-Waterman algorithm, future iterations could evaluate the automated implementation of diagonal-based or v-parallel strategies to determine which algorithmic structure best aligns with RISC-V's VLA model. Second, the pipeline should be evaluated on additional SIMD-intensive domains—such as image processing, cryptography, and signal processing—to assess its generality. Third, extending the approach to AVX2 and AVX-512 source code would broaden its applicability to a wider range of legacy x86 codebases. Finally, as high-performance RISC-V hardware with wider vector units and enhanced memory subsystems becomes available, real-hardware benchmarking beyond 256-bit vector widths will be essential to confirm that the generated code scales effectively in production HPC environments.

Acknowledgements

This paper has been partially funded by the EU (FEDER), the Spanish MINECO under grants PID2024-158311NB-I00 and PCI2024-162578-3 funded by MCIN/AEI/10.13039/501100011033 and by European Union "ERDF A way of making Europe" and the NextGenerationEU/PRT.

Additionally, Alejandro F. C. has been supported by a predoctoral research contract funded by Universidad Complutense de Madrid and Banco Santander.

Referencias

[1] Pattonkan, "sse2rvv," 2024, A translator from Intel SSE intrinsics to RISC-V Vector Extension.

[2] RISC-V Software Source, "Spike RISC-V ISA Simulator," `https://github.com/riscv-software-src/riscv-isa-sim`, Accessed: 2026-04-10.

[3] Heng Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018.

[4] Jeff Daily, "Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments," *BMC Bioinformatics*, vol. 17, no. 1, pp. 81, 2016.

[5] Mengyao Zhao, Wan-Ping Lee, Erik P. Garrison, and Gabor T. Marth, "SSW Library: An SIMD Smith-Waterman C/C++ library for use in genomic applications," *PLOS ONE*, vol. 8, no. 12, pp. e82138, 12 2013.

[6] RISC-V International, "RISC-V "V" vector extension, version 1.0," Tech. Rep., RISC-V International, 2021.

[7] Michael Farrar, "Striped Smith–Waterman speeds database searches six times over other SIMD implemen-

tations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 2007.

[8] sse2neon contributors, "sse2neon," 2024, A translator from Intel SSE intrinsics to Arm/Aarch64 NEON intrinsics.

[9] Evan Nemerson and SIMDe contributors, "SIMDe: SIMD everywhere," 2024, Portable implementations of SIMD intrinsics for non-native architectures.

[10] Google, "Highway," 2026, A C/C++ auto-vectorizing compiler for high-performance code.

[11] Rangeet Pan, Ali Reza Ibrahimzada, Rahul Krishna, Divya Sanber, Lambert Pouguem Wassi, Michele Merber, and Reyhaneh Jabbarvand, "Lost in translation: A study of bugs introduced by large language models while translating code," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE)*, 2024.

[12] Liutong Han, Zhiyuan Tan, Hongbin Zhang, Pengcheng Wang, Chu Kang, Mingjie Xing, and Yanjun Wu, "Intrintrans: Llm-based intrinsic code translator for risc-v vector," 2026.

[13] Jhih-Kuan Lin, Yu-Lun Yang, Hung-Ming Lai, and Jenq-Kuen Lee, "Rewriting and optimizing vector length agnostic intrinsics from Arm SVE to RVV," in *Proceedings of the ICPP Workshops*, 2024.

[14] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981.

[15] Joseph K. L. Lee, Maurice Jamieson, Nick Brown, and Ricardo Jesus, "Test-driving RISC-V vector hardware for HPC," in *High Performance Computing – ISC High Performance 2023*, 2023, Lecture Notes in Computer Science.

[16] Nick Brown et al., "Is RISC-V ready for HPC prime-time: Evaluating the 64-core Sophon SG2042 RISC-V CPU," in *SC '23 Workshops*, 2023.

[17] Pablo Vizcaíno, Filippo Mantovani, Thomas Fecht, and Judit Gimenez, "Short reasons for long vectors in HPC CPUs: A study based on RISC-V," in *SC '23 Workshops*, 2023.

[18] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao, "ReAct: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[19] Banana Pi Team, "Banana pi BPI-F3 RISC-V development board," `https://wiki.banana-pi.org/Banana_Pi_BPI-F3`, 2024, Accessed: 2026-03-09.

[20] SpacemiT, "SpacemiT K1 RISC-V processor," `https://www.spacemit.com/products/keystone/k1`, 2024, Accessed: 2026-03-09.

[21] MiniMax, "MiniMax M2.7: Model self-improvement, driving productivity innovation through technological breakthroughs," 2026, LLM with 200k-token context and enhanced agent-building capabilities.

[22] Wan-Ping Lee, Michael P. Stromberg, Alistair Ward, Chip Stewart, Erik P. Garrison, and Gabor T. Marth, "MOSAIK: A hash-based algorithm for accurate next-generation sequencing short-read mapping," *PLOS ONE*, vol. 9, no. 3, pp. e90581, 2014.

[23] Jiantao Wu, Wan-Ping Lee, Alistair Ward, Jerilyn A. Walker, Miriam K. Konkel, Mark A. Batzer, and Gabor T. Marth, "Tangram: a comprehensive toolbox for mobile element insertion detection," *BMC Genomics*, vol. 15, no. 1, pp. 795, 2014.

[24] Tomasz Olas, Lukasz Szustak, Roman Wyrzykowski, Mateusz Olas, and Marco Lapegna, "Advances in adapting memory-bound cfd computations to risc-v multicore architecture," in *International Conference on Computational Science*. Springer, 2025, pp. 151–166.

Arquitecturas heterogéneas y modelos de programación (4/4)

Transformers Eficientes para Imágenes Hiperespectrales: Optimización de Memoria mediante SpectralStem y Kernels Triton

Benjamín Sánchez-Calza[1], Mercedes E. Paoletti[1] y Juan M. Haut[1]

Resumen— **Los modelos basados en Transformers presentan un cuello de botella crítico en la jerarquía de memoria de las unidades de procesamiento gráfico (GPU) debido a la complejidad cuadrática de la auto-atención y al volumen masivo de datos en tensores de alta dimensionalidad espectral. Este trabajo presenta una estrategia de optimización doble para acelerar la inferencia en estos escenarios. Primero, se introduce SpectralStem, un módulo de tokenización 3D diferenciable que reduce la redundancia espectral mediante un mecanismo de atención matricial, optimizando el número de tokens antes de la etapa global. Segundo, se implementa una versión personalizada de FlashAttention utilizando el lenguaje Triton, la cual optimiza el tráfico a la memoria de alto ancho de banda (HBM) mediante un esquema de tiling adaptativo en la memoria compartida (SRAM). Los resultados experimentales en la arquitectura NVIDIA Ada Lovelace demuestran que el kernel propuesto reduce el tráfico HBM en un 98.5 % y logra una aceleración de 1.62x frente a PyTorch estándar, manteniendo una precisión global del 98.76 % sin degradación aritmética.**

Palabras clave— **Arquitectura de computadores, paralelismo, Triton, jerarquía de memoria, Transformers, GPU.**

I. Introducción

EL escalado de los modelos de aprendizaje profundo ha impulsado avances decisivos en visión artificial, desde las redes neuronales convolucionales clásicas [1], [2] hasta las modernas arquitecturas Transformer [3], [4]. La eficiencia del software se mantiene como factor determinante para maximizar la utilización del hardware [5], especialmente en dominios con alta dimensionalidad como las imágenes hiperespectrales (HSI). Las HSI capturan la reflectancia electromagnética en cientos de bandas espectrales contiguas, ofreciendo riqueza informativa superior a la fotografía convencional, pero presentando dos obstáculos computacionales críticos. Primero, la redundancia espectral extrema: un cubo de datos típico exhibe correlación elevada entre bandas adyacentes con número de condición espectral superior al centenar, donde métodos como PCA pueden perder información crítica al operar sin supervisión. Segundo, la jerarquía de memoria en GPU: el mecanismo de auto-atención original exhibe complejidad cuadrática en tokens y requiere materializar grandes matrices de afinidad en la memoria global (HBM), generando tráfico que frecuentemente supera la capacidad de cómputo según el modelo Roofline [6], obligando a usar lenguajes intermedios como Triton [7] para optimizar el mapeo a memoria local.

Este artículo propone una estrategia integral que aborda ambos cuellos de botella. Se introduce SpectralStem, un módulo de tokenización 3D diferenciable que reduce la dimensionalidad espectral mediante atención aprendida sin pérdida de expresividad, y se implementa FlashAttention [8] en Triton con tiling optimizado para SRAM, minimizando el tráfico HBM y maximizando ocupación de warps. Los resultados demuestran aceleración de 1.62x en inferencia frente a PyTorch estándar, elevando precisión al 98.76 %. La Sección II describe los desafíos de memoria en Transformers, la Sección III detalla la formulación matemática y la implementación del kernel, la Sección IV presenta la caracterización de rendimiento, y la Sección V resume contribuciones y hallazgos.

II. Arquitectura Base: PCSformer

A. Evolución de los Transformers en Visión por Computadora

La revolución de los Transformers en visión comenzó con el trabajo Attention Is All You Need [9], que demostró que la auto-atención pura podía superar a las redes recurrentes en tareas de procesamiento de secuencias. Su extensión a imágenes quedó consolidada con el Vision Transformer (ViT) [10], el primer modelo que demostró que un Transformer puro podía alcanzar un desempeño competitivo en clasificación de imágenes cuando se preentrenaba en conjuntos masivos. ViT divide la imagen en parches de 16×16 píxeles, linealiza cada parche como un token de dimensión d, y aplica auto-atención multi-cabeza con complejidad $\mathcal{O}(\mathcal{N}^{\in})$, donde $N = (H/16) \cdot (W/16)$.

Este cambio paradigmático impulsó diversas líneas de investigación, incluyendo el desarrollo de Transformers eficientes que reducen la complejidad cuadrática mediante atención local o aproximaciones dispersas [3], [11]. Asimismo, surgieron modelos híbridos CNN-Transformer que combinan la localidad convolucional con la atención global [12], [4], [13]. En paralelo, se han propuesto optimizaciones de hardware que explotan la jerarquía de memoria de la GPU, destacando algoritmos como FlashAttention [8], [14].

B. CNN vs. Transformers: Localidad vs. Dependencias Globales

Las redes neuronales convolucionales clásicas [15], [16], [2] inducen una estructura jerárquica mediante

[1]Departamento de Tecnología de Computadores y Comunicaciones, Universidad de Extremadura, correo: juanmariohaut@unex.es

campos receptivos locales que se expanden logarítmicamente con la profundidad. Aunque esto favorece la detección de características locales como bordes y texturas, penaliza el modelado de dependencias a largo alcance. Los Transformers, por el contrario, establecen conexiones directas entre todos los tokens desde la primera capa, permitiendo al modelo aprender qué información distante es relevante sin imposición estructural de localidad. Esta capacidad resulta crítica en el procesamiento de tensores con alta variabilidad interna, donde características situadas en planos distantes pueden ser conjuntamente discriminativas.

C. Imágenes Hiperespectrales: Desafíos Computacionales

Sea $\mathbf{X} \in \mathbb{R}^{\mathbb{C}\times\mathbb{H}\times\mathbb{W}}$ el cubo hiperespectral con C bandas espectrales. La estructura tridimensional natural del cubo presenta características que los modelos puramente convolucionales no explotan de forma óptima. En primer lugar, la correlación espectral elevada implica que las bandas adyacentes exhiben redundancia inherente que infla el coste computacional sin aportar información adicional significativa. Esto se refleja en un número de condición espectral elevado, indicando que el rango efectivo es mucho menor que la dimensionalidad total. En segundo lugar, existe una asimetría en el poder discriminativo de las bandas. Mientras que la varianza espectral captura energía no supervisada, el poder discriminativo para la clasificación suele estar sesgado hacia bandas de pequeña varianza global, las cuales son frecuentemente descartadas por métodos lineales como PCA. Finalmente, la escasez de datos etiquetados en conjuntos públicos impone restricciones severas, ya que la sobrecapacidad de los modelos Transformer puros puede provocar un sobreajuste severo sin mecanismos de regularización o inducción de localidad.

Desde una perspectiva de arquitectura de computadores, el procesamiento de estos cubos impone una presión extrema sobre la jerarquía de memoria. La disposición secuencial de cientos de canales espectrales por cada píxel espacial dificulta la coalescencia de los accesos a la memoria global. Cuando un warp de hilos intenta acceder a bandas contiguas de un mismo píxel, los patrones de acceso pueden divergir si el layout de los datos no está optimizado, resultando en múltiples transacciones de memoria para una sola instrucción de carga. Además, la persistencia de estos tensores en la caché L2 se ve comprometida por su volumen; para una resolución de 224×224 con 274 bandas, el tensor de entrada en punto flotante de media precisión (FP16) supera los 27MB, lo que puede provocar constantes desalojos en cachés de menor tamaño, degradando el throughput efectivo del sistema.

D. PCSformer: Arquitectura Híbrida CNN-Transformer

PCSformer [17] es una red híbrida de doble rama que procesa simultáneamente representaciones locales y globales, fusionándolas mediante módulos de acoplamiento de características. La tokenización inicial aplica sobre el cubo proyectado externamente mediante PCA:

$$\mathbf{Y} = \mathrm{PCA}(\mathbf{X}_{\mathrm{orig}}) \in \mathbb{R}^{\mathbb{B}\times\mathbb{C}_{\mathrm{in}}\times\mathbb{H}\times\mathbb{W}}, \tag{1}$$

donde el stem convolutivo aplica filtros de tamaño 7×7 con stride 4 para generar tokens espaciales. Esta salida trata todas las bandas de entrada como un único tensor 4D, sin capacidad de adaptación espectral diferenciable. Cada etapa implementa una actualización cruzada donde los mapas de características se proyectan entre las ramas mediante interpolación y pooling. Dentro de cada bloque Transformer, se aplica una técnica de submuestreo espacial de claves y valores para aliviar el coste cuadrático de la autoatención mediante el uso de convoluciones con stride elevado. La predicción final combina ambas ramas suavizando los resultados mediante un ensemble.

Sin embargo, esta arquitectura presenta limitaciones fundamentales relacionadas con la rigidez de su tokenización y el uso ineficiente de la memoria global. La implementación estándar de PyTorch materializa la matriz de scores íntegramente en la memoria HBM, lo que genera un tráfico del orden $\mathcal{O}(\mathcal{N}^{\in})$ bytes. En chips de alta capacidad como la arquitectura NVIDIA Ada Lovelace, las latencias de acceso a HBM dominan la ejecución cuando el número de tokens crece, violando la eficiencia predicha por el modelo Roofline.

El problema se agrava por la baja intensidad aritmética de la fase de atención en implementaciones ingenuas. La intensidad aritmética, definida como la ratio entre operaciones de punto flotante (FLOPs) y bytes transferidos desde la memoria principal, determina si un kernel está limitado por computo o por ancho de banda. En la auto-atención HSI, la necesidad de leer y escribir matrices de scores intermedias de gran tamaño reduce esta ratio drásticamente. Para maximizar el rendimiento, es imperativo rediseñar el flujo de datos para que el cómputo se realice íntegramente sobre la memoria SRAM de baja latencia, evitando por completo la materialización de la matriz de atención en la HBM.

III. Metodología

A. Fundamentación Teórica: Redundancia Espectral y Filtrado Adaptativo

El procesamiento de tensores hiperespectrales está condicionado por la redundancia espectral inherente a la captura física de la reflectancia. Sea $\mathbf{X} \in \mathbb{R}^{\mathbb{C}\times\mathbb{H}\times\mathbb{W}}$ el cubo de datos con C bandas contiguas. Debido a la naturaleza del muestreo espectral, las bandas adyacentes exhiben una correlación muy elevada, lo que resulta en una matriz de autocorrelación $\mathbf{R}_{\mathbf{C}\times\mathbf{C}}$ deficiente en rango. Esta redundancia se cuantifica mediante el número de condición espectral:

$$\kappa_{\mathrm{spec}} = \frac{\lambda_{\mathrm{máx}}}{\lambda_{\mathrm{mín}}} > 100, \tag{2}$$

el cual indica que una fracción mínima de componentes principales es suficiente para capturar la mayor

parte de la varianza del sistema. El módulo SpectralStem implementa un filtrado adaptativo en el dominio espectral que aprende ponderaciones dinámicas optimizadas mediante el gradiente de la función de pérdida.

B. *SpectralStem: Tokenización 3D con Atención Espectral*

Módulo SpectralStem sustituye el stem convolucional convencional por una etapa diferenciable que opera sobre el cubo completo. En la primera fase, se aplica un submuestreo espectral controlado mediante un parámetro de stride s, que reduce la dimensión efectiva:

$$C_{\text{eff}} = \lceil C/s \rceil \,. \tag{3}$$

En la segunda fase, cada banda genera tokens espaciales mediante una convolución compartida con stride 4:

$$\mathbf{T_b} = \text{Conv2D}_{\mathbf{s=4}}(\mathbf{X'_b}) \in \mathbb{R}^{\mathbb{P}\times}, \tag{4}$$

donde P es el número de posiciones espaciales. La fase crítica consiste en la implementación de una atención espectral basada en el producto escalar escalado sobre la dimensión de las bandas. Para cada posición espacial (i, j), se calculan las proyecciones lineales:

$$\mathbf{Q_{ij}} = \mathbf{x_{ij}}\mathbf{W^Q}, \quad \mathbf{K_{ij}} = \mathbf{x_{ij}}\mathbf{W^K}, \quad \mathbf{V_{ij}} = \mathbf{x_{ij}}\mathbf{W^V} \tag{5}$$

La matriz de afinidad resultante se obtiene mediante:

$$\mathbf{S_{ij}} = \text{softmax}\left(\frac{\mathbf{Q_{ij}}\mathbf{K_{ij}^\top}}{\sqrt{\mathbf{d_k}}}\right), \tag{6}$$

donde el factor $1/\sqrt{d_k}$ estabiliza los gradientes en el régimen de saturación del softmax. La salida se reduce a un único vector local mediante un promediado ponderado:

$$\mathbf{T_{sp}(i,j)} = \sum_{\mathbf{b=1}}^{\mathbf{C_{eff}}} \mathbf{S_{ij}^{(b)}} \cdot \mathbf{V_{ij}^{(b)}}, \tag{7}$$

integrando finalmente embeddings posicionales 3D factorizados para preservar la coherencia espectro-espacial:

$$\text{Pos}(b, i, j) = \text{Pos}_{\text{band}}(b) + \text{Pos}_{\text{row}}(i) + \text{Pos}_{\text{col}}(j). \tag{8}$$

C. *FlashAttention-Triton: Optimización de la Jerarquía de Memoria*

Para mitigar el cuello de botella generado por la materialización de la matriz de atención en la memoria global (HBM), se implementa una versión optimizada del algoritmo FlashAttention utilizando Triton. La estrategia se basa en un esquema de tiling asíncrono sobre la memoria compartida (SRAM) del multiprocesador. El kernel gestiona la actualización incremental del softmax mediante factores de escala dinámicos:

$$\mathbf{O_i} = \sum_{\mathbf{j=0}}^{\lceil \mathbf{N/B_N} \rceil} \text{attn}(\mathbf{Q_i}, \mathbf{K_j})\mathbf{V_j}, \tag{9}$$

donde $\mathbf{O_i}$ es la salida del bloque i. Desde un punto de vista de implementación, el kernel utiliza primitivas asíncronas para solapar las operaciones de carga desde HBM hacia SRAM con el cómputo de los productos matriciales. Al emplear un esquema de P-buffering (definido por el parámetro *stages* en la Tabla I), los Streaming Multiprocessors (SM) pueden procesar un bloque mientras el siguiente está siendo transferido. Esta técnica es vital para ocultar las latencias de acceso a la memoria global, que en arquitecturas modernas pueden superar los cientos de ciclos de reloj.

La gestión de la presión de registros es otro factor determinante en el diseño. El procesado de tensores de alta dimensionalidad espectral eleva el número de acumuladores necesarios por cada hilo. Al fijar el número de warps (generalmente entre 4 y 8), se equilibra el paralelismo a nivel de hilo con la disponibilidad de memoria compartida y registros por SM. Un valor de $B_M = 128$ permite una mayor reutilización de datos de los bloques de claves ($\mathbf{K}$) y valores ($\mathbf{V}$), pero incrementa el uso de registros; el autoajuste automático evalúa estas métricas para maximizar la ocupación efectiva (SM Occupancy) sin incurrir en desalojos de registros hacia la cache local.

Esta organización jerárquica de la memoria en la arquitectura Ada Lovelace se ilustra en la Figura 1, donde se visualiza el flujo de datos optimizado desde la HBM hacia la SRAM y los registros, permitiendo que el kernel Triton evite completamente la materialización de la matriz de scores $\mathbf{S}$ en la memoria global.

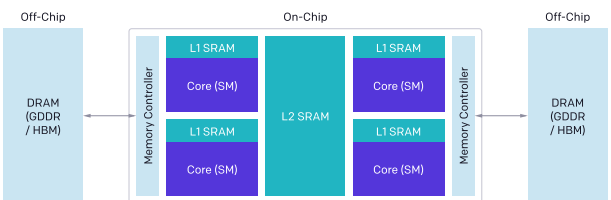


Fig. 1: Jerarquía de memoria en GPU NVIDIA Ada Lovelace. El kernel FlashAttention optimiza el tiling para mantener datos en SRAM (128 KB, latencia ∼100 ciclos) evitando materializaciones en HBM (47.4 GB, latencia ∼500 ciclos). La reducción del tráfico HBM es crítica para alcanzar eficiencia computacional en operaciones de atención.

Tabla I: Configuraciones de autoajuste en el kernel Triton (RTX 6000 Ada).

Cfg.	B_M	B_N	**Stages**	**Warps**
A	128	64	4	4
B	64	64	4	4
C	128	32	4	4
D	64	32	4	4

IV. Resultados Experimentales

A. *Configuración Experimental*

Conjunto de datos. Se utiliza el conjunto Hanchuan del benchmark WHU-Hi [18]: un cubo hiperespectral de 274 bandas espectrales en el rango

400–1000 nm, con una resolución espacial original de 1217 × 303 píxeles. Para el protocolo de evaluación estándar, se utilizan parches de 100 × 100 con 16 clases de cobertura terrestre. La partición estadística empleada consiste en un 10 % para entrenamiento (25 753 muestras), 10 % para validación y el 80 % restante (206 024 muestras) para test, siguiendo el protocolo establecido en trabajos previos [19].

Hiperparámetros. La dimensión de incrustación se fija en $d = 64$ con 8 cabezales de atención y una profundidad de 12 bloques. El submuestreo espectral emplea un stride $s = 6$, generando parches espaciales de 4×4. Se utiliza el optimizador AdamW con una tasa de aprendizaje inicial de 10^{-3} y un decaimiento de peso de 10^{-5}, aplicando un esquema de cosine annealing durante 200 épocas con un período de warmup inicial.

Entorno hardware y software. Todas las pruebas se han realizado en una NVIDIA RTX 6000 Ada Generation, basada en la arquitectura Ada Lovelace con 12 960 núcleos CUDA y 47.4 GB de memoria GDDR6X. El entorno de software incluye PyTorch 2.6, cuDNN y el compilador Triton 3.5, haciendo uso del mecanismo de autoajuste para optimizar los parámetros del kernel según el régimen de carga.

B. Precisión de Clasificación

La Tabla II resume las métricas de clasificación obtenidas sobre el conjunto de test. El modelo configurado con SpectralStem logra una mejora de 5.52 puntos porcentuales en precisión global (OA) frente al modelo base con tokenización fija. Este incremento demuestra que la selección adaptativa de bandas permite capturar características discriminativas que los métodos lineales no logran preservar.

Tabla II: Métricas de clasificación en Hanchuan (conjunto de test, 206k muestras).

Métrica	PCSformer + PCA	+ SpectralStem
OA (%)	93.24	**98.76**
AA (%)	88.71	**97.64**
Kappa	0.9217	**0.9855**
mIoU (%)	83.10	**95.94**

La mejora en el coeficiente Kappa (0.9855) sugiere que el modelo mantiene una alta consistencia en todas las categorías, incluyendo aquellas con menor representación estadística. La convergencia del modelo se monitoriza mediante la precisión en validación, como se muestra en la Figura 2, alcanzando la estabilidad antes de la época 150.

C. Rendimiento Computacional

La caracterización del rendimiento en inferencia se detalla en la Tabla III, comparando la implementación estándar frente al kernel Triton.

El uso de Triton permite reducir el tráfico a la memoria HBM en un 98.5 % para la resolución estándar, logrando una aceleración máxima de 1.63x. Esta eficiencia se debe a la optimización de los accesos a memoria y al uso intensivo de los registros y la SRAM,

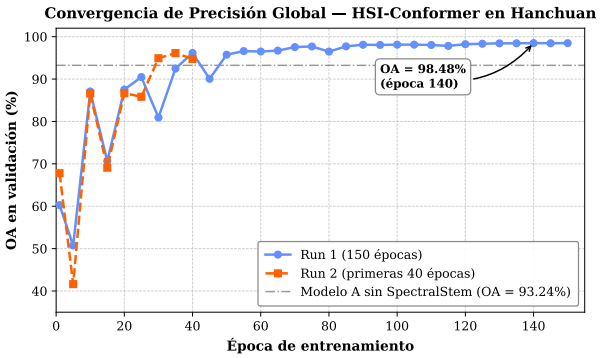


Fig. 2: Curva de convergencia de la precisión global en validación para el modelo optimizado.

Tabla III: Latencia (ms) y aceleración del paso de inferencia.

Res.	Batch	PyT (ms)	Tri (ms)	Speedup
224^2	16	34.0	**20.8**	**1.63x**
224^2	32	69.4	**42.7**	**1.62x**
448^2	4	34.4	**24.0**	**1.43x**
448^2	8	72.8	**49.2**	**1.48x**

evitando que la latencia quede limitada por los ciclos de acceso a la memoria global.

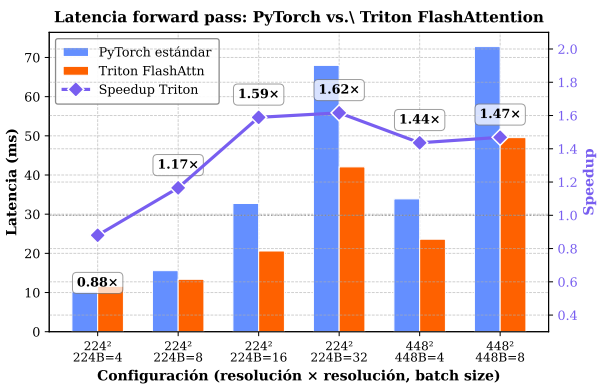


Fig. 3: Comparativa de aceleración (Speedup) del kernel Triton frente a PyTorch para diferentes resoluciones y tamaños de lote.

El análisis de throughput revela que la eficiencia del kernel Triton escala positivamente con la carga computacional. Para una resolución de 224^2 y un batch de 16, el sistema alcanza 16.36 TFLOPS efectivos. Sin embargo, al incrementar la resolución a 448^2, el throughput se eleva hasta los 56.77 TFLOPS. Este comportamiento evidencia que, a mayores densidades de tokens, la intensidad aritmética del kernel aumenta, permitiendo que el hardware de la RTX 6000 Ada opere más cerca de su techo teórico de cómputo. Mientras que PyTorch exhibe una saturación temprana del bus de memoria HBM, la implementación en Triton mantiene una alta tasa de ocupación de las unidades de proceso, transformando una operación tradicionalmente limitada por memoria en una limitada por cómputo. La Figura 3 ilustra cómo la ganancia de rendimiento es consistente a través de diferentes configuraciones, confirmando la robustez del esquema de tiling asíncrono propuesto.

V. Conclusiones

Este artículo ha presentado una arquitectura híbrida que integra dos optimizaciones sinérgicas para abordar los desafíos de cómputo en el procesamiento de tensores hiperespectrales a alta resolución. La introducción del módulo SpectralStem valida que la selección dinámica y aprendida de bandas espectrales supera a los métodos lineales tradicionales, elevando la precisión global al 98.76 % sobre el conjunto Hanchuan. Su diseño factorizado permite una gestión eficiente de la dimensionalidad sin requerir etapas de preprocesamiento externas, adaptándose fluidamente a la señal supervisada de la tarea.

Por su parte, la implementación del kernel FlashAttention en Triton confirma que la optimización a nivel de microarquitectura es fundamental para maximizar el rendimiento en GPUs modernas. Al gestionar manualmente la jerarquía de memoria y realizar un tiling asíncrono sobre la SRAM, se ha logrado reducir el tráfico a la memoria global en un 98.5 % y alcanzar aceleraciones de hasta 1.62x en inferencia. Ambas contribuciones son modulares y pueden integrarse en modelos existentes que presenten cuellos de botella similares por alta dimensionalidad o complejidad cuadrática de atención.

Como parte del trabajo futuro, se contempla la validación de este enfoque en conjuntos de datos adicionales y la extensión del kernel Triton para soportar convoluciones 3D espectrales de forma nativa. Asimismo, se explorarán técnicas de compresión espectral mediante destilación de conocimiento y la optimización de los kernels para su despliegue en entornos de computación embebida y plataformas edge con restricciones energéticas severas.

Agradecimientos

Los autores agradecen la financiación proporcionada por la Consejería de Economía, Ciencia y Agenda Digital de la Junta de Extremadura y por el Fondo Europeo de Desarrollo Regional (FEDER), en el marco de las ayudas a grupos de investigación de la Junta de Extremadura, expediente GR24035. Asimismo, este trabajo ha sido financiado parcialmente por el proyecto de I+D PID2023-149880NA-I00, financiado por MICIU/AEI/10.13039/501100011033; por la Junta de Extremadura, referencia IB24090; y por una Beca Leonardo 2024 para Investigación Científica y Creación Cultural de la Fundación BBVA. La Fundación BBVA no se hace responsable de las opiniones, declaraciones o contenidos incluidos en el proyecto, ni de los resultados derivados del mismo, que son responsabilidad exclusiva de los autores. Este trabajo también ha recibido financiación del Fondo Europeo de Desarrollo Regional (FEDER) de la Unión Europea, a través del programa Interreg España-Portugal (POCTEP) 2021-2027, bajo la ayuda 0206_RAT_EOS_PC_6_E.

Referencias

[1] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] Kaiming He, Xiangxiao Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[3] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo, "Swin Transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 10012–10022.

[4] Zhiliang Peng, Wei Huang, Shanzhi Gu, Lingxi Xie, Yaowei Wang, Jianbin Jiao, and Qixiang Ye, "Conformer: Local features coupling global representations for visual recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 367–376.

[5] Rajat Raina, Anand Madhavan, and Andrew Y. Ng, "Large-scale deep unsupervised learning using graphics processors," *International Conference on Machine Learning (ICML)*, 2009.

[6] Samuel Williams, Andrew Waterman, and David Patterson, "Roofline: An insightful visual performance model for floating-point programs and multicore architectures," Tech. Rep., Lawrence Berkeley National Laboratory, 2009, LBNL-128100.

[7] Philippe Tillet, H. T. Kung, and David Cox, "Triton: An intermediate language and compiler for tiled neural network computations," in *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)*. 2019, pp. 10–19, ACM.

[8] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré, "FlashAttention: Fast and memory-efficient exact attention with IO-awareness," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022, vol. 35, pp. 16344–16359.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention Is All You Need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, vol. 30.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations (ICLR)*, 2021.

[11] Jiarui Xu, Branislav Kisačanin, Yunchao Wei, and Boqing Guo, "Focal self-attention for local-Global interactions in vision transformers," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023, vol. 35.

[12] Zhuang Liu, Hanzi Mao, Cheng-Yang Fu, Kyle Morningstar, Tete Xiao, Boqing Guo, Christian Szegedy, and Bart van Merriënboer, "A ConvNet for the 2020s," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11976–11986, 2022.

[13] Hugo Touvron, Matthijs Douze, Maxime Oquab, Hervé Jégou, and Matthieu Cord, "Training data-efficient image transformers & distillation through attention," in *International Conference on Machine Learning (ICML)*, 2021, pp. 10347–10357.

[14] Tri Dao, "FlashAttention-2: Faster attention with better parallelism and work partitioning," in *International Conference on Learning Representations (ICLR)*, 2024, arXiv:2307.08691.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2012, vol. 25, pp. 1097–1105.

[16] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] Yong Wang, Xin Gao, Haokui Zhang, and Qiang Liu, "PCSFormer: Spectral-spatial transformer with point cloud support for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 61, pp. 1–16, 2023.

[18] Yanfei Zhong, Xin Hu, Chen Luo, Xinyu Wang, Ji Zhao, and Liangpei Zhang, "WHU-Hi: UAS-borne hyperspec-

tral with high spatial resolution (WHU-Hi) dataset and benchmark," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 167, pp. 256–271, 2020.

[19] Swalpa Kumar Roy, Subhrasankar Chatterjee, Shiv Ram Dubey, Bidyut Baran Chaudhuri, and Jón Atli Benediktsson, "Lightweight spectral–spatial squeeze-and-excitation attention embedded CNN model for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 8, pp. 5823–5839, 2020.

Optimizing AI Inference on Android Devices: A Comparative Study of GPU and NPU Acceleration

Iulius Gherasim and Carlos García Sánchez [1]

Abstract— **The pervasive integration of Artificial Intelligence (AI) into the Computing Continuum is driving a shift toward decentralized, real-time processing at the Edge. However, achieving autonomous and efficient Edge-Cloud operations requires overcoming significant challenges in performance sustainability and hardware heterogeneity. This paper investigates optimal execution strategies for AI models within the context of intelligent Edge service management, focusing on two critical tasks for infrastructure observability and smart services: object detection (YOLO family) and image classification (ResNet). These configurations evaluate various model quantization schemes and the utilization of on-device accelerators, specifically the GPU and NPU. Our core objective is to empirically determine the combination that achieves the best trade-off between minimal accuracy degradation and maximal inference speed-up.**

Key words— **Artificial Intelligence, Heterogeneous Computing, GPU, NPU, SoC, Performance, YOLO, ResNet, Android**

I. Introduction

The emergence of the *Computing Continuum*, with the expansion of Cloud boundaries toward Fog, Edge, and IoT paradigms, is transforming the deployment of digital solutions in critical sectors. In this post-digital transformation era, the agility and effectiveness of applications depend on dynamic service management capable of reconciling real-time constraints with geo-distributed and massively scalable infrastructures. As Artificial Intelligence (AI) systems transition from research prototypes to large-scale commercial applications, computational cost and energy consumption have become major concerns. The inference phase, where models operate continuously, dominates this consumption. Specifically, according to Google's estimates, AI-related tasks accounted for 10-15 % of its total energy use between 2019 and 2021, with inference consuming 60 % of that energy [1]. Similarly, Meta's internal reports indicate an energy capacity distribution of 10:20:70 for experimentation, training, and inference, respectively [2].

The escalating demands of AI workloads have driven the evolution of computing personal systems away from homogeneous processors toward heterogeneous architectures. This approach integrates multiple processing units, such as CPUs, GPUs, TPUs, and specialized accelerators, within a single system to optimize performance and energy efficiency by assigning tasks to the unit best suited for them. Furthermore, these advances are critical to the emerging Edge-to-Cloud paradigm [3–6]. In this model, data is processed locally at the "Edge" when possible, optimizing latency and bandwidth, while complex processing is offloaded to the "Cloud".

In the contemporary mobile computing landscape, AI performance is a core differentiator. The increasing complexity of AI models, pushed by the rapid popularization of AI [7], demands execution that meets strict real-time and accuracy requirements. Initial solutions involved offloading computation to the GPU, but recently approaches has been moved to the use of a specific *D*omain-*S*pecific-*A*rchitecture (DSA) as NPUs for this specific task. This DSA is engineered specifically to accelerate the tensor operations that dominate AI model layers, achieving better results in both compute speed and energy efficiency. Along with advancements on hardware, numerous frameworks emerged to allow developers to make use of artificial intelligence in their applications, such as PyTorch, ONNX and TensorFlow. However, when mobile devices became powerful enough, TensorFlow Lite, an extension of TensorFlow with select operations, enabled local execution on the device itself of certain lightweight models. As of 2024, TensorFlow Lite has been rebranded into LiteRT, short for *Lite Run Time*, and in the near future, it will be renewed again to LiteRT Next as Google keeps adding functionality to the framework, with the main focus being native support for accelerator offloading of model operations.

In the context of AI benchmarking, MLPerf [8] has emerged as the industry standard for evaluating Machine Learning (ML) inference across a spectrum of systems, ranging from embedded devices to large-scale data centers. Its primary objective is to provide a framework-agnostic and architecture-neutral evaluation of performance. This paper adopts a methodology aligned with MLPerf principles, tailored specifically for assessing inference performance on commercial Android-based SoCs. To ensure a representative evaluation, we selected two core models from the MLPerf suite that are fundamental to modern computer vision: ResNet [9] for image classification and the YOLO family [10] for object detection. By contrasting these models, we capture both the baseline requirements of classification and the complex, real-time demands of object detection, yielding robust data that reflect the energy-efficiency needs of current mobile applications.

The manuscript is structured as follows: Section II

[1]Dpto. de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, e-mail: {iuliusgh,garsanca}@ucm.es.

describes topics related to the experimentation environment used in this work; Section III describes the methodology for this study; Section IV presents the results obtained, including performance comparisons; and Section VI summarises our contributions and proposes future research directions.

II. Environment

A. Hardware aspects

All the experimentation has been carried out in a stock Android tablet device, namely a Samsung Galaxy Tab S9. This device has been chosen as it comes with a System-of-Chip (SoC) which includes a NPU alongside the GPU and CPU, the devices that will be evaluated in this paper. The specific SoC is a Qualcomm Snapdragon SM8550-AC also referred to as Snapdragon 8 Gen 2 Mobile Platform. There are the three main computing units inside the SoC of interest for this paper, CPU, GPU and NPU, alongside other smaller dedicated units for networking or image processing.

The SoC CPU, named *Kryo*, is made up of eight cores arranged in a big.LITTLE configuration, with five *big* cores and three *little* cores. The name of each one and their maximum frequency is listed in table I. Of those cores, the three *little* ones are ARM Cortex-A510 cores, the first generation of high efficiency cores from the ARMv9 series of processors running at up to 2 GHz. The *big* cores on the other hand are not five of the same type of processor, instead being made up of three different kinds of ARM core, two Cortex-A710, another two Cortex-A715, and finally one Cortex-X3. Of these *big* cores, the X3 stands out as it is the most powerful one of the set reaching up to 3.36 GHz compared to the A7XX cores that run at up to 2.8 GHz.

Table I: Kryo CPU configuration

ID	Type	Frequency (GHz)
0	A510	2.016
1	A510	2.016
2	A510	2.016
3	A710	2.803
4	A710	2.803
5	A715	2.803
6	A715	2.803
7	X3	3.360

In addition to the CPU, the SoC incorporates a dedicated Qualcomm Adreno 740 GPU, supporting single and half floating-point precision execution (GPU16 and GPU32 execution modes), and a dedicated NPU branded by Qualcomm as the Hexagon Processor, capable of executing both integer and half-precision float computations. Publicly available specifications for the GPU architecture are scarce; the only detailed internal information comes from a third-party blogger's analysis[1]. From this analysis, it is deduced that the GPU comprises 12 Shader Processor units running at up to 719 MHz, a configuration also corroborated by internal Android system configuration files.

[1]Adreno 740 information Kurnal's blog

The SoC also includes a dedicated NPU featuring specialized units for tensor, scalar, and vector operations, all connected to a shared internal memory. This shared memory is crucial for reducing data transfer latency between these execution units, as the primary feature of the Hexagon is its ability to utilize Micro Tile Inferencing. This technique involves partitioning the layers of the AI model and dispatching each specific operation (tensor, scalar, or vector) to the corresponding dedicated execution unit for parallel and optimized execution.

Table II: Peak performance summary

Device	Peak Ops
CPU	19.6 GFLOPS
GPU	2.2 GFLOPS
NPU	26 TOPS

Table II summarizes the theoretical peak performance for the three available execution devices. It is important to note that while CPU and GPU figures are expressed in floating-point operations per second (GFLOPS), the NPU performance is reported in tera-operations per second (TOPS), reflecting its optimisation for integer-based inference.

B. Software environment

To fully exploit the available heterogeneous hardware resources, developers utilize the LiteRT framework to integrate AI functionalities into their applications. Since LiteRT natively executes *.tflite* models, a mandatory conversion step is required for models originally developed in frameworks like PyTorch. This conversion is performed externally on a host machine before deployment to the target device. The final *tflite* model is obtained via a multi-step pipeline: first, using PyTorch's export utilities to generate the intermediate *.onnx* format, and subsequently employing Katsuya Hyodo's *onnx2tf* package for the conversion to *.tflite* target. Although LiteRT offers a direct conversion path via its *ai_edge_torch* package, this feature is still in early development and lacks broad model support; therefore, the *ONNX*-intermediate pipeline was selected. Crucially, the *onnx2tf* toolchain allows for the explicit selection of different model quantization schemes, as summarized in Table III.

Table III: Quantization reference

Name	Datatype			
	I/O	Ops	Acts	Weight
float32	FP32	FP32	FP32	FP32
float16	FP32	FP32	FP32	FP16
int_q	FP32	INT8	INT8	INT8
int_q_int16_act	FP32	INT8	INT16	INT16
full_int_q	INT8	INT8	INT8	INT8
full_int_q_int16_act	INT16	INT8	INT16	INT16
dynamic_range_q	FP32	Mixed	FP32	Mixed

Post-training quantization (PTQ) can be implemented through two primary approaches: static and dynamic quantization. This study primarily employs static quantization, a method where both model weights and activation distributions are converted

to the target data type (e.g., INT8) offline. This results in a model with fixed parameters that are directly used in layer operations, maximizing execution speed. In contrast, Dynamic Quantization (hereafter DYN) utilizes 'dynamic-range operators'[2] that quantize activations to 8-bit precision at runtime, immediately before computation. While weights are stored in 8-bit format, they are often cast back to floating-point precision during the operation. Although this scheme incurs a slight runtime overhead compared to the static approach, it remains faster than full FP32 execution and typically offers superior accuracy preservation.

Even though many integer widths can be used for operations, weights and activations when saving the model and are actually supported by the *Tensor* class in LiteRT, there are no optimized implementations of operations for 16 bit integers[3].

In order to facilitate the experimentation a small Android application has been developed that runs the models and stores inference results and speeds. Starting from LiteRT version 2.0 is planned to change to LiteRT Next, which will make previous applications incompatible due to changes in the API. As of version 2.0.2 of LiteRT, the pipeline to run models in the NPU does not work properly[4], so for this paper version 1.4 was also needed to be able to use this accelerator.

To successfully evaluate performance across the entire heterogeneous architecture, a mixed framework approach was needed due to software compatibility constraints. Specifically, CPU and GPU workloads were executed using the LiteRT Next API, which offered superior control over these units, while NPU workloads required the use of the more stable standard LiteRT framework. This methodological split allows for the comprehensive benchmarking of each accelerator's capabilities, as detailed in Table IV.

Table IV: LiteRT version requirements

Device	LiteRT version
CPU GPU	2.0.2
NPU	1.4.0

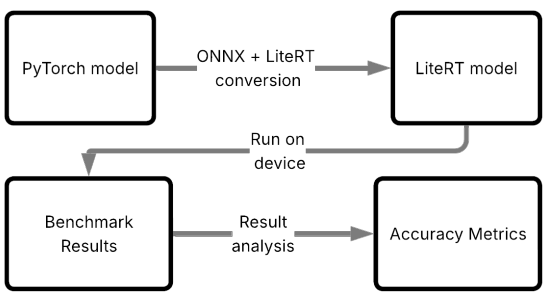
III. Methodology

The experimental workflow follows a multi-stage model preparation pipeline, as illustrated in Figure 1. Initially, models of various sizes were sourced from their respective official repositories: the PyTorch *torchvision* module[5] for ResNet, and the Ultralytics platform[6] for the YOLO family. The conversion process consists of two main steps: first, PyTorch models are exported to the intermediate ONNX format; second, these are converted to the final TFLite (*.tflite*) format using the *onnx2tf* tool. During this second stage, the models are also quantized. While *onnx2tf* outputs floating-point versions (FP32 and FP16) by default, we configured the tool with specific parameters[7] to generate the dynamic and integer quantized variants described in Table III.

The benchmarking phase is performed by running inferences against the standard validation datasets corresponding to each model family. Specifically, the ImageNet 1K validation dataset[8] is used for ResNet family,while COCO[9] validation dataset is employed for YOLO family. The primary performance data collected includes the average inference time (latency) across the entire dataset and the corresponding accuracy metric. This comprehensive evaluation was repeated for every model size, quantization variant, and target device.

Finally, the last phase involved the quantitative assessment of the obtained classification and detection results to determine model accuracy. For the YOLO models, the most common accuracy metric is mean average precision (mAP) directly obtained from the Python package *pycocotools*. For ResNet, the top-1 percentage metric is also calculated by computing how many of the total predictions recorded match the ground truth for the image class. Figure 1 illustrates the process at a glance.

Fig. 1: Experimental Workflow and Model Preparation Pipeline



IV. Results

Before presenting the results obtained for the ResNet image classification and YOLO object detection models, some observations are necessary. Firstly, the Table V summarizes the acronyms used throughout the following subsections to refer to the precision, device, and quantization variants used in the experiments. Secondly, the FP32-bit model executed on the CPU on a single thread is considered as a baseline which will serve as reference for speed-up and accuracy loss comparisons. And finally, the accuracy metric depends on the model evaluated: Top 1 accuracy for ResNet, and the *m*ean *A*verage *P*recision (mAP) for YOLO.

[2]Dynamic operators in LiteRT
[3]LiteRT documentation states lack of INT16 kernels
[4]LiteRT Commit 34682d3
[5]ResNet models in *torchvision*
[6]Ultralytics website
[7]onnx2tf documentation
[8]ImageNet 1K validation dataset
[9]COCO validation dataset

Table V: Acronym correspondence

	Acronym	Meaning
Device	CPU-SC	Single threaded
	CPU-MC	Multithreaded
	GPU32	FP32 GPU execution
	GPU16	FP16 GPU execution
	NPU	NPU Execution
Quantization	FP32	Floating Point 32 bits
	FP16	Floating Point 16 bits
	INT8	Integer 8 bits
	FINT8	Full Integer 8 bits
	INT16	Integer 16 bits
	FINT16	Full Integer 16 bits
	DYN	Dynamic

A. *ResNet*

Table VI presents the inference time measured in milliseconds (ms) for the different quantization and device variants. As the data shows, the 16-bit integer variant exhibits very poor performance (high latency), a trend observed even across different model sizes. Therefore, these quantizations are excluded from the analysis for the remainder of this section. Results also show the clear dominance of the NPU as the fastest executing device, followed by the GPU and lastly the CPU.

Table VI: ResNet18 Average inference time (ms)

	CPU-SC	CPU-MC	GPU32	GPU16	NPU
FP32	79.06	26.34	13.68	5.54	1.20
FP16	79.31	26.52	15.71	5.61	1.19
INT8	23.26	5.63	21.77	22.68	0.61
INT16	791.35	798.39	831.95	857.21	882.30
FINT8	23.06	60.36	32.57	32.48	0.51
FINT16	883.34	883.18	883.50	883.16	880.98
DYN	32.31	7.43	40.42	37.14	37.14

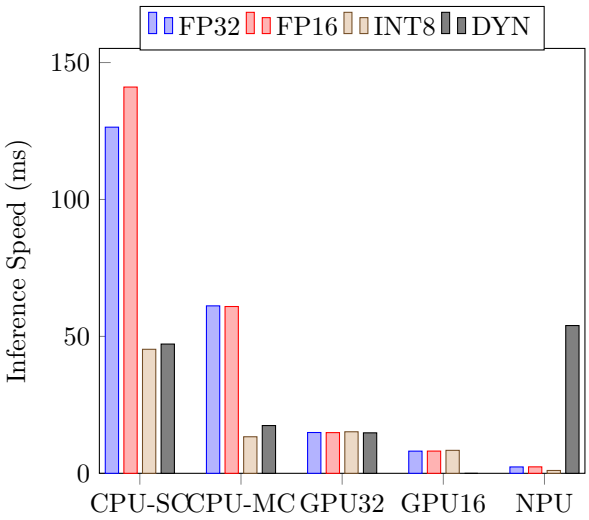
Table VII shows the Top-1 classification accuracy observed during ResNet18 model inference, varying the quantization scheme and the target device. Almost all variants achieve satisfactory accuracy, close to 69 %. The single notable exception is the FINT8 variant, which reaches a drastically low Top-1 accuracy of only 0.08 %. Therefore, this variant is also excluded from the remaining experimentation.

Table VII: ResNet18 Top1 accuracy

	CPU-SC	CPU-MC	GPU32	GPU16	NPU
FP32	68.81	68.81	68.81	68.8	68.82
FP16	68.82	68.82	68.82	68.79	68.81
INT8	65.87	65.87	65.87	65.87	65.96
INT16	65.95	65.95	65.95	65.95	65.95
FINT8	0.08	0.08	0.09	0.09	0.08
FINT16	65.95	65.95	65.95	65.95	65.95
DYN	68.71	68.71	68.71	68.71	68.71

Consistent with previous findings, Figure 2 illustrates the inference latency for the ResNet50 model. The NPU delivers superior performance across all variants, achieving a 120× speedup compared to the CPU-SC baseline. An exception is observed with Dynamic Quantization (DYN), which performs significantly slower on the NPU. This inefficiency arises because the NPU lacks native support for the runtime data-type casting required by DYN. Consequently, these operations are offloaded to the CPU, incurring substantial data transfer overhead that negates the accelerator's performance gains. This result confirms that the NPU is optimized for static, pre-calculated data rather than dynamic, runtime conversions.

Fig. 2: ResNet50 inference speeds across devices & quantizations



Focusing on CPU performance (see Table VIII), the analysis revealed that the multithreaded speed-up (CPU-MC, utilizing the 8 available cores) was significantly lower than expected. The highest acceleration was achieved with INT8 quantization, reaching a factor of 3,4×, which suggests that the asymmetric multicore architecture significantly limits scalability.

Table VIII: Multithreaded vs single threaded execution (ms) and speed-up in ResNet50

	CPU-SC	CPU-MC	Speed-up
FP32	126.4	61.2	2.1x
FP16	141.1	60.9	2.3x
INT8	45.3	13.4	3.4x
DYN	47.2	17.4	2.7x

On the GPU side, we observe that quantization hardly affects inference performance (see Table IX). Instead, the largest performance impact stems from the GPU's operating mode (FP32 vs. FP16), resulting in speed-ups approaching 2× if the GPU is configured to operate as GPU16.

Table IX: GPU FP32 vs FP16 execution (ms) in ResNet50

	GPU32	GPU16
FP32	14.9	8.1
FP16	14.86	8.12
INT8	15.17	8.39
DYN	14.78	8.13

Table X presents the accuracy variations resulting from model quantization. Switching from single-precision (FP32) to half-precision (FP16) floating point has a negligible impact on accuracy, with variations remaining within ±0,04 %. In contrast, quantizing the model to integer (INT8) or dynamic formats leads to a more notable decline, although the maximum accuracy loss remains below 0,5 %, demonstrating the robustness of the quantization process.

Table X: ResNet50 Top1 accuracy (%)

	CPU-SC	CPU-MC	GPU32	GPU16	NPU
FP32	79.09	79.09	79.06	79.09	79.1
FP16	79.1	79.1	79.1	79.06	79.1
INT8	78.68	78.73	78.68	78.68	78.77
DYN	78.9	78.91	78.9	-	78.94

The analysis of quantization impact reveals a clear correlation between model size and accuracy preservation. As shown in Table XI, the accuracy loss incurred by INT8 quantization decreases significantly as the model complexity increases (from 2.94 % for ResNet18 down to $\leq$ 0,5 % for larger variants). This trend confirms that larger models mitigate the information loss associated with reduced data width due to their deeper feature extraction capabilities. In contrast, Dynamic Quantization (DYN) consistently introduces negligible loss.

Table XI: Summary of Top-1 Accuracy Loss by Quantization vs. FP32 (%)

Quant.	ResNet18	ResNet34	ResNet50	ResNet101	ResNet152
INT8	2.94 %	0.50 %	0.41 %	0.01 %	0.20 %
DYN	0.10 %	0.04 %	0.19 %	0.00 %	0.07 %

Furthermore, Table XII demonstrates that converting the original single-precision floating-point models from PyTorch to the LiteRT framework introduces an inherent accuracy loss, independent of any subsequent quantization. This discrepancy, ranging between 0.83 % and 1.77 %, suggests that variations in operation implementations and numerical precision handling across frameworks prevent a bit-exact conversion. This baseline degradation is a critical factor that must be decoupled from the accuracy loss specifically attributed to quantization.

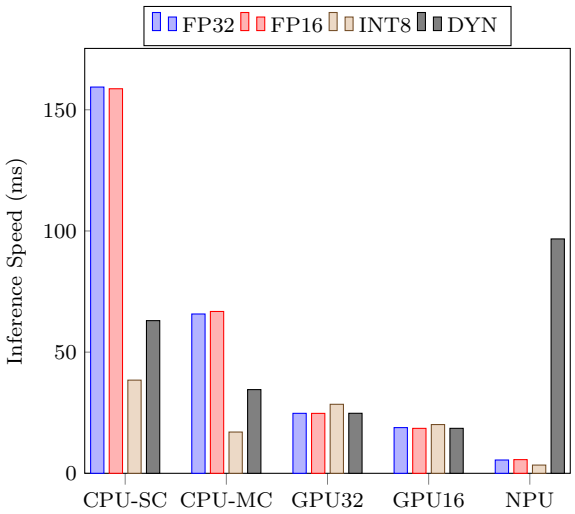
Table XII: Precision lost (Top-1) due to framework conversion(LiteRT vs. PyTorch) (%)

Model	ResNet18	ResNet34	ResNet50	ResNet101	ResNet152
Lost	0.95 %	0.83 %	1.77 %	1.16 %	0.98 %

B. YOLOv8

Results for the YOLOv8n model exhibit a performance pattern consistent with ResNet, where hardware accelerators (GPU and NPU) significantly reduce inference latency. As expected, YOLOv8n presents higher absolute latency compared to ResNet; this is attributed to the increased architectural complexity required for object detection versus the simpler requirements of image classification. Figure 3 illustrates these inference times across different devices and quantization levels. Notably, the NPU continues to provide the highest throughput, outperforming both the CPU and GPU, particularly when utilizing INT8 quantization.

Fig. 3: YOLOv8n inference speeds across devices & quantizations



Regarding accuracy, Table XIII reveals that integer-quantized variants experience more severe degradation compared to their ResNet counterparts. This significant drop in mAP suggests that object detection models, such as YOLOv8, are more sensitive to lower precision (INT8) due to the simultaneous demands of bounding box localization and object classification. In contrast, Dynamic Quantization (DYN) maintains high fidelity, with a negligible maximum loss of only 0.1 mAP across all models. Conversely, INT8 quantization leads to a substantial performance decline, with an average reduction of 6.5 mAP points, highlighting the trade-off between the NPU's inference speed and model precision.

Table XIII: YOLOv8 accuracy loss (mAP) compared to FLOAT32 precision

Quant.	YOLOv8n	YOLOv8s	YOLOv8m	YOLOv8l	YOLOv8x
INT8	6.5	6.2	6.2	7.5	6.1
DYN	0.1	0.0	0.0	0.1	0.1

C. YOLO11

With YOLO11 being an upgraded version of YOLOv8 it is expected to behave similarly but come with some improvements compared to its predecessor. The Figure 4 presents the newer YOLO11 inference times. It generally behaves like YOLOv8 but shows minor changes: CPU inference is slightly faster, while GPU speeds are nearly identical. Execution on the NPU is marginally slower for both float and INT8 models, and dynamic quantization fails to run entirely on the NPU. This performance shift is likely due to YOLO11's more complex architecture and new layers compared to its predecessor.

Fig. 4: YOLO11n inference speeds across devices & quantizations

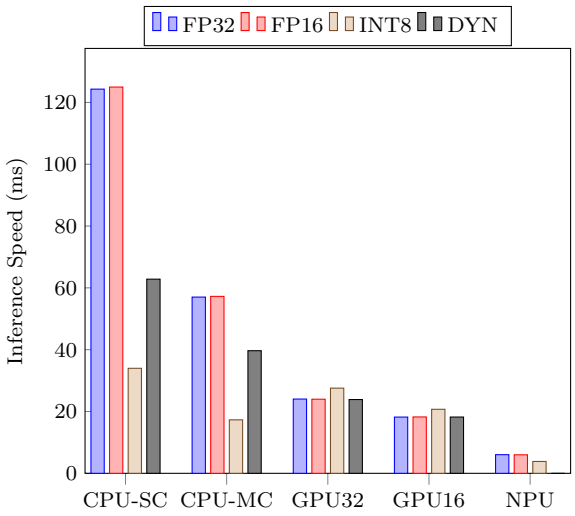
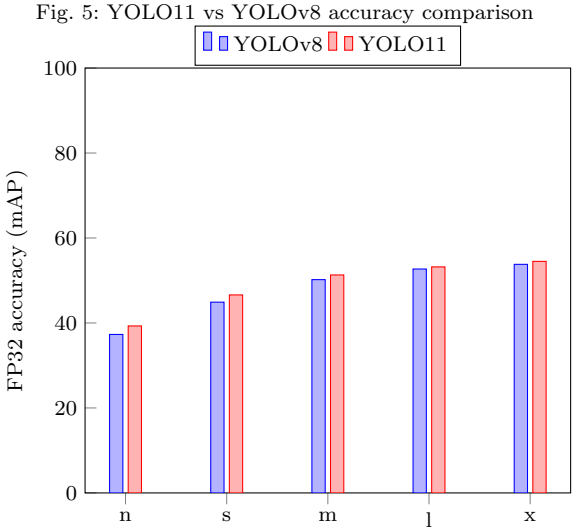


Fig. 5: YOLO11 vs YOLOv8 accuracy comparison



Accuracy results for YOLO11 follow the general trend observed in YOLOv8, as shown in Table XIV. However, integer quantization (INT8) induces a slightly more pronounced degradation in this architecture, with a mean accuracy loss of approximately 7.2 mAP across all variants. Notably, YOLO11l exhibits the most significant decline, reaching a loss of 8.7 mAP. Consistent with previous findings, Dynamic Quantization (DYN) maintains near-optimal fidelity, with a maximum loss of only 0.2 mAP. These results suggest that while YOLO11 offers architectural improvements, its internal weights and activations are more sensitive to the precision loss inherent in 8-bit integer mapping compared to the YOLOv8 family.

Table XIV: YOLO11 accuracy loss (mAP) compared to FLOAT32 precision

Quant.	YOLO11n	YOLO11s	YOLO11m	YOLO11l	YOLO11x
INT8	6.6	6.8	6.7	8.7	7.2
DYN	0.1	0.1	0.1	0.2	0.1

Finally, it is worth noting that YOLO11 exhibits a minor accuracy degradation when compared directly to its original PyTorch implementation. This loss, which ranges between 0.2 and 0.4 mAP points, is attributed to the framework conversion process into the LiteRT runtime environment. This suggests that the architectural innovations and novel layers introduced in YOLO11 pose a slightly greater challenge for bit-exact translation than previous iterations. Nevertheless, empirical evidence confirms that YOLO11 achieves superior overall FP32 accuracy across all model scales compared to YOLOv8, as illustrated in Figure 5. Consequently, YOLO11 stands as the more precise architecture for object detection tasks within the scope of this study.

V. Discussion

Based on the comprehensive evaluation of all models across the available hardware, the NPU emerges as the most efficient accelerator. It achieves substantial speed-ups even for the smallest architectures, such as 47× for YOLOv8n and 130× for ResNet18. Notably, the performance advantage of the NPU scales with model complexity, reaching a peak speed-up of 298× for the YOLOv8x variant. Table XV summarizes these performance gains relative to the CPU-SC baseline. These results highlight that the NPU's architectural advantages—specifically its ability to handle massive parallel integer arithmetic—are most effectively leveraged by deeper and wider neural networks.

Table XV: NPU execution speed-ups

		FP32	FP16	INT8	DYN
ResNet	18	65.9x	66.4x	129.6x	2.1x
	34	89.2x	88.2x	190x	3.7x
	50	54.3x	53.3x	121.5x	2.3x
	101	64.9x	63.6x	149.7x	3x
	152	70x	67.5x	164.1x	-
YOLOv8	n	29x	28.2x	46.8x	1.6x
	s	60.2x	60.5x	128.3x	2.9x
	m	77.6x	81.6x	209.4x	-
	l	99.5x	100.7x	277.6x	-
	x	98.2x	100x	298x	-
YOLO11	n	20.6x	20.8x	32.4x	-
	s	44.2x	41.5x	76.5x	-
	m	62.3x	64.9x	146.1x	-
	l	71.2x	72.5x	160.3x	-
	x	72.9x	72.6x	173.9x	-

Another key observation is the performance bottleneck associated with Dynamic Quantization (DYN) across all architectures. This method consistently yields lower speed-ups compared to other quantization variants and, in certain larger models, fails to execute on the NPU. This limitation is primarily due to the lack of native support for runtime casting, as previously discussed. Furthermore, the results demonstrate clear scalability: as model complexity increases, the NPU's acceleration becomes more pronounced. This trend is particularly evident in the YOLOv8 family, where the transition from nano to extra-large scales maximizes the hardware's compu-

tational throughput.

Regarding CPU performance, Table XVI illustrates that the speed-up achieved through multi-threading is predominantly driven by integer precision (INT8). While floating-point variants yield a modest 2× to 3× acceleration, INT8 consistently achieves a more significant 10× to 15× speed-up. Multi-threading efficiency, however, varies by architecture: ResNet models exhibit a diminishing speed-up in INT8 as model depth increases. In contrast, YOLO models demonstrate an opposite trend, leveraging multi-threading more effectively as they scale up, suggesting a more parallelizable workload in larger detection architectures.

Table XVI: Multithreaded execution speed-ups

		FP32	FP16	INT8	DYN
ResNet	18	3x	3x	14x	10.6x
	34	3.1x	3.1x	15.1x	12x
	50	2x	2x	9.5x	7.2x
	101	2.2x	2.1x	9.2x	7.2x
	152	2.2x	2.3x	9.4x	7.6x
YOLOv8	n	2.4x	2.4x	9.4x	4.6x
	s	2.3x	2.3x	11.1x	6.7x
	m	2.4x	2.4x	12.5x	7.5x
	l	2.3x	2.7x	12.8x	9.7x
	x	2.7x	2.1x	13.4x	10.1x
YOLO11	n	2.2x	2.2x	7.2x	3.1x
	s	2.1x	2.1x	10.4x	4.3x
	m	2.4x	2.3x	13.1x	6.2x
	l	2.6x	2.3x	13.4x	6.6x
	x	2.2x	2.1x	12.5x	7.3x

Finally, the analysis of GPU performance, presented in Table XVII, reveals that the highest speed-ups are achieved in GPU16 mode. In contrast to the CPU, variations in quantization precision (FP32, FP16, INT8, and DYN) have a marginal impact on the speed-up factor. This suggests that GPU performance is constrained by other factors, such as memory bandwidth or arithmetic complexity, rather than data-type precision alone. Nevertheless, FP16 often emerges as the optimal choice, providing an ideal balance between inference speed and accuracy.

Regarding scaling, YOLO models demonstrate superior efficiency on the GPU, with the largest variants reaching a peak acceleration of approximately 39× (YOLOv8x). In comparison, ResNet models maintain a more consistent speed-up across all sizes. These results indicate that larger models leverage half-precision execution more effectively than smaller ones. This behavior aligns with standard GPU performance characteristics, where massively parallel workloads—typical of deeper and wider architectures—are required to fully saturate the device's computational resources.

Table XVII: GPU16 execution speed-ups

		FP32	FP16	INT8	DYN
ResNet	18	17.5x	-	16.9x	-
	34	20x	20x	-	20x
	50	15.6x	15.6x	15.1x	15.6x
	101	15.8x	15.8x	15.5x	15.8x
	152	16.5x	16.5x	16.3x	16.3x
YOLOv8	n	8.5x	8.6x	8x	8.5x
	s	17.2x	17.1x	15.5x	17.1x
	m	14.6x	14.6x	13.1x	14.6x
	l	35x	34.8x	31.9x	34.9x
	x	38.3x	38.2x	34.8x	38.6x
YOLO11	n	6.8x	6.8x	6x	6.8x
	s	14x	14x	12.3x	14x
	m	23.2x	23.2x	19.3x	23.1x
	l	25.5x	25.2x	22x	25.2x
	x	30.1x	30.1x	26.5x	30.2x

Beyond individual metric analysis, selecting the optimal configuration is inherently challenging due to conflicting objectives, such as maximizing accuracy while minimizing inference latency and power consumption. To address this multi-criteria decision problem, a multi-objective optimization approach based on Pareto Front analysis is proposed. To maintain visual clarity, only FP16 and INT8 quantizations are displayed, as they represent the most competitive configurations for deployment.

Figures 6 and 7 illustrate the resulting Pareto Fronts, showcasing the non-dominated trade-offs between accuracy and latency. For the ResNet family (Figure 6), INT8 quantization emerges as the superior choice; the substantial reduction in latency effectively compensates for the marginal decline in Top-1 accuracy, placing these variants closer to the ideal bottom-right corner of the objective space. In contrast, for the YOLO models (Figure 7), the significant accuracy degradation associated with integer precision shifts the INT8 variants away from the optimal front. In the case of object detection, the trade-off is more polarized, requiring designers to carefully prioritize either extreme throughput (INT8) or high detection fidelity (FP16).

Fig. 6: Pareto Front ResNet

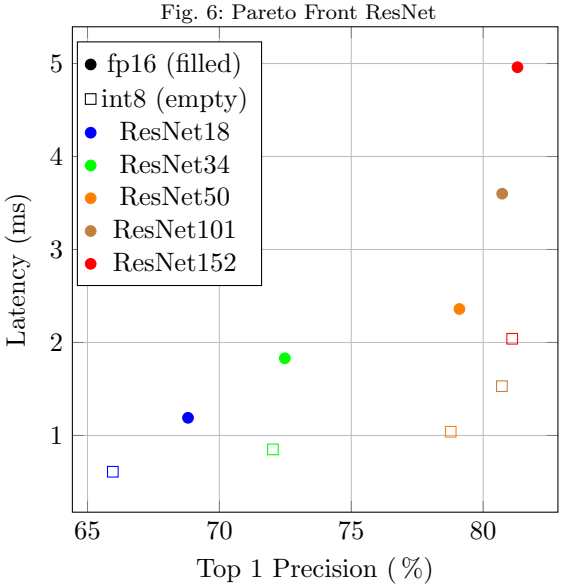
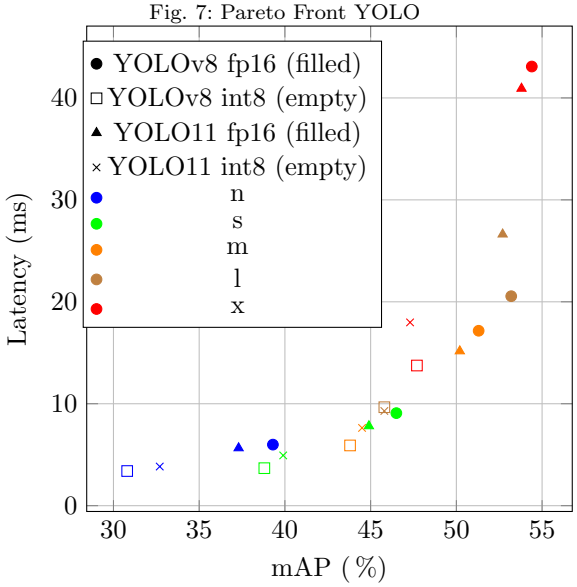


Fig. 7: Pareto Front YOLO



An additional takeaway is that YOLO11 tends to outperform YOLOv8 primarily in the smaller model scales (nano and small), where the substantial gain in accuracy justifies the marginal increase in inference latency. However, this relative advantage diminishes as the model size scales up. In the larger variants (large and extra-large), the accuracy gap between the two architectures narrows, while the latency overhead becomes more pronounced, resulting in a less favorable performance-to-cost ratio for the latest iteration.

VI. Conclusions

This paper evaluated the performance of ResNet and YOLO models for image classification and object detection, respectively, across three devices (CPU, GPU, NPU) and four precisions in an Android tablet. The key findings are:

- **NPU Dominance:** The NPU emerged as the superior execution device, achieving speed-ups of up to 298× for large-scale detection models (YOLOv8x) compared to the single-core CPU baseline. This confirms the NPU's essential role in enabling real-time, low-latency edge AI.
- **Optimal Quantization Trade-offs:** The choice between FP16 and INT8 is model-dependent:
  - For ResNet, INT8 is the optimal precision, as its performance gains on the NPU and multi-threaded CPU significantly outweigh its negligible accuracy loss.
  - Conversely, for GPU execution, quantization yields marginal benefits, making FP16 the preferred choice to preserve full accuracy without sacrificing speed.
- **Architecture Scaling:** YOLO11 exhibits a superior accuracy-to-latency ratio in smaller model scales (nano and small) compared to YOLOv8. However, this advantage diminishes in larger variants where increased architectural complexity results in higher latency overhead.
- **System Limitations:** Critical performance bottlenecks were identified.
  - Dynamic Quantization (DYN) proved inefficient on the NPU due to a lack of native support for runtime casting.
  - Furthermore, while CPU multi-threading provides notable gains (≈ 3,4× max speed-up) for INT8, scalability is constrained by the platform's asymmetric core architecture.

The results further reveal a clear correlation between model occupancy and accelerator efficiency. Smaller models (e.g., Nano and Small variants) appear to be memory-bound, failing to fully saturate the NPU and GPU computational units. In contrast, larger architectures effectively leverage the massively parallel nature of these accelerators, as evidenced by the exponential growth in speed-up factors. Additionally, the poor performance of Dynamic Quantization on the NPU underscores a critical hardware-software mismatch: while DYN offers flexibility, the lack of dedicated hardware for runtime data-type casting makes static INT8 quantization the only viable path for high-efficiency edge deployment.

As future work, we plan to integrate power consumption metrics into the Pareto analysis to finalize a tri-objective optimization (accuracy, latency, and energy efficiency). Additionally, we intend to investigate methods to mitigate the ≈ 1 % accuracy loss observed during framework conversion and extend this benchmark to the MLPerf suite, including tasks such as natural language processing and image segmentation.

Acknowledgements

This paper has been partially funded by the EU (FEDER), the Spanish MINECO under grants PID2021-126576NB-I00 and PID2024-158311NB-I00 funded by MCIN/AEI/10.13039/501100011033 and by European Union "ERDF A way of making Europe" and the NextGenerationEU/PRT.

Bibliography

[1] Zeyu Yang, Karel Adamek, and Wesley Armour, "Double-exponential increases in inference energy: The cost of the race for accuracy," 2024.

[2] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga Behram, James Huang, Charles Bai, Michael Gschwind, Anurag Gupta, Myle Ott, Anastasia Melnikov, Salvatore Candido, David Brooks, Geeta Chauhan, Benjamin Lee, Hsien-Hsin S. Lee, Bugra Akyildiz, Maximilian Balandat, Joe Spisak, Ravi Jain, Mike Rabbat, and Kim Hazelwood, "Sustainable ai: Environmental implications, challenges and opportunities," arXiv preprint arXiv:2111.00364, 2022.

[3] Sergio Moreschini, Elham Younesian, David Hästbacka, Michele Albano, Jiří Hošek, and Davide Taibi, "Edge to cloud tools: A multivocal literature review," *Journal of Systems and Software*, vol. 210, pp. 111942, 2024.

[4] Sukhpal Singh Gill, Huaming Wu, Panos Patros, Carlo Ottaviani, Priyansh Arora, Victor Casamayor Pujol, David Haunschild, Ajith Kumar Parlikad, Oktay Cetinkaya, Hanan Lutfiyya, et al., "Modern computing: Vision and challenges," *Telematics and Informatics Reports*, vol. 13, pp. 100116, 2024.

[5] Aitzaz Ahmed Murtaza, Amina Saher, Muhammad Hamza Zafar, Syed Kumayl Raza Moosavi, Muhammad Faisal Aftab, and Filippo Sanfilippo, "Paradigm shift for predictive maintenance and condition monitoring from industry

4.0 to industry 5.0: A systematic review, challenges and case study," *Results in Engineering*, vol. 24, pp. 102935, 2024.

[6] Wen Fei, Hiroyuki Ohno, and Srinivas Sampalli, "A systematic review of iot security: Research potential, challenges, and future directions," *ACM computing surveys*, vol. 56, no. 5, pp. 1–40, 2023.

[7] OpenAI, "Introducing ChatGPT," 2022.

[8] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, and George Yuan, "Mlperf inference benchmark," in *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*. 2020, ISCA '20, p. 446–459, IEEE Press.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778, Ganador del ILSVRC 2015 en clasificación, detección y segmentación.

[10] Leo Thomas Ramos and Angel D. Sappa, "A decade of you only look once (yolo) for object detection: A review," 2025.

[11] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean, "Carbon emissions and large neural network training," in *Proceedings of Machine Learning and Systems*, 2021, vol. 3, pp. 1–15.

[12] Google LLC, "Google environmental report 2024," Online, 2024, Accessed: 2025-01-07.

[13] Statista, "Google's electricity consumption worldwide from 2019 to 2023," Online, 2024, Accessed: 2025-01-07.

[14] Soumya Sudhakar, Vivienne Sze, and Sertac Karaman, "Data centers on wheels: Emissions from computing onboard autonomous vehicles," *IEEE Micro*, vol. 43, no. 1, pp. 37–43, January-February 2023.

[15] DataCenter Dynamics, "Meta's data center electricity usage reaches 15 TWh in 2023," Online, 2024, Accessed: 2025-01-07.

[16] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey, "Recalibrating global data center energy-use estimates," *Science*, vol. 367, no. 6481, pp. 984–986, 2020.

[17] Nicola Jones, "How to stop data centres from gobbling up the world's electricity," *Nature*, vol. 561, no. 7722, pp. 163–166, 2018.

[18] Google LLC, "Google environmental report 2023," Environmental report, Google LLC, 2023.

[19] Meta Platforms, Inc., "Meta sustainability report 2023," Sustainability report, Meta Platforms, Inc., 2023.

[20] cpu monkey, "Qualcomm Snapdragon 8 Gen 2 for Galaxy Benchmarks & Specs," .

[21] Passmark Software, "Snapdragon 8 gen2 mobile platform for galaxy," .

Creando contigüidad para la escalabilidad extrema en aplicaciones de tipo *stencil*

David Díez-Poza [1], Sergio Alonso Pascual [2], Yuri Torres [3], Arturo Gonzalez-Escribano[4]

Resumen— **La computación exaescala requiere desarrollar y ejecutar aplicaciones con enormes capacidades de escalabilidad. Una clase importante de aplicaciones científicas con un alto potencial de escalabilidad son las aplicaciones ISL (*Iterative Stencil Loop*). EPSILOD es una herramienta para simplificar el desarrollo y ejecución de aplicaciones ISL en entornos heterogéneos distribuidos. En este trabajo se proponen y evalúan técnicas para optimizar la comunicación de datos en aplicaciones ISL distribuidas en sistemas multi-GPU, multi-nodo. En concreto, se exploran políticas de partición multidimensionales y técnicas para optimizar las comunicaciones MPI, tanto *host-staging* como *GPU-aware MPI*, creando automáticamente continuidad de datos en memoria en función de la política de partición. Se presentan resultados experimentales con hasta 1024 GPUs distribuidas en 256 nodos, que muestran importantes mejoras en escalabilidad fuerte y débil en diferentes tipos de escenarios y aplicaciones ISL.**

Palabras clave— **Computación paralela, Aplicaciones stencil, Escalabilidad, Sistemas distribuidos, Sistemas heterogéneos**

I. Introducción

LOS sistemas pre-exaescala y exaescala son cada vez más accesibles. Para aprovechar al máximo este tipo de supercomputadoras es cada vez más necesario desarrollar soluciones que permitan obtener mayores niveles de escalabilidad. Estos sistemas presentan normalmente arquitecturas heterogéneas, que emplean distintos tipos de unidades de cómputo CPU y GPU para todo tipo de procesamiento de propósito general (GPGPU). Uno de los retos clásicos en este contexto de HPC (High Performance Computing) es el desarrollo herramientas, modelos de programación y modelos de ejecución muy eficientes en entornos altamente paralelos. Se busca mantener un alto grado de abstracción con el objetivo de reducir la complejidad inherente a la programación en estos entornos complejos. Algunas de estas soluciones son más genéricas, mientras que otras están adaptadas a un tipo de problema concreto.

Los ISLs o *Iterative Stencil Loops* son una clase representativa de aplicaciones científicas con un alto potencial de escalabilidad. En estas aplicaciones, el valor de los elementos del dominio se obtiene repitiendo el mismo cálculo sobre cada elemento en cada iteración. Este cálculo depende de los valores de un conjunto de elementos *vecinos* en la iteración anterior. El criterio de vecindad lo define cada aplicación concreta, creando un patrón de acceso fijo (*stencil*) que introduce dependencias con los datos vecinos. Esta característica implica que paralelizar el cómputo de *stencils* requiere comunicaciones de datos entre unidades de proceso que tienen asignados elementos vecinos en el dominio. Una solución muy eficiente es partir el dominio en bloques contiguos de grano grueso para maximizar la localidad y minimizar las comunicaciones. La localidad que presentan las aplicaciones tipo *stencil* hace que sean problemas especialmente adecuados para el uso GPUs para el cómputo local. Todo esto los convierte en problemas especialmente interesantes de cara a su uso en sistemas HPC de gran escala.

El grupo de investigación Trasgo propuso EPSILOD [1] como modelo para el desarrollo de aplicaciones ISL. Está basado en el concepto de esqueletos paralelos. Abstrae los detalles del reparto de trabajo y los patrones de cómputo y comunicación para simplificar la implementación de ISLs eficientes en sistemas heterogéneos distribuidos. EPSILOD se ha convertido en una herramienta muy apropiada para experimentar nuevas técnicas de implementación, despliegue y ejecución de aplicaciones de tipo ISL en sistemas heterogéneos distribuidos de todo tipo de escala. EPSILOD se apoya en dos bibliotecas de funciones subyacentes que conforman su capa de portabilidad entre dispositivos heterogéneos (Controller) y de distribución de carga y comunicaciones en sistemas distribuidos (Hitmap).

En este trabajo presentamos nuevos problemas y las correspondientes soluciones introducidas en EPSILOD para utilizar políticas de partición multidimensionales en EPSILOD para: (1) Reducir los volúmenes de comunicación; (2) Crear contigüidad de datos en memoria para reducir los costes de transferencia y comunicación; (3) Utilizar comunicaciones directas entre dispositivos mediante *GPU-aware MPI* de forma eficiente; y (4) Permitir un mayor grado de solapamiento de cómputo y comunicación; para finalmente mejorar los niveles de escalabilidad y rendimiento.

En concreto, presentamos las siguientes contribuciones:

- Un análisis de las ventajas e inconvenientes de las políticas de partición multidimensionales para diferentes tipos de dominios, incluyendo los problemas introducidos en la comunicación de datos discontinuos en memoria. Tanto para comunicaciones a través del host (*host-staging*) o directamente desde los dispositivos (*GPU-aware*

[1]Dpto. de Informática, Universidad de Valladolid, e-mail: david.diez.poza@uva.es
[2]Dpto. de Informática, Universidad de Valladolid, e-mail: sergio.alonso.pascual@uva.es
[3]Dpto. de Informática, Universidad de Valladolid, e-mail: yuri.torres@infor.uva.es
[4]Dpto. de Informática, Universidad de Valladolid, e-mail: arturo@infor.uva.es

MPI).

- Una solución basada en duplicación de *buffers* y tareas específicas de agrupación de elementos discontinuos (*marshall*/*unmarshall* o *pack*/*unpack*), adaptada para reducir el sobrecoste en la comunicación de bordes en aplicaciones *stencil* multidimensionales.
- Un estudio experimental en un supercomputador pre-exaescala para mostrar las nuevas mejoras de EPSILOD y su escalabilidad utilizando tres aplicaciones de ejemplo, dos con dominio de dos dimensiones y otra con dominio de tres dimensiones. Este estudio contiene comparaciones de rendimiento con la versión anterior de EPSILOD y con la versión actual de una herramienta del estado del arte que permite implementar fácilmente aplicaciones ISL distribuidas: Celerity (basada en SYCL). Este estudio incluye pruebas de escalabilidad fuerte y débil.

Los resultados experimentales muestran que las nuevas técnicas introducidas permiten aprovechar mejor las características de los sistemas heterogéneos distribuidos para mejorar la latencia de las comunicaciones, permitir un mayor solapamiento de cómputo y comunicaciones y mejorar el rendimiento y la escalabilidad de EPSILOD respecto a las versiones anteriores y otras herramientas del estado del arte.

La nueva versión de EPSILOD presenta en escalabilidad débil degradaciones de rendimiento de menos de 0,98 % para 216 nodos (864 GPUs) con respecto a la referencia en 1 nodo, en un escenario de tres dimensiones en el que el dominio crece de forma regular en todas las dimensiones. En escalabilidad fuerte, la misma aplicación muestra una aceleración de hasta 55,19x en 64 nodos y de 156,86x en 256 nodos con respecto al tiempo de referencia en 1 nodo. La comparación con Celerity, dadas las limitaciones de la política de partición de esta herramienta, muestra que EPSILOD mejora sus resultados de rendimiento consiguiendo mejor escalabilidad tanto débil como fuerte en todos los casos.

El código de la nueva versión de EPSILOD junto con las implementaciones de las aplicaciones utilizadas en el estudio experimental está a disposición pública[1].

El resto del artículo tiene la siguiente estructura. En la sección II se detalla trabajo relacionado incluyendo otros modelos similares. La sección III describe EPSILOD y las bibliotecas en las que se apoya. La sección IV describe las mejoras propuestas. Describimos el estudio experimental en la sección V. Por último, la sección VI trata las conclusiones y el trabajo futuro.

II. Estado del arte

Se pueden encontrar en la literatura diversos modelos, herramientas o soluciones de programación para desarrollar y ejecutar aplicaciones ISL en sistemas heterogéneos y distribuidos. Estas soluciones presentan abstracciones para ocultar los detalles del manejo de dispositivos heterogéneos y/o aceleradores y los detalles de la distribución y comunicación de datos.

Existen diversos modelos y herramientas de programación que permiten trabajar con códigos ISL en sistemas con GPUs [2], [3], [4], [5], [6], [7], [8]. Algunas están basadas en esqueletos o patrones paralelos. Estos permiten explotar características específicas del patrón de cómputo y comunicación del tipo de aplicación. SkelCL [9] o SkePU3 [10] incluyen un esqueleto *MapOverlap* que puede ser utilizado para implementar aplicaciones ISL. Ambos soportan multi-GPU en este esqueleto, pero no multi-nodo con aceleradores. Lo mismo sucede con el patrón *iterative stencil + reduce* en Fastflow [11]. Otras bibliotecas de esqueletos como Muesli [12] soportan multi-GPU y multi-nodo para aplicaciones ISL, con optimizaciones específicas para dominios complejos [13].

Todas estas soluciones no implementan mecanismos que exploten completamente el solapamiento de cómputo y comunicaciones, tanto entre dispositivos y nodo anfitrión como entre nodos. Estas soluciones escalan cuando el ratio entre cómputo y comunicación es lo suficientemente alto como para que las comunicaciones no sean muy significativas. Sin embargo, su coste afecta a la escalabilidad en escenarios comunes en los que el ratio de comunicación con respecto al cómputo crece con el número de dispositivos o nodos, siendo especialmente notable en aplicaciones ISL complejas con alto volumen de comunicación. Además, estos costes de comunicación se revelan ineludiblemente en estudios de escalabilidad fuerte cuando la cantidad de cómputo por nodo se reduce lo suficiente.

Una solución semejante a EPSILOD se propuso simultáneamente en 2022 para la herramienta Astaroth [14]. Utiliza técnicas semejantes a EPSILOD para ejecutar primero los bordes de cada subdominio y solapar cómputo y comunicación. Sin embargo, solo se exploran particiones de tantas dimensiones como el dominio. Por otra parte, el sistema de sincronización y solapamiento solo se ha probado hasta 64 dispositivos, con eficiencia de no más del 85 % en escalabilidad fuerte para dominios tridimensionales de tamaño limitado.

Otra solución para implementar aplicaciones ISL es utilizar una herramienta suficientemente abstracta aunque no especifica para esta clase de problemas. Por ejemplo, Celerity [15] utiliza como capa de portabilidad para dispositivos heterogéneos SYCL [16]. Celerity añade por encima un sistema abstracto para expresar distribuciones de datos y dependencias, que genera internamente las comunicaciones en MPI solapando parte de las transferencias de datos con el cómputo.

III. Antecedentes

Para facilitar el desarrollo de aplicaciones eficientes en entornos HPC, el grupo Trasgo ha desarrollado previamente varias bibliotecas de funciones es-

[1] https://gitlab.com/trasgo-group-valladolid/controllers/-/tree/26_JP_epsilod?ref_type=tags

critas en C11, compatibles con cualquier compilador moderno de C/C++. Las aplicaciones desarrolladas sobre estas bibliotecas se despliegan y adaptan en entornos distribuidos con configuraciones de hardware diversas tanto homogéneas como heterogéneas. Por tanto, proporcionan una capa de abstracción que simplifica el desarrollo y el despliegue en múltiples tipos de entornos: desde contextos de pequeña escala como sistemas empotrados con hardware heterogéneo, hasta clústeres exaescala con arquitecturas uniformes que incluyen aceleradores.

A. Hitmap

Hitmap [17] facilita las tareas relacionadas con la partición jerárquica de arrays multidimensionales u otras estructuras de datos con dominios tanto densos como dispersos. Se encarga del particionado de los datos en *tiles* (teselas), del mapeado de los mismos a los procesos y la comunicación y sincronización entre ellos. Implementa un sistema de plug-ins para las políticas de partición y mapeo. Las comunicaciones se expresan en función de los objetos resultantes. Por tanto, el código escrito con Hitmap es independiente de la forma en que se distribuye el trabajo. Permite también agrupar conjuntos de acciones de comunicación en patrones completos que se pueden reutilizar en diferentes puntos del programa. Hitmap está construida sobre MPI como sistema de comunicación interproceso. Se ha mostrado con diversas aplicaciones que Hitmap puede ser tan eficiente como la implementación manual en MPI, reduciendo el esfuerzo de programación.

B. Controller

Controller [18], [19] es una biblioteca de funciones. Implementa el concepto de controlador, una entidad abstracta que se encarga de gestionar, de forma transparente para el programador, la memoria y el lanzamiento de tareas en un dispositivo de cómputo. Un dispositivo puede ser desde un conjunto de núcleos de CPU hasta un acelerador hardware como una GPU o FPGA. Al utilizar unas abstracciones comunes para todos los dispositivos reduce el esfuerzo de programación. Controller implementa diferentes *backends* o sistemas de ejecución que se encargan de enlazar el código del usuario con las tecnologías específicas para el manejo del dispositivo correspondiente. Por ejemplo, OpenMP para conjuntos de núcleos o procesadores CPU; CUDA, HIP u OpenCL para tarjetas gráficas de NVIDIA y AMD; y OpenCL para FPGAs.

Controller ofrece varias abstracciones. Tiene un mecanismo para definir versiones específicas de un kernel para diferentes dispositivos con el mismo nombre y enlazarlos en un programa (*fat binary*). El programa escoge la implementación apropiada para cada dispositivo en el momento de la ejecución. También permite definir *kernels* genéricos independientes de detalles del modelo de programación. Se definen una única vez y se reutilizan en cualquier dispositivo. Los *kernels* y operaciones de cómputo se lanzan de forma asíncrona. El sistema analiza las dependencias de datos entre los argumentos de los *kernels* y se ocupa de forma transparente de mantener la consistencia de memoria entre los diferentes espacios de memoria y de mantener la consistencia secuencial con las sincronizaciones necesarias. Además, ofrece una abstracción para indexar de forma unificada un espacio abstracto de hilos de grano fino que se adapta a la arquitectura del dispositivo, junto con una interfaz para acceder a las estructuras de datos en función de los índices de los hilos que unifica los criterios de coalescencia y explotación de memorias caché tanto en CPUs como en aceleradores.

C. Aplicaciones ISL y EPSILOD

Una aplicación ISL es una aplicación iterativa. En cada iteración todas las celdas de una estructura de datos se actualizan con los valores de la iteración anterior en un conjunto de celdas *vecinas*. Qué celdas se consideran vecinas lo determina el patrón geométrico u operador de *stencil* que define cada aplicación. Una implementación paralela parte el espacio de la estructura de datos en bloques de datos contiguos que se asignan a cada proceso o dispositivo. A partir del operador de *stencil* se determina el tamaño y forma de los *bordes* de cada parte. Estos bordes representan las partes de la estructura que son celdas vecinas de alguna de las asignadas a otras partes. Por tanto, será necesario comunicar sus valores actualizados a otros dispositivos. La parte de datos asignada a cada dispositivo se amplía para tener unas zonas denominadas *halos*, donde se recibe la información de los bordes de otras partes que se han de recibir. Estos halos se utilizan en los cálculos de la siguiente iteración.

EPSILOD [1] es una herramienta para automatizar la creación y ejecución de aplicaciones ISL. Utiliza el modelo de programación de Controller (discutido en la Sección III-B) como su capa de portabilidad. Se implementa como un esqueleto paralelo. Un esqueleto paralelo [20] es un constructo de programación en forma de función de segundo orden que abstrae los detalles de un patrón de computación e interacción paralelos. El esqueleto paralelo definido en EPSILOD recibe como argumentos el tamaño del espacio de datos a partir y computar, una especificación del patrón u operador de *stencil* y la función que calcula o actualiza cada celda. EPSILOD utiliza mecanismos tanto de Controller como de Hitmap. Internamente, reserva un espacio para la frontera del dominio global. Este espacio está formado por un número de filas, columnas, de celdas de valor constante para implementar condiciones de contorno de Dirichlet. Esto permite eliminar estructuras condicionales y divergencias en el interior de los *kernels* que computan las celdas. Aplica a la parte interior del dominio una partición de datos por bloques y con el resultado de la misma calcula los tamaños y posiciones de los bordes y halos necesarios en cada dispositivo. En cada iteración lanza *kernels* para calcular de forma separada los valores de las celdas en

el borde del dominio de cada dispositivo. Cuando se detecta que los valores en un borde ya se han calculado realiza automáticamente de forma asíncrona las comunicaciones necesarias para copiar esos datos en los halos correspondientes de los dispositivos adyacentes. De esta forma se maximizan las oportunidades para solapar estas comunicaciones con el cómputo de la parte principal o central de los datos asignados al dispositivo.

EPSILOD también ofrece un Lenguaje Específico de Dominio (DSL) [21] basado en Python para simplificar su uso. Este DSL permite generar *kernels* genéricos que funcionan en cualquier tipo de dispositivo soportado, así como varias estructuras de datos y código que EPSILOD necesita. Permite generar el kernel automáticamente como una media ponderada de los pesos de los vecinos o especificar un kernel personalizado escrito en un dialecto simplificado de Python para aplicaciones que requieran cómputos más complejos.

Los resultados experimentales presentados en la literatura sobre EPSILOD [1], [21] indican que permite obtener una buena escalabilidad fuerte y débil tanto para sistemas homogéneos como heterogéneos, con mejor rendimiento que utilizando las versiones disponibles en aquel momento de otras herramientas del estado del arte.

IV. Propuesta e implementación

En esta sección se describen las técnicas de EPSILOD que se evalúan en este trabajo, para mejorar el rendimiento y escalabilidad mediante la optimización de comunicaciones y su solapamiento con el cómputo, así como la introducción de comunicaciones directas entre dispositivos a través de la tecnología *GPU-aware MPI*.

A. Introducción de tecnología GPU-aware MPI

La tecnología *GPU-aware MPI* [22], [23], introduce la posibilidad de utilizar direcciones de memoria de un dispositivo GPU en llamadas a funciones de comunicación de MPI. Internamente, MPI mueve los datos directamente entre GPUs del mismo nodo, o entre GPUs y el dispositivo de red (NIC) cuando implica GPUs en nodos diferentes. Esta tecnología intenta mejorar la eficiencia de las comunicaciones con respecto al tradicional modelo *host-staging*, donde el programa debe realizar explícitamente los movimientos de datos entre la memoria de las GPUs y el host, realizando las comunicaciones MPI siempre entre procesos en el host.

En EPSILOD, introducir este tipo de tecnología es sencillo gracias a las funcionalidades de las librerías subyacentes. Controller permite recuperar el puntero a la región de memoria en el dispositivo o GPU de un conjunto de datos. Hitmap permite usar este puntero en la estructura de metadatos asociada a un *array* o estructura de datos. Esta modificación se puede realizar al crear las estructuras de datos en la fase de inicialización. El resto del código de EPSILOD no necesita modificaciones, ya que las invocaciones a funcionalidades de comunicación de Hitmap, utilizan internamente MPI con los nuevos punteros a la memoria de los dispositivos. Esta modificación se puede activar o desactivar de forma opcional con una variable de entorno.

B. Descomposición del dominio

Dos de los factores fundamentales que influyen en la eficiencia de las comunicaciones son el volumen de datos y la dispersión de estos en memoria.

La elección de una adecuada descomposición del dominio permite reducir la cantidad de datos que cada proceso debe comunicar a sus vecinos, tanto envíos como recepciones [24]. En la figura 1 podemos ver como crece el volumen de los datos a comunicar en proporción a los datos computados en un escenario de escalabilidad fuerte. Esto nos da una idea de la capacidad de las distintas políticas de partición para solapar cómputo y comunicaciones al distribuir la carga entre un número creciente de dispositivos. Podemos observar que la proporción es menor cuando para el dominio más grande, aunque la tendencia es similar en ambos casos, lo que indica que una partición en todas las dimensiones del dominio sería la más indicada. Sin embargo, esto no tiene en cuenta la disposición de los datos en memoria.

En EPSILOD, cada proceso define una matriz con su dominio local. Sobre esta matriz se definen las regiones para la comunicación de datos (halos y bordes). Estas regiones son *vistas* definidas sobre la región de memoria del dominio local. En la figura 2 se muestra la disposición de los datos de las diferentes vistas cuando EPSILOD utiliza diferentes políticas de partición en un ejemplo en dos dimensiones. EPSILOD utiliza ordenamiento *row-major* para almacenar las estructuras de datos. Cuando se parte el dominio en una dimensión (1D), para *stencils* que accedan solo a vecinos adyacentes (*stencils* de 2º orden), los halos y los bordes son regiones de memoria contiguas. En el caso de órdenes mayores (acceso a vecinos de mayor distancia), los halos/bordes contienen varias filas. Eso introduce una discontinuidad debido a que los elementos de los extremos de las filas no se computan (condiciones de contorno) y en el modelo de comunicaciones exactas que implementa EPSILOD no se comunican.

En la partición 2D (dos dimensiones) los halos/-bordes horizontales presentan la misma situación. Sin embargo, los halos y bordes verticales siempre implican dispersión, aunque solo tengan una columna. El borde o halo tiene el ancho del radio del *stencil* (r). Los datos están colocados en memoria en grupos de r elementos contiguos, con un salto entre cada grupo igual al número de columnas de la matriz.

Esto se extrapola a dominios de más dimensiones. La discontinuidad es mayor para bordes/halos de dimensiones más altas, donde hay una jerarquía de saltos representables en MPI con una jerarquía de tipos derivados de clase *vector*.

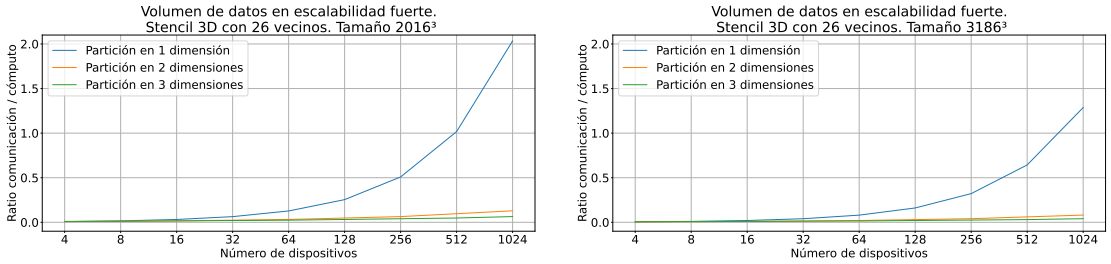


Fig. 1: Cálculo de la proporción, por proceso, entre el volumen de datos en comunicaciones y cómputo para distintos tipos de partición. Los datos describen un escenario de escalabilidad fuerte. A la izquierda sobre un dominio de 2016^3 elementos; a la derecha 3186^3.

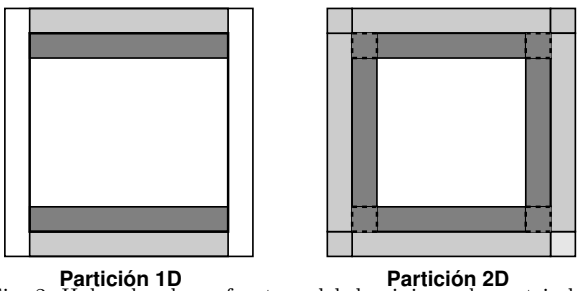


Fig. 2: Halos, bordes y frontera del dominio en la matriz local sobre un dominio de 2D. Partición en 1D (izq.) y en 2D (dcha.). En la partición en 2D, los halos conforman un conjunto disjunto de regiones de datos, mientras que los bordes presentan solapamiento.

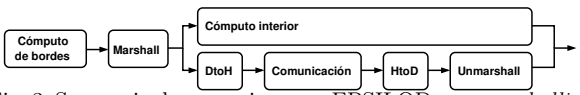


Fig. 3: Secuencia de operaciones en EPSILOD con *marshalling* y *unmarshalling* de bordes y halos en regiones contiguas de memoria.

C. Eliminando la discontinuidad

Como hemos descrito, reducir el volumen de datos a comunicar utilizando políticas de partición de más dimensiones, introduce regiones de datos con mayor y más compleja discontinuidad de memoria. Esto afecta negativamente a las transferencias de memoria que se realizan entre host y dispositivo involucrando dichas regiones. En el caso del modelo de comunicaciones *host-staging*, las transferencias entre GPU y host son más lentas cuando hay discontinuidades.

En el caso de utilizar comunicaciones *GPU-aware MPI* el efecto negativo es especialmente notable en las implementaciones actuales (como OpenMPI o Mpich). Se realiza una transferencia de datos diferente entre GPU y NIC por cada segmento de datos contiguo [25]. Esto se puede resolver introduciendo *kernels* específicos para copiar los datos discontinuos a *buffers* contiguos en la memoria de la GPU antes de invocar las transferencias MPI [25], [26].

Este tipo de técnica se puede aplicar en EPSILOD, tanto para solucionar el problema de las comunicaciones *GPU-aware MPI* como para mejorar la eficiencia de las transferencias entre GPU y host en el modelo *host-staging*.

La aplicación de esta técnica en EPSILOD se basa en una idea sencilla: generar *buffers* adicionales para halos y bordes con regiones de memoria contiguas. Para transferir los datos de la matriz original hacia y desde estas nuevas regiones de memoria se realiza un proceso de *marshalling* y *unmarshalling* incorporado al flujo habitual de EPSILOD que solapa cómputo y comunicación. El proceso para una iteración, que podemos ver en la figura 3, es el siguiente:

- Se computan los datos pertenecientes a los bordes.
- Se transfieren los datos de los bordes a su contrapartida contigua (*marshalling*).
- Transferencias de dispositivo a host si se sigue un modelo *host-staging*.
- Comunicaciones entre procesos.
- Transferencias de host a dispositivo si se sigue un modelo *host-staging*.
- Se transfieren los datos de los halos contiguos a su contrapartida en la matriz original (*unmarshalling*).

El proceso de *marshalling* y *unmarshalling* se realiza mediante lanzamientos de un kernel específico que se ejecuta en el dispositivo. En una primera aproximación estos se lanzan de forma asíncrona para que se solapen con el cómputo de la zona interior. Sin embargo, las tecnologías de los diferentes dispositivos como CUDA o HIP no garantizan concurrencia en la ejecución de *kernels* asíncronos. Observamos experimentalmente que, en ocasiones, el *marshalling* se realiza después del cómputo, impidiendo por completo el solapamiento de las comunicaciones con del cómputo principal. Por tanto, esta trasferencia inicial de los datos de los bordes a los *buffers* contiguos, ha de realizarse de forma síncrona antes del cómputo del interior. Esto introduce un cierto sobrecoste en el proceso, que puede ser significativo en escenarios de escalabilidad fuerte.

El uso de *buffers* supone un gasto de memoria adicional en el dispositivo. También en el host en el caso de usar el modelo *host-staging*. Este gasto adicional depende del método de partición como vimos en la figura 1. Los *stencils* de mayor orden también influyen proporcionalmente en este coste de memoria, ya que los halos y bordes son más grandes. Además, los bordes en EPSILOD pueden representar regiones solapadas, ya que varios procesos vecinos pueden requerir datos de una misma zona del dominio local. Esto sucede en patrones de *stencil* con *esquinas* (vecinos con desplazamientos en más de un eje). En la versión original de EPSILOD, los datos de los bordes estaban ubicados en una única región de memo-

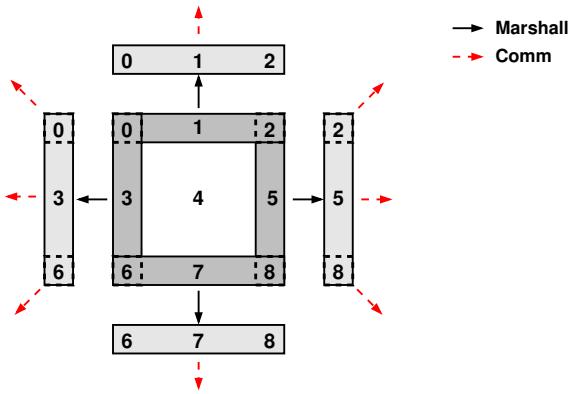


Fig. 4: Bordes originales en el interior rodeados por los *buffers* contiguos. Los *buffers* contiguos se han simplificado reduciendo su número y la duplicación de datos. Los números de borde repetidos representan regiones de datos duplicados.

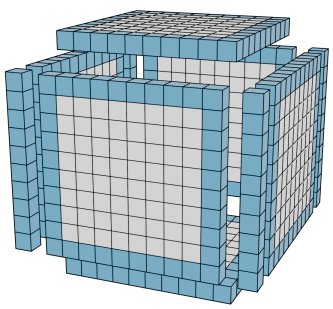


Fig. 5: *buffers* contiguos simplificados en un dominio 3D. Las regiones coloreadas representan datos duplicados.

ria física, en la matriz original. Con la introducción de *buffers* adicionales será necesario duplicar algunos de los datos, que al ser requeridos en diferentes procesos vecinos, deben copiarse en más de un *buffer* contiguo.

En la figura 4 podemos ver un ejemplo con los bordes originales de EPSILOD, rodeados de los *buffers* contiguos en los que se copian los datos. Por ejemplo, si partimos de un *stencil* de orden 2 (radio `r=1`), el borde 0 (dirigido a un proceso localizado arriba a la izquierda en la topología virtual de procesos) podría expresarse como una vista del *buffer* contiguo correspondiente al borde 1 o del borde 3. En ambos casos los datos mantendrían su contigüidad. Sin embargo, para *stencils* de mayor orden, el borde 0 contiene datos en varias filas. Para mantener la contigüidad de los datos, solo puede ser una vista del *buffer* 3.

En la figura 4 podemos ver el número mínimo de *buffers* necesarios para un dominio de 2D en el que cada proceso necesite comunicarse con sus 8 vecinos. Para comunicar 8 bordes necesitamos únicamente 4 *buffers* adicionales. Los bordes de las esquinas 0, 6, 2 y 8 se han expresado como vistas de los *buffers* generados para los bordes de las aristas 3 y 5. En el caso de un dominio de 3 dimensiones, para *stencils* que requieren datos de 27 vecinos, el número de *buffers* necesarios se puede reducir a 10 (Figura 5).

Por otra parte, los halos en EPSILOD representan regiones de datos disjuntos. Aunque no haya duplicación en los *buffers* para los halos, se ha seguido un método similar para reducir el número necesario.

De esta forma conseguimos reducir la duplicación de datos y el número de *kernels* de *marshalling* y *unmarshalling* necesarios para la copia de los datos a *buffers* contiguos. En el modelo de comunicaciones *host-staging*, también implica menos transferencias de memoria. En conjunto, se reduce el sobrecoste de la aplicación de esta técnica.

V. Estudio experimental

En esta sección presentamos un estudio experimental para evaluar el comportamiento de las técnicas propuestas en sistemas de supercomputación de gran escala en EPSILOD. El diseño de los experimentos contempla estudios de escalabilidad fuerte y débil, centrándose principalmente en los escenarios de escalabilidad fuerte al ser estos en los que las optimizaciones aplicadas en el sistema de comunicaciones son más relevantes porque la carga de cómputo por proceso llega a ser muy pequeña y cualquier ineficiencia puede derivar en perdida de solapamiento.

En este estudio contemplamos todas las combinaciones posibles de los siguientes parámetros experimentales:

- Política de partición: particiones 1D y 2D para dominios bidimensionales, particiones 1D, 2D y 3D para dominios tridimensionales.
- Tipo de comunicaciones: *GPU-aware MPI* o *host-staging*.
- Distribución de datos: discontinua (EPSILOD original) o contigua usando *kernels* para *marshalling* y *unmarshalling*.

También incluimos comparaciones de rendimiento con Celerity [15], un mecanismo abstracto para construir aplicaciones con comunicaciones genéricas en sistemas distribuidos con aceleradores, que utiliza SYCL [16] como capa de portabilidad.

A. Aplicaciones utilizadas

Como casos de estudio hemos elegido tres aplicaciones ISL que cubren una variedad de escenarios diferentes incluyendo operadores en dos y tres dimensiones, compactos y no compactos, que implican bordes de formas y tamaños diferentes. Las aplicaciones seleccionadas son:

2d4: Stencil compacto de 4 puntas de dos dimensiones. Elegido como caso base al ser el ejemplo clásico de aplicación *stencil* que se usa para calcular la ecuación de Poisson mediante el método de Jacobi.

2dnc8: Stencil no compacto de 8 puntas de dos dimensiones. Elegido al ser un ejemplo de *stencil* de mayor orden (radio del vecindario) y, por lo tanto, requiere comunicaciones más grandes que el caso anterior.

3d27: Stencil compacto con esquinas de 27 puntas de tres dimensiones. Elegido por ser un caso en tres dimensiones que además incluye esquinas en el patrón de *stencil*, lo que requiere más operaciones de comunicación y permite mayor fusión de *buffers* contiguos.

B. Plataforma experimental

La experimentación se ha realizado en Leonardo, un supercomputador pre-exaescala al que se ha accedido a través del programa EuroHPC JU. Leonardo está localizado en un centro de datos del Tecnopolo de Bologna en Italia y está gestionado por Cineca. Se ha empleado la partición Booster de la máquina, dedicada al cómputo con GPUs. Esta partición cuenta con 3456 nodos, cada uno con 4 GPUs NVIDIA Ampere A100 con 64 GB de memoria y una CPU Intel Xeon Platinum "Ice Lake" de 32 núcleos. El sistema de colas permite ejecutar hasta en 256 nodos simultáneamente en esta partición. Las pruebas en Leonardo se han realizado utilizando GCC 12.2, CUDA 12.2, Open MPI 4.1.6 y Celerity 0.6.0 y basado en oneAPI 2025.2.0.592.

C. Escenarios de escalabilidad

Hemos realizado pruebas de escalabilidad débil y de escalabilidad fuerte. Para estas pruebas utilizamos como referencia la ejecución en un nodo completo con un dominio cuadrado en 2D o cúbico en 3D repartido entre las cuatro GPUs del nodo.

En las pruebas de escalabilidad débil, al escalar el número de dispositivos el dominio crece en todas las dimensiones por igual. Es decir, un cuadrado o un cubo repartido entre todos los nodos, manteniendo un número de elementos igual al de la referencia, multiplicado por el número de nodos. Para las pruebas de escalabilidad fuerte el tamaño del dominio se mantiene constante y se reparte con la política seleccionada entre todas las GPUs de los nodos escogidos en cada experimento.

D. Tamaños del dominio

Hemos elegido como tamaños de partida los tamaños más grandes posibles considerando la memoria disponible de cada GPU y las limitaciones específicas del modelo de Celerity. En Celerity el tamaño de índices está limitado por el uso de enteros de 32 bits cuando se usa oneAPI como implementación de SYCL. Para poder realizar comparaciones con Celerity hemos realizado experimentos limitando el tamaño para no sobrepasar este límite. Al usar matrices más pequeñas el ratio de volumen de comunicación con respecto al cómputo es mayor, empeorando así la escalabilidad como se muestra en la figura 1. También se incluye un caso de estudio exclusivamente con versiones de EPSILOD usando un tamaño que ocupe toda la memoria disponible.

Las implementaciones de ejemplo en Celerity escogen automáticamente el tamaño de bloque para los *kernels* de GPU. Internamente, se escogen de forma que en cada dimensión, el tamaño del dominio sea divisible por el tamaño del bloque. Para algunos tamaños de dominio se pueden generar tamaños de bloque ineficientes. Se han escogido los tamaños de dominio de la experimentación para evitar dichos tamaños de bloque.

E. Resultados de experimentación

La figura 6 muestra los resultados de escalabilidad fuerte para la aplicación *2d4*. En esta aplicación, al ser un patrón de *stencil* de primer orden (compacto), solo aparecen discontinuidades en los bordes verticales cuando se aplica una partición 2D. Por lo tanto, observamos que la partición 1D tiene un comportamiento similar en las versiones con *buffers* contiguos (derecha) y sin ellos (izquierda). Mientras tanto, en la partición 2D observamos que las versiones con *buffers* contiguos mejoran el rendimiento tanto en comunicaciones *host-staging* como utilizando *GPU-aware MPI*, siendo la diferencia especialmente notable en el caso de comunicaciones *GPU-aware MPI*. En cuanto a Celerity, observamos que su tendencia de escalado es peor que EPSILOD desde el principio. Esto se debe a que carece de un mecanismo para solapar las comunicaciones de los bordes con el cómputo de la parte interna.

En este ejemplo, la partición 2D con *buffers* contiguos en EPSILOD deriva en un rendimiento y tendencias parecidas o incluso ligeramente peores a la partición 1D. Las mejoras conseguidas en los tiempos de comunicación no se compensan con los sobrecostes del lanzamiento y ejecución de los *kernels* de copia. A partir de 64 nodos, la escalabilidad se ve afectada por retrasos estocásticos introducidos por la tecnología de red.

En la figura 7 se muestran los resultados de escalabilidad fuerte de la aplicación *2dnc8*. Al tener esta aplicación un patrón de *stencil* de cuarto orden (radio 2), presenta un volumen de comunicación mayor que el caso de estudio anterior, así como discontinuidades en los bordes incluso con partición 1D. El mayor volumen de comunicación causa, en general, que los resultados estén ligeramente más separados de la línea que representa la escalabilidad ideal. Debido a las discontinuidades que presenta este patrón en la partición 1D, las versiones sin *buffers* contiguos ofrecen menor escalabilidad que las versiones con *buffers* contiguos, especialmente en el caso de comunicaciones *GPU-aware MPI*. En este caso la reducción de los tiempos de comunicación sí compensa el sobrecoste de los lanzamientos y ejecución de los *kernels* de copia. El comportamiento de Celerity es similar a la aplicación anterior debido a que su método de partición no presenta estas discontinuidades.

En la figura 8 se muestran los resultados de escalabilidad fuerte de la aplicación *3d27*. Al ser una aplicación en 3D, los bordes presentan discontinuidad en todos los tipos de particiones y aparecen discontinuidades jerárquicas en la última dimensión. Debido a esto, podemos observar en la parte izquierda de la figura que todas las versiones que usan comunicaciones de tipo *GPU-aware MPI* tienen mal rendimiento. En el caso de partición 3D es particularmente problemático, llegando a ser más de una orden de magnitud peor que los resultados mostrados en la gráfica. Para mejorar la claridad de la gráfica y evitar un gasto excesivo de tiempo de cómputo, hemos decidido omitir este resultado así como los puntos de

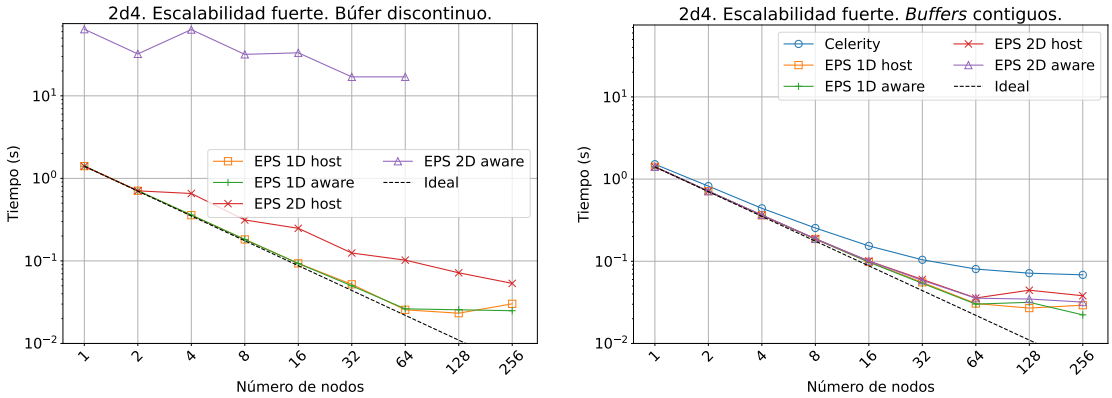


Fig. 6: Escalabilidad fuerte. 2d4. *buffers* no contiguos a la izquierda, *buffers* contiguos a la derecha.

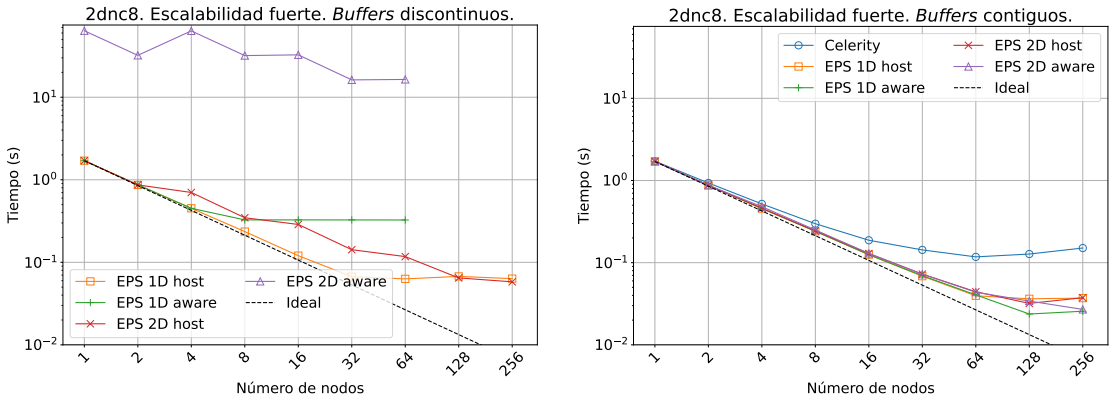


Fig. 7: Escalabilidad fuerte. 2dnc8. *buffers* no contiguos a la izquierda, *buffers* contiguos a la derecha.

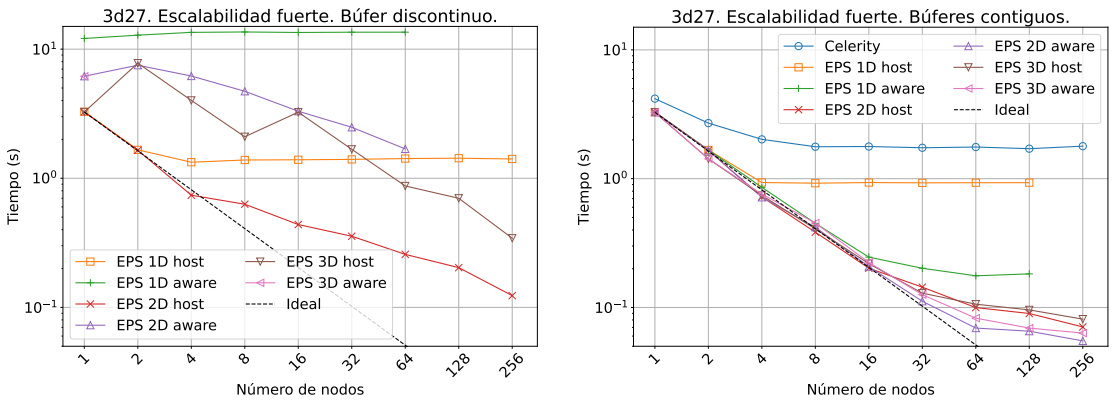


Fig. 8: Escalabilidad fuerte. 3d27. Tamaño limitado por enteros de 32 bits. *buffers* no contiguos a la izquierda, *buffers* contiguos a la derecha.

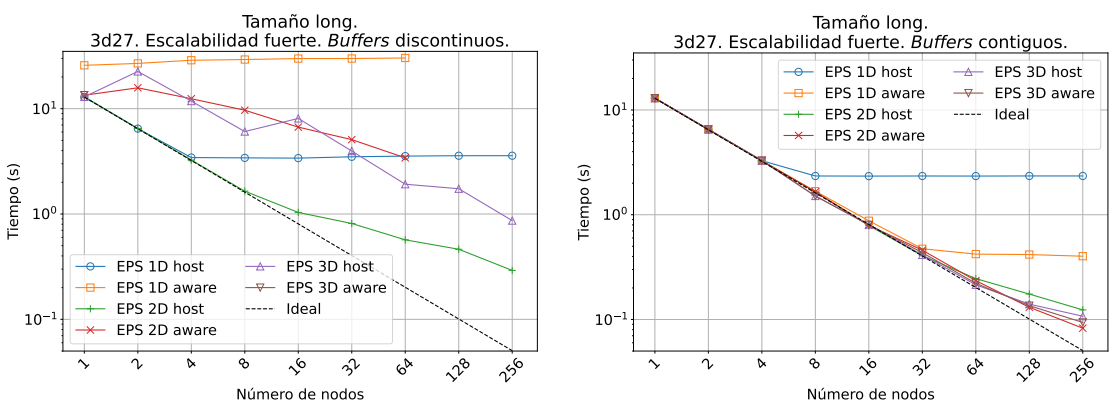


Fig. 9: Escalabilidad fuerte. 3d27. Tamaño superior a int. *buffers* no contiguos a la izquierda, *buffers* contiguos a la derecha.

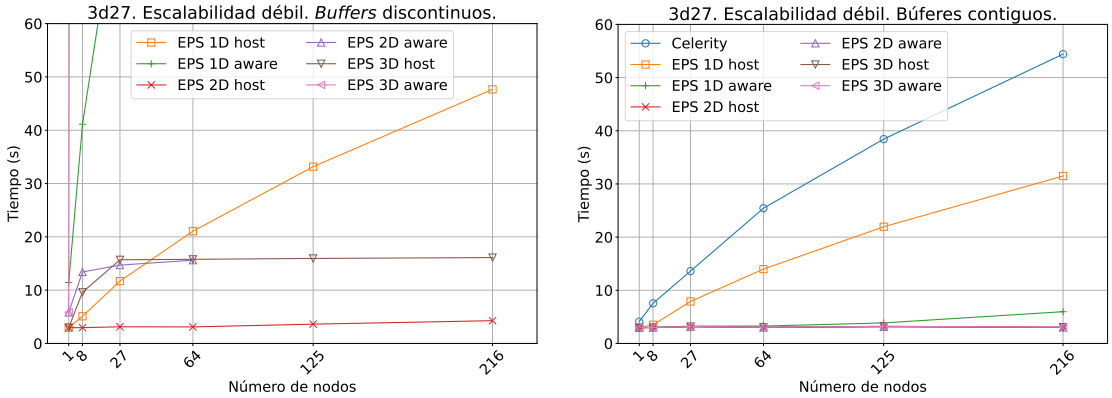


Fig. 10: Escalabilidad débil. 3d27. *buffers* no contiguos a la izquierda, *buffers* contiguos a la derecha.

128 y 256 nodos de las particiones 1D y 2D. En la parte derecha de la imagen podemos observar cómo el sistema de *buffers* contiguos soluciona este problema y las comunicaciones de tipo *GPU-aware MPI* se convierten en las que presentan mejores resultados.

También podemos observar que al ser una aplicación con dominio en 3D, las particiones 1D (incluyendo Celerity) no son adecuadas para altos niveles de escalabilidad, ya que el ratio de volumen de comunicaciones con respecto al cómputo crece mucho más rápido que con los otros tipos de particiones, como ya discutimos en la sección IV.

La figura 9 muestra los resultados de escalabilidad fuerte para la aplicación *3d27* con un tamaño que ocupa toda la memoria disponible de las GPUs en el caso de referencia con un nodo. En este escenario, al comenzar con un tamaño mayor, el ratio de volumen de comunicaciones con respecto al cómputo es menor, como se puede ver en la figura 1. Gracias a esto, conseguimos mantener una tendencia de escalabilidad más cercana al ideal hasta un número mayor de nodos.

La figura 10 muestra los resultados de escalabilidad débil para la aplicación *3d27*. Al igual que en los casos de escalabilidad fuerte de esta aplicación discutidos previamente, las comunicaciones de tipo *GPU-aware MPI* presentan un rendimiento claramente inferior en los casos no contiguos (especialmente con la partición 3D). Debido a esto, hemos omitido los puntos de 125 y 216 nodos para evitar un gasto excesivo de recursos de cómputo y hemos ignorado estos resultados a la hora de elegir la escala de la gráfica que se muestra para facilitar su interpretación. Podemos observar que las versiones con *buffers* contiguos mejoran claramente con respecto a sus correspondientes versiones no contiguas. Observamos también que las particiones con mejor escalabilidad son 2D y 3D. El motivo es que, como ya hemos discutido, con estas particiones el volumen de comunicaciones crece más lentamente que con la partición 1D. En este estudio de escalabilidad débil las comunicaciones permanecen completamente solapadas con *buffers* contiguos para particiones 2D y 3D. Esto causa que el tipo de comunicaciones (*GPU-aware MPI* y *host-staging*) no afecte al rendimiento.

VI. Conclusiones

En este trabajo se estudian los problemas que aparecen en aplicaciones ISL distribuidas sobre sistemas heterogéneos multi-GPU y multi-nodo, cuando se trabaja con particiones multidimensionales que pueden generar comunicaciones de datos no contiguos en memoria. Se presenta una propuesta para introducir en la herramienta EPSILOD nuevas técnicas para resolver dichos problemas. La propuesta combina el uso de particiones multidimensionales con la creación automática de contigüidad en memoria mediante operaciones de *marshalling*/*unmarshalling* en un conjunto reducido de *buffers* adicionales. La solución propuesta permite su aplicación de forma automática tanto en comunicaciones *host-staging* como utilizando *GPU-aware MPI*.

Los resultados experimentales muestran que la contigüidad de los datos comunicados es un factor decisivo cuando se utilizan particiones multidimensionales y, especialmente, cuando se recurre a comunicaciones *GPU-aware MPI*. En los casos bidimensionales y tridimensionales estudiados, la nueva versión de EPSILOD mejora de forma clara el comportamiento de las versiones previas con regiones no contiguas, y también supera a Celerity en escalabilidad fuerte y débil en los escenarios analizados. De forma destacada, en la aplicación 3d27 se alcanzan degradaciones inferiores al 0,98 % en escalabilidad débil hasta 216 nodos (864 GPUs), así como aceleraciones en escalabilidad fuerte de hasta 55,19x en 64 nodos y 156,86x en 256 nodos, respecto a la referencia de un nodo.

El estudio también pone de manifiesto que no existe una única configuración óptima para todos los escenarios. En patrones simples, como 2d4, el sobrecoste asociado a los kernels que crean la contigüidad puede degradar el incremento de eficiencia, mientras que en *stencils* de mayor orden o dominios tridimensionales las mejoras obtenidas con los *buffers* contiguos son mucho más acusadas. El sobrecoste se compensa con las mejoras de eficiencia escogiendo de forma adecuada la política de partición en función de las dimensiones del dominio y el tipo de *stencil*.

En conjunto, las soluciones propuestas permiten utilizar de forma eficiente las comunicaciones, reduciendo los efectos negativos de la discontinuidad de

memoria, mejorando el solapamiento entre cómputo y comunicación, consiguiendo mayores niveles de escalabilidad y, por tanto, aprovechando mejor las capacidades de los sistemas pre-exaescala actuales.

Como líneas de trabajo futuro, se plantea reducir los sobrecostes asociados a las operaciones de *marshalling/unmarshalling*, aumentando la eficiencia de los mecanismos de sincronización y estudiando métodos alternativos para generar la continuidad de datos en memoria. También se considerará extender el estudio con ejemplos de otros tipos de *stencil* de mayor orden o más complejos, así como estudiar la aplicación e impacto de estas técnicas en otras plataformas o con otras tecnologías.

Agradecimientos

El presente trabajo es parte de la actuación PID2022-142292NB-I00 (Proyecto de Generación de Conocimiento 2022), financiada por MICIU/AEI /10.13039/501100011033 y por FEDER, UE. Agradecemos a la iniciativa "EuroHPC Joint Undertaking" por concedernos mediante el proyecto EUHPC_D30_164 acceso a la supercomputadora LEONARDO, gestionada por CINECA y el "Leonardo consortium", así como por el soporte provisto por el proyecto EPICURE, ambos a través de una convocatoria de Desarrollo de EuroHPC.

Referencias

[1] Manuel de Castro, Inmaculada Santamaria-Valenzuela, Yuri Torres, Arturo Gonzalez-Escribano, and Diego R. Llanos, "Epsilod: efficient parallel skeleton for generic iterative stencil computations in distributed gpus," *The Journal of Supercomputing*, vol. 79, no. 9, pp. 9409–9442, 06 2023.

[2] Alyson D. Pereira, Luiz Ramos, and Luís F. W. Góes, "Pskel: A stencil programming framework for cpu-gpu systems," *Concurr. Comput. : Pract. Exper.*, vol. 27, no. 17, pp. 4938–4953, Dec. 2015.

[3] M. Viñas, B.B. Fraguela, D. Andrade, and R. Doallo, "Facilitating the development of stencil applications using the heterogeneous programming library," *Concurrency Computation*, vol. 29, no. 12, 2017.

[4] N. Maruyama, K. Sato, T. Nomura, and S. Matsuoka, "Physis: An implicitly parallel programming model for stencil computations on large-scale gpu-accelerated supercomputers," in *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2011, pp. 1–12.

[5] Thibaut Lutz, Christian Fensch, and Murray Cole, "Partans: An autotuning framework for stencil computation on multi-gpu systems," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, pp. 59:1–59:24, Jan. 2013.

[6] Takashi Shimokawabe, Takayuki Aoki, and Naoyuki Onodera, "A high-productivity framework for multi-gpu computation of mesh-based applications," in *Proceedings of the 1st International Workshop on High-Performance Stencil Computations*, Armin Gr.ºsslinger and Harald K.ºstler, Eds., Vienna, Austria, Jan 2014, p. 23–30.

[7] Luporini F, Louboutin M, Lange M, Kukreja N, Witte P, Hückelheim J, Yount C, Kelly PHJ, Herrmann FJ, and Gorman GJ, "Architecture and performance of devito, a system for automated stencil computation," *ACM Transactions on Mathematical Software*, vol. 4, no. 1, 2020.

[8] Hyojong Kim, Ramyad Hadidi, Lifeng Nai, Hyesoon Kim, Nuwan Jayasena, Yasuko E ckert, Onur Kayiran, and Gabriel Loh, "Coda: Enabling co-location of computation and data for multiple gpu systems," *ACM Trans. Archit. Code Optim.*, vol. 15, no. 3, pp. 32:1–32:23, Sept. 2018. tations on multi-gpu systems using the skelcl library," *Parallel Processing Letters*, vol. 24, 2014.

[9] Michel Steuwer, Michael Haidl, Stefan Breuer, and Sergei Gorlatch, "High-level programming of stencil compu-

[10] A. Ernstsson, J. Ahlqvist, S. Zouzoula, and C. Kessler, "SkePU 3: Portable high-level programming of heterogeneous systems and HPC clusters," *International Journal of Parallel Programming*, vol. 49, no. 6, pp. 846–866, 2021.

[11] M. Aldinucci, M. Danelutto, M. Drocco, P. Kilpatrick, C. Misale, G. Peretti Pezzi, and M. Torquati, "A parallel pattern for iterative stencil + reduce," *The Journal of Supercomputing*, vol. 74, no. 11, pp. 5690–5705, Nov 2018.

[12] Herrmann N., de Melo Menezes B.A., and Kuchen H.", "Stencil calculations with algorithmic skeletons for heterogeneous computing environments," *International Journal of Parallel Programming*, vol. 50, no. 5, pp. 433–453, 2022.

[13] Nina Herrmann, Justus Dieckmann, and Herbert Kuchen, "Optimizing three-dimensional stencil-operations on heterogeneous computing environments," *International Journal of Parallel Programming*, vol. 52, no. 4, pp. 274–297, August 2024.

[14] Johannes Pekkilä, Miikka S. Väisälä, Maarit J. Käpylä, Matthias Rheinhardt, and Oskar Lappi, "Scalable communication for high-order stencil computations using cuda-aware mpi," *Parallel Computing*, vol. 111, pp. 102904, 2022.

[15] Distributed and Parallel Systems Group, University of Innsbruck, "High-level c++ for accelerator clusters," `https://celerity.github.io/`.

[16] Khronos Group, "C++ programming for heterogeneous parallel computing," `https://www.khronos.org/sycl/`.

[17] Arturo Gonzalez-Escribano, Yuri Torres, Javier Fresno, and Diego R. Llanos, "An extensible system for multilevel automatic data partition and mapping.," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1145–1154, May 2014.

[18] Ana Moreton–Fernandez, Hector Ortega–Arranz, and Arturo Gonzalez–Escribano, "Controllers: An abstraction to ease the use of hardware accelerators," *The International Journal of High Performance Computing Applications*, vol. 32, no. 6, pp. 838–853, 2018.

[19] Yuri Torres, Francisco J. Andújar, Arturo Gonzalez-Escribano, and Diego R. Llanos, "Supporting efficient overlapping of host-device operations for heterogeneous programming with ctrlevents," *Journal of Parallel and Distributed Computing*, vol. 179, pp. 104708, 2023.

[20] Sergei Gorlatch and Murray Cole, *Parallel Skeletons*, pp. 1417–1422, Springer US, Boston, MA, 2011.

[21] David Díez-Poza, Sergio Alonso Pascual, and Arturo Gonzalez-Escribano, "Easy programming of stencil applications for large-scale systems with a parallel skeleton," *International Journal of Parallel Programming*, vol. 54, no. 6, April 2026.

[22] Hao Wang, Sreeram Potluri, Devendar Bureddy, Carlos Rosales, and Dhabaleswar K. Panda, "GPU-Aware MPI on RDMA-Enabled Clusters: Design, Implementation and Evaluation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 10, pp. 2595–2605, 2014.

[23] Wei Wu, George Bosilca, Rolf vandeVaart, Sylvain Jeaugey, and Jack Dongarra, "Gpu-aware non-contiguous data movement in open mpi," in *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, New York, NY, USA, 2016, HPDC '16, p. 231–242, Association for Computing Machinery.

[24] Michael J. Quinn, *Parallel Computing: Theory and Practice*, McGraw-Hill, New York, NY, second edition, 1994.

[25] Kaushik Kandadi Suresh, Kawthar Shafie Khorassani, Chen Chun Chen, Bharath Ramesh, Mustafa Abduljabbar, Aamir Shafi, Hari Subramoni, and Dhabaleswar K. Panda, "Network assisted non-contiguous transfers for gpu-aware mpi libraries," in *2022 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2022, pp. 13–20.

[26] Carl Pearson, Kun Wu, I-Hsin Chung, Jinjun Xiong, and Wen-Mei Hwu, "TEMPI: An interposed MPI library with a canonical representation of CUDA-aware datatypes," 2020.

[27] Timm Krüger, Halim Kusumaatmaja, Alexandr Kuzmin, Orest Shardt, Goncalo Silva, and Erlend Magnus Viggen, "The lattice boltzmann method," *Springer International Publishing*, vol. 10, no. 978-3, pp. 4–15, 2017.

Atención de Alto Rendimiento en Arquitectura Hopper para Clasificación Hiperespectral

Carlos Cañada-Rostro[1], María B. García-Flores[1], Mercedes E. Paoletti[1], y Juan M. Haut [1]

Resumen— **La clasificación de imágenes hiperespectrales mediante arquitecturas basadas en atención enfrenta un cuello de botella crítico derivado de la materialización de la matriz de afinidad $O(N^2)$ en la memoria de alto ancho de banda (HBM). Este fenómeno, agravado por la alta dimensionalidad espectral, delimita un muro de memoria que impide el procesamiento eficiente en secuencias largas. En este trabajo, se propone la integración de *FlashAttention-4* (FA4) en las arquitecturas HyperEAST y MVNet para mitigar este cuello de botella mediante una gestión *I/O-aware* de la jerarquía de memoria. Para optimizar la intensidad aritmética y activar el régimen de eficiencia del hardware, se introduce una estrategia de tokenización por sub-ventana espectral que densifica la secuencia de entrada, forzando la transición desde un régimen limitado por memoria (*memory-bound*) hacia uno limitado por cómputo (*compute-bound*). Evaluaciones exhaustivas en aceleradores NVIDIA H100 demuestran que esta sinergia permite una reducción drástica de la huella de memoria y una aceleración significativa del entrenamiento sin degradar la precisión predictiva, validando la viabilidad de procesar volúmenes masivos de datos en una única unidad de procesamiento.**

Palabras clave: Intensidad aritmética, jerarquía de memoria SRAM/HBM, FlashAttention-4, NVIDIA Hopper, fusión de kernels, modelado Roofline, Vision Transformers, paralelismo masivo.

I. Introducción

El procesamiento de imágenes hiperespectrales (HSI) representa un desafío computacional caracterizado por el manejo de volúmenes de datos tensoriales de gran dimensionalidad. A diferencia de la visión natural, cada píxel en HSI integra una firma espectral continua distribuida en centenares de bandas contiguas, lo que deriva en estructuras de datos masivas provenientes de misiones satelitales (EnMAP [1], PRISMA [2]) y constelaciones de CubeSats [3]. Esta carga de trabajo exige una capacidad de modelado global que capture dependencias de largo alcance, donde los mecanismos de atención han demostrado una eficacia sobresaliente [4, 5].

No obstante, el algoritmo de atención convencional presenta una complejidad cuadrática $O(N^2)$ tanto en cómputo como en memoria respecto a la longitud de la secuencia N. En arquitecturas de vanguardia como NVIDIA Hopper (H100), el procesamiento de secuencias hiperespectrales se enfrenta al denominado *muro de memoria* (*memory wall*) [6]: el cuello de botella sistémico no reside en la capacidad aritmética de los *Tensor Cores*, sino en la latencia y el ancho de banda asociados al tráfico de datos entre la memoria de alto ancho de banda (HBM) y los registros internos de la GPU. Cuando la longitud de secuencia es reducida ($N = C$), el sistema experimenta una infrautilización de los *Streaming Multiprocessors* (SMs), operando en un régimen estrictamente limitado por memoria (*memory-bound*).

Para mitigar esta limitación y explotar el paralelismo masivo del hardware moderno, se propone el empleo de *FlashAttention-4* (FA4) [7]. FA4 es una implementación de kernels fusionados (*fused kernels*) con gestión explícita del movimiento de datos (*I/O-aware*) que evita la materialización de la matriz de atención en la HBM, reduciendo el tráfico de datos a un orden lineal $O(N)$. La arquitectura H100 constituye el ecosistema idóneo para esta optimización, al permitir un solapamiento avanzado entre cómputo y transferencia mediante *Tensor Memory Accelerators* (TMA) y estrategias de especialización de hilos (*warp specialization*) [8].

Este trabajo analiza la integración de FA4 en arquitecturas del estado del arte como HyperEAST [9] y MVNet [10]. La contribución central reside en una estrategia de tokenización por sub-ventanas diseñada no solo para enriquecer la representación, sino para actuar como un balanceador de carga que incremente la ocupación de los SMs y desplace el régimen de ejecución hacia una limitación por cómputo (*compute-bound*). Las contribuciones principales se resumen a continuación:

- Estrategia de densificación de secuencia: Se propone una metodología de tokenización que incrementa N de forma controlada ($N = C \times \lfloor H/P_h \rfloor \times \lfloor W/P_w \rfloor$). Este aumento de la granularidad eleva la intensidad aritmética del mecanismo de atención, condición *sine qua non* para que FA4 alcance su máxima eficiencia en TFLOPS.
- Implementación hardware-aware: Se integra FA4 como operador de fusión determinista en modelos Vision Transformers (ViT) y arquitecturas híbridas Mamba-DenseNet, optimizando el *throughput* mediante la reducción de escrituras redundantes en la jerarquía de memoria.
- Análisis de rendimiento y escalabilidad: Se realiza una evaluación cuantitativa exhaustiva empleando aceleradores NVIDIA H100, analizando métricas de latencia de kernel, utilización de VRAM y el impacto del *Transformer Engine* en el escalado del sistema sobre siete conjuntos de datos de referencia.

[1]Departamento de Tecnología de Computadores y Comunicaciones, Universidad de Extremadura, correo: juanmariohaut@unex.es.

II. Trabajo Relacionado

A. Perfil Computacional de las Arquitecturas Hiperespectrales

La evolución de la clasificación de imágenes hiperespectrales (HSI) ha estado marcada por la adopción de modelos con una demanda creciente de ancho de banda y capacidad de cómputo. Inicialmente, los *Spectral Transformers* [5] introdujeron el mecanismo de auto-atención para modelar firmas espectrales, desplazando el cuello de botella desde las operaciones convolucionales locales hacia productos tensoriales globales. No obstante, la convergencia hacia arquitecturas híbridas ha buscado equilibrar la extracción de características mediante operadores densos con la agregación de dependencias de largo alcance.

En este ámbito, HyperEAST [9] ejemplifica el paradigma *Vision Transformer* (ViT) aplicado a HSI, donde el perfil de ejecución está dominado por operaciones de atención global con una complejidad cuadrática $O(N^2)$ en el plano de memoria. Por otro lado, MVNet [10] introduce Modelos de Espacio de Estados (SSM) de la familia Mamba, cuya primitiva fundamental es un escaneo selectivo con escalado lineal. A pesar de la eficiencia teórica de los SSM en la fase recurrente, la etapa de fusión global en estas redes híbridas sigue dependiendo de operadores de atención estándar. Desde una perspectiva de sistemas, estas implementaciones "naive"sufren de una infrautilización de la jerarquía de memoria, ya que no minimizan el tráfico hacia la HBM en secuencias de alta dimensionalidad.

B. Optimización de Kernels de Atención y Eficiencia en el Movimiento de Datos

La optimización de la capa de atención es un área crítica en la computación de alto rendimiento (HPC) para mitigar el impacto del movimiento de datos. FlashAttention [11,12] reestructuró el algoritmo mediante técnicas de particionamiento por bloques (*tiling*) y recomputación, permitiendo que el cálculo de la atención resida de forma persistente en la memoria compartida (*SRAM on-chip*). Este enfoque de "fusión de kernels"transforma una operación limitada por ancho de banda en una limitada por cómputo, maximizando la utilización de los *Tensor Cores*.

Con la arquitectura NVIDIA Hopper, FlashAttention-3 [13] introdujo la asincronía de hardware mediante unidades *Tensor Memory Accelerator* (TMA), permitiendo solapar el movimiento de datos con el cómputo aritmético. La iteración FlashAttention-4 (FA4) [7] profundiza en esta especialización mediante un *co-pipelining* agresivo y optimizaciones en el operador *softmax* condicional, aprovechando la precisión reducida y la aceleración de estructuras dispersas en arquitecturas Blackwell, manteniendo la compatibilidad de rendimiento en Hopper.

Se constata una brecha técnica significativa en la literatura de teledetección: las implementaciones dominantes en HSI delegan la ejecución en librerías de alto nivel que materializan matrices intermedias en la HBM. Este enfoque ignora las capacidades de gestión explícita de la jerarquía de memoria (*memory-hierarchy awareness*), resultando en una baja intensidad aritmética. El presente trabajo aborda esta deficiencia mediante la integración sistemática de kernels FA4 optimizados, diseñados para explotar el paralelismo masivo y la persistencia de datos en SRAM en escenarios de alta dimensionalidad espectral.

III. Metodología

A. Densificación de Carga mediante Tokenización por Sub-ventana

Considérese un volumen de datos de entrada en HSI denotado por $\mathbf{X} \in \mathbb{R}^{\mathbb{C}\times\mathbb{H}\times\mathbb{W}}$, donde C representa el número de bandas espectrales y $H \times W$ la resolución espacial del parche. En arquitecturas convencionales como HyperEAST, la proyección de bandas individuales genera una longitud de secuencia $N = C$. Para aceleradores de arquitectura Hopper (H100), este valor de N (≈ 270) resulta insuficiente para saturar los 132 *Streaming Multiprocessors* (SMs) disponibles. En este régimen, el sistema experimenta un fenómeno de *SM starvation*, operando bajo una limitación estricta de ancho de banda y dominado por el *kernel launch overhead*, lo que impide alcanzar el rendimiento pico teórico del hardware.

Con el objetivo de desplazar la ejecución hacia el límite de cómputo del modelo *Roofline*, se propone una estrategia de densificación de la secuencia mediante particionamiento espacial. El dominio se subdivide en una malla uniforme de $\lfloor H/P_h \rfloor \times \lfloor W/P_w \rfloor$ bloques disjuntos, tal como se ilustra en la Figura 1. Para cada banda $c \in \{1, \ldots, C\}$ y cada sub-ventana (i, j), se define el token resultante como:

$$t_{c,i,j} = \mathbf{X}_{\mathbf{c},\ \mathbf{iP_h}:\mathbf{(i+1)P_h},\ \mathbf{jP_w}:\mathbf{(j+1)P_w}} \in \mathbb{R}^{\mathbb{P}_{\approx}\times\mathbb{P}_{\lessapprox}}. \quad (1)$$

La longitud de secuencia resultante bajo esta reestructuración es:

$$N = C \times \left\lfloor \frac{H}{P_h} \right\rfloor \times \left\lfloor \frac{W}{P_w} \right\rfloor. \quad (2)$$

Esta reformulación incrementa significativamente N, permitiendo un *grid sizing* más agresivo en CUDA y facilitando el solapamiento de accesos a memoria con el cómputo aritmético. Bajo la configuración $H = W = 7$ y $P_h = P_w = 2$, se obtienen 9 sub-ventanas por banda, elevando la longitud total a $N > 2{,}400$ (Tabla I). Este incremento desplaza la ejecución desde un régimen *memory-bound* hacia uno *compute-bound*. De forma precisa, la intensidad aritmética I puede aproximarse como:

$$I \approx \frac{O(N^2 d)}{O(N^2)} = O(d), \quad (3)$$

permitiendo que la razón FLOPs/byte alcance el umbral necesario para que el rendimiento quede limitado por los TFLOPS de los *Tensor Cores* y no por el tráfico efectivo hacia la HBM.

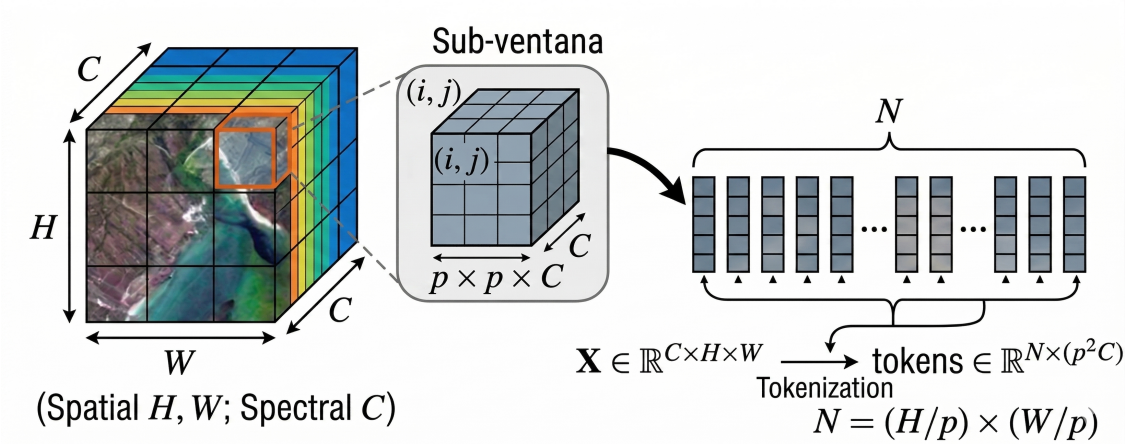


Fig. 1: Esquema de densificación de carga. La partición espacial incrementa el paralelismo a nivel de hilo (*TLP*) y permite saturar los *Tensor Cores* mediante secuencias de alta intensidad aritmética.

Tabla I: Evolución de la longitud de secuencia N.

Dataset	N_{orig}	N
WHU-Hi HanChuan	274	2.466
WHU-Hi HongHu	270	2.430

Para preservar la coherencia estructural entre las dimensiones espectral y espacial bajo esta secuencia densa, se emplea un *embedding* posicional tridimensional factorizado:

$$p_{c,i,j} = e^{(c)} + e^{(i)} + e^{(j)}. \tag{4}$$

Esta descomposición permite codificar de forma independiente la identidad espectral y la localización espacial, evitando ambigüedades tras la expansión del espacio de búsqueda de la atención.

B. Orquestación de Memoria mediante FlashAttention-4

Sea $\mathbf{X} \in \mathbb{R}^{\mathbb{N}\times}$ la secuencia densificada obtenida. Las proyecciones en los espacios de consulta, clave y valor se definen como $\mathbf{Q} = \mathbf{X}\mathbf{W_Q}$, $\mathbf{K} = \mathbf{X}\mathbf{W_K}$ y $\mathbf{V} = \mathbf{X}\mathbf{W_V}$. El cálculo de atención convencional implica la materialización de la matriz de afinidad $\mathbf{S} = \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{\mathbf{d_h}}} \in \mathbb{R}^{\mathbb{N}\times\mathbb{N}}$, lo que conlleva una complejidad de memoria $O(N^2)$ y una saturación del bus de datos. Para $N \approx 2{,}500$, el tráfico hacia la HBM domina el tiempo de ejecución ($T_{\text{I/O}} \gg T_{\text{compute}}$).

FA4 mitiga este cuello de botella mediante una implementación de kernels fusionados con conciencia de la jerarquía (*hierarchy-aware*). Utilizando la librería CuTe para la gestión de *layouts* tensoriales, el algoritmo implementa un *software pipelining* que mantiene acumuladores persistentes en la memoria compartida (*SRAM on-chip*):

$$m_i^{(j)} = \text{máx}\left(m_i^{(j-1)}, \text{rowmax}(\mathbf{S_{ij}})\right), \tag{5}$$

$$l_i^{(j)} = e^{m_i^{(j-1)} - m_i^{(j)}} l_i^{(j-1)} + \text{rowsum}(e^{\mathbf{S_{ij}} - \mathbf{m_i^{(j)}}}). \tag{6}$$

La salida $\mathbf{O_i}$ se actualiza de forma iterativa mediante el esquema de *online softmax*:

$$\mathbf{O_i^{(j)}} = \frac{\mathbf{e^{m_i^{(j-1)} - m_i^{(j)}} l_i^{(j-1)} O_i^{(j-1)}} + \mathbf{e^{S_{ij} - m_i^{(j)}} V_j}}{\mathbf{l_i^{(j)}}}. \tag{7}$$

Este procedimiento garantiza estabilidad numérica y reduce la complejidad de memoria a $O(Nd)$. Nuestra implementación en NVIDIA H100 explota tres primitivas de hardware críticas para maximizar el *throughput*:

- Tensor Memory Accelerator (TMA): Gestiona la transferencia asíncrona de bloques $\mathbf{K}$ y $\mathbf{V}$ desde la HBM a la SRAM, eliminando instrucciones explícitas de carga (*LDG*) y liberando registros para el cómputo.
- Warp Group Specialization: Asigna roles diferenciados a los grupos de *warps* para solapar las etapas de carga (*producers*) y cómputo (*consumers*), mitigando los *stalls* por dependencia de datos.
- WGMMA (Warpgroup MMA): Ejecuta operaciones de multiplicación de matrices directamente sobre los descriptores de memoria en SRAM, aprovechando los *Tensor Cores* de 4ª generación.

Fijando $d_{\text{head}} = 128$ para asegurar el alineamiento óptimo con las líneas de caché L2, esta orquestación técnica permite que la eficiencia del sistema transite hacia un régimen determinado por la capacidad aritmética de la arquitectura Hopper, eliminando el desperdicio de ciclos asociado al movimiento de datos en escenarios de alta dimensionalidad.

IV. Experimentos y Análisis de Rendimiento

La evaluación experimental se ha diseñado para caracterizar el comportamiento del sistema bajo condiciones de alta carga computacional y estrés de memoria. Se emplean tres conjuntos de datos hiperespectrales de la serie WHU-Hi: HanChuan y HongHu [14]. Estos *workloads* presentan una alta dimensionalidad espectral ($C \approx 270$), lo que tras el proceso de densificación de carga propuesto, genera secuencias de tokens masivas ideales para evaluar el escalado de kernels en la arquitectura Hopper.

A. *Marco Experimental y Verificación de Fidelidad Algorítmica*

La plataforma experimental consiste en nodos equipados con GPUs NVIDIA H100 (SXM5), que integran Tensor Cores de cuarta generación y el motor *Transformer Engine*. Se utiliza precisión mixta (`bfloat16` y FP8) para maximizar el *throughput* aritmético. Las métricas registradas incluyen latencia de kernel (*forward* y *backward*), *throughput* global del sistema (muestras/s) y consumo pico de VRAM. Para garantizar la reproducibilidad, cada medición es el promedio de 100 iteraciones tras una fase de *warm-up* de 20 épocas.

Se analizan cuatro configuraciones: HyperEAST y MVNet, ambas bajo atención estándar (SDPA) y bajo el operador optimizado FA4. Bajo la denominación *Std*, se emplea el operador nativo de PyTorch para SDPA. Es importante mencionar que, dependiendo de la versión del framework, este operador actúa como un despachador automático que puede ejecutar implementaciones de FlashAttention-1 [11] o FlashAttention-2 [12] según la compatibilidad de los tipos de datos y el hardware. Por tanto, los resultados obtenidos con FA4 representan una mejora directa sobre el estado del arte de las librerías optimizadas actuales, y no simplemente sobre una implementación de referencia no acelerada.

Tabla II: Validación de fidelidad algorítmica: OA, AA y Kappa.

Dataset	Modelo / Atención	OA (%)	AA (%)	κ
WHU-Hi HanChuan	HyperEAST / Std	75,3	68,1	0,71
	HyperEAST / FA4	75,6	68,9	0,72
	MVNet / Std	95,7	90,8	0,95
	MVNet / FA4	**96,1**	**92,4**	**0,95**
WHU-Hi HongHu	HyperEAST / Std	74,4	70,7	0,69
	HyperEAST / FA4	72,1	67,3	0,66
	MVNet / Std	**97,3**	**93,5**	**0,97**
	MVNet / FA4	97,1	92,6	0,96

B. *Análisis de Eficiencia Sistémica y Utilización de Hardware*

El núcleo de la evaluación reside en la capacidad de FA4 para mitigar el *memory wall*. La Tabla III y la Figura 2 detallan el comportamiento del sistema. Se observa una asimetría de rendimiento explicable mediante el análisis de la ruta crítica de ejecución. En HyperEAST, donde la atención es el componente computacional dominante tras la densificación de la secuencia ($N \approx 2{,}400$), FA4 logra una reducción de VRAM de hasta 35,9 GB. Esta mejora disruptiva proviene de evitar la materialización de la matriz de afinidad en la HBM, permitiendo que el *working set* resida de forma persistente en la SRAM. El *speedup* total de hasta 3,01× indica que el sistema ha superado el régimen *memory-bound*, maximizando la ocupación de los *warps* y aprovechando las unidades TMA de la H100 para ocultar latencias de transferencia.

En contraste, el perfil de ejecución de MVNet presenta una saturación del beneficio. De acuerdo con

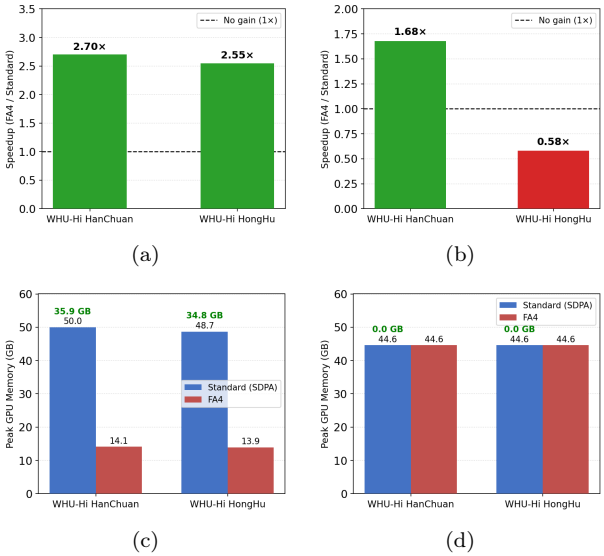


Fig. 2: Análisis de escalabilidad y eficiencia: (a) speedup del paso adelante en HyperEAST, (b) speedup del paso adelante en MVNet, (c) consumo pico de VRAM en HyperEAST, (d) consumo pico de VRAM en MVNet. Se observa la transición exitosa al régimen *compute-bound* en HyperEAST, mientras que MVNet presenta una saturación debida a la Ley de Amdahl.

la Ley de Amdahl, la aceleración global está limitada por la fracción del código no optimizable por el kernel de atención; en este caso, el coste dominante reside en las capas Mamba (scans selectivos) y bloques convolucionales densos. A pesar de lograr *speedups* locales en el paso adelante, la aceleración total converge a 1,00×. Mención especial requiere la regresión observada en el *forward* de MVNet sobre HongHu (0,58×), fenómeno característico de kernels optimizados como FA4 cuando se aplican a bloques de cómputo insuficientes para amortizar el *overhead* asociado a la configuración del TMA y la sincronización de *Warpgroups*. En conclusión, la tokenización propuesta actúa como la técnica de balanceo de carga necesaria para que el mecanismo de atención alcance la masa crítica de hilos requerida por la arquitectura Hopper.

Tabla III: Métricas de eficiencia computacional sobre arquitectura Hopper.

Modelo	Dataset	Speedup Fwd	Speedup Total	VRAM Std	VRAM FA4
HyperEAST	HanChuan	2,70×	2,30×	50,0	14,1
	HongHu	2,55×	2,35×	48,7	13,9
MVNet	HanChuan	1,68×	1,01×	44,6	44,6
	HongHu	0,58×	1,00×	44,6	44,6

V. Conclusiones

Este trabajo ha presentado la integración de *FlashAttention-4* (FA4) en el módulo de fusión espectro-espacial de arquitecturas hiperespectrales de vanguardia. Se ha demostrado que la estrategia de tokenización por sub-ventana espectral actúa como un prerrequisito sistémico fundamental: al densificar la secuencia de tokens, se eleva la intensidad aritmética del operador de atención, desplazando la ejecución desde un régimen limitado por ancho de banda (*memory-bound*) hacia uno limitado por cómputo (*compute-bound*), activando así las capacidades de

aceleración de FA4.

Los resultados obtenidos en aceleradores NVIDIA H100 validan esta sinergia, logrando una reducción en el consumo de VRAM (hasta 35,9 GB en HyperEAST) y aceleraciones en el paso adelante de entre 2,55× y 2,70×. Asimismo, el análisis comparativo con MVNet revela que la efectividad de estos kernels optimizados está sujeta a la Ley de Amdahl; en arquitecturas donde el mecanismo de atención no reside en la ruta crítica de ejecución o representa una fracción minoritaria de la intensidad aritmética total, los beneficios de la fusión de kernels quedan diluidos por el coste de otros módulos funcionales.

Como trabajo futuro, se prevé extender esta evaluación a la arquitectura NVIDIA Blackwell (B200). En dicho ecosistema, se espera que la creciente asimetría entre el *throughput* de los *Tensor Cores* y el ancho de banda de memoria acentúe las ventajas competitivas de las implementaciones *I/O-aware*. Finalmente, se explorará la integración de estos kernels en los componentes recurrentes de los Modelos de Espacio de Estados y el desarrollo de estrategias de tokenización adaptativas fundamentadas en la entropía espectral local.

Agradecimientos

Los autores agradecen la financiación proporcionada por la Consejería de Economía, Ciencia y Agenda Digital de la Junta de Extremadura y por el Fondo Europeo de Desarrollo Regional (FEDER), en el marco de las ayudas a grupos de investigación de la Junta de Extremadura, expediente GR24035. Asimismo, este trabajo ha sido financiado parcialmente por el proyecto de I+D PID2023-149880NA-I00, financiado por MICIU/AEI/10.13039/501100011033; por la Junta de Extremadura, referencia IB24090; y por una Beca Leonardo 2024 para Investigación Científica y Creación Cultural de la Fundación BBVA. La Fundación BBVA no se hace responsable de las opiniones, declaraciones o contenidos incluidos en el proyecto, ni de los resultados derivados del mismo, que son responsabilidad exclusiva de los autores.Este trabajo también ha recibido financiación del Fondo Europeo de Desarrollo Regional (FEDER) de la Unión Europea, a través del programa Interreg España-Portugal (POCTEP) 2021-2027, bajo la ayuda 0206_RAT_EOS_PC_6_E.

Referencias

[1] Luis Guanter, Hermann Kaufmann, Karl Segl, Saskia Foerster, Christian Rogass, Sabine Chabrillat, Theres Kuester, André Hollstein, Godela Rossner, Christian Chlebek, et al., "The enmap spaceborne imaging spectroscopy mission for earth observation," *Remote Sensing*, vol. 7, no. 7, pp. 8830–8857, 2015.

[2] Stefano Pignatti, Angelo Palombo, Simone Pascucci, Filomena Romano, Federico Santini, Tiziana Simoniello, Amato Umberto, Cuomo Vincenzo, Nicola Acito, Marco Diani, et al., "The prisma hyperspectral mission: Science activities and opportunities for agriculture and land monitoring," in *2013 IEEE international geoscience and remote sensing symposium-IGARSS*. IEEE, 2013, pp. 4558–4561.

[3] Kristina Lemmer, "Propulsion for cubesats," *Acta astronautica*, vol. 134, pp. 231–243, 2017.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, vol. 30.

[5] Xin He, Yushi Chen, and Zhouhan Lin, "Spatial-spectral transformer for hyperspectral image classification," *Remote Sensing*, vol. 13, no. 3, pp. 498, 2021.

[6] Avinash Kumar Maurya, M Mustafa Rafique, Franck Cappello, and Bogdan Nicolae, "Mlp-offload: Multi-level, multi-path offloading for llm pre-training to break the gpu memory wall," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2025, pp. 1381–1394.

[7] Ted Zadouri, Markus Hoehnerbach, Jay Shah, Tao Liu, Vijay Thakkar, and Tri Dao, "FlashAttention-4: Algorithm and kernel pipelining co-design for asymmetric hardware scaling," *arXiv preprint arXiv:2603.05451*, 2026.

[8] NVIDIA Corporation, "NVIDIA hopper architecture in-depth," Tech. Rep., NVIDIA Whitepaper, 2022.

[9] Jialin Tang, Nan Ma, Chen Jia, Rui Tian, and Yanhui Guo, "Hypereast: An enhanced attention-based spectral–spatial transformer with self-supervised pretraining for hyperspectral image classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 18, pp. 22241–22255, 2025.

[10] Guandong Li and Mengxia Ye, "Mvnet: Hyperspectral remote sensing image classification based on hybrid mamba-transformer vision backbone architecture," 2025.

[11] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré, "FlashAttention: Fast and memory-efficient exact attention with IO-awareness," in *Advances in Neural Information Processing Systems*, 2022, vol. 35, pp. 16344–16359.

[12] Tri Dao, "FlashAttention-2: Faster attention with better parallelism and work partitioning," *arXiv preprint arXiv:2307.08691*, 2023.

[13] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao, "FlashAttention-3: Fast and accurate attention with asynchrony and low-precision," in *Advances in Neural Information Processing Systems*, 2024, vol. 37, pp. 68658–68685.

[14] Xin Hu, Yanfei Zhong, Chang Luo, and Xinyu Wang, "Whu-hi: Uav-borne hyperspectral with high spatial resolution (h2) benchmark datasets for hyperspectral image classification," *arXiv preprint arXiv:2012.13920*, 2020.

[15] José M. Bioucas-Dias, Antonio Plaza, Gustau Camps-Valls, Paul Scheunders, Nasser Nasrabadi, and Jocelyn Chanussot, "Hyperspectral remote sensing data analysis and future challenges," *IEEE Geoscience and Remote Sensing Magazine*, vol. 1, no. 2, pp. 6–36, 2013.

[16] Farhan Ullah, Inam Ullah, Rizwan Ullah Khan, Salabat Khan, Khalil Khan, and Giovanni Pau, "Conventional to deep ensemble methods for hyperspectral image classification: A comprehensive survey," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 17, pp. 3878–3916, 2024.

[17] Telmo Adão, Jonáš Hruška, Luís Pádua, José Bessa, Emanuel Peres, Raul Morais, and Joaquim J. Sousa, "Hyperspectral imaging: A review on UAV-based sensors, data processing and applications for agriculture and forestry," *Remote Sensing*, vol. 9, no. 11, pp. 1110, 2017.

[18] Saskia Foerster, Véronique Carrère, Michael Rast, and Karl Staenz, "Preface: The environmental mapping and analysis program (EnMAP) mission: Preparing for its scientific exploitation," 2016.

[19] Minsu Kim, Sangwook Park, Christopher Anderson, and Gary L. Stensaas, "System characterization report on precursore iperspettrale della missione applicativa (PRISMA)," Tech. Rep., US Geological Survey, 2022.

Aplicaciones de la computación de altas prestaciones (3/3)

Análisis y optimización microarquitectónica del algoritmo de Ruido Perlin 2D en AMD Zen 3

Luis Miguel Trinidad Salvador[1], Maria José Morón Fernández[2]

Resumen— **Este artículo presenta un conjunto de optimizaciones microarquitectónicas del algoritmo de ruido Perlin bidimensional sobre procesadores AMD Zen 3. Mediante el lenguaje Zig, se aplican iterativamente cuatro técnicas con el fin de maximizar el *throughput*: reestructuración de accesos a memoria (Pseudotiling), desacoplamiento matemático de ejes (LICM), vectorización explícita SIMD (AVX2) y aritmética de precisión fija (Q15). Los resultados evidencian una aceleración de 151x, alcanzando una producción de más de 70GB/s, superando a las librerías generalistas del estado del arte en la generación de mallas regulares.**

Palabras clave— **Ruido Perlin, Optimización microarquitectónica, SIMD, AVX2, Zen 3, Punto fijo, Generación procedimental.**

I. Introducción

La generación procedimental de contenido (PCG) se ha consolidado como una herramienta fundamental en la computación gráfica moderna, permitiendo la creación de entornos vastos y detallados con un coste de almacenamiento mínimo. En el núcleo de estas técnicas se encuentra el ruido coherente de gradiente, siendo el ruido Perlin, desarrollado por Ken Perlin en 1985 [1], el estándar de facto en la industria.

El algoritmo funciona mediante la interpolación del producto escalar entre gradientes generados de forma aleatoria dentro de una rejilla hipercúbica y la posición relativa del punto a evaluar dentro de la misma, de modo que para cada punto k, sus gradientes asociados corresponden a los 2^n vértices de la celda.

Originalmente diseñado con el fin de introducir irregularidades de apariencia natural en texturas sintéticas, el algoritmo ha evolucionado para abordar limitaciones estéticas y computacionales. Sin embargo, su implementación eficiente en sistemas de tiempo real sigue representando un desafío debido a su carga aritmética y la necesidad de accesos a memoria no contiguos.

Para la fase inicial de la evaluación, se ha seleccionado la microarquitectura Zen 3 como línea base. Esta elección se fundamenta en que Zen 3 se comporta como una fiel representación del mercado de hardware de consumo actual, al implementar el conjunto de instrucciones AVX2 (presente en el 94.84 % de los sistemas en uso) pero carecer de soporte para AVX-512 (cuya adopción se limita al 21.89 %) [2].

[1]Estudiante del Grado de Ingeniería Informática - Tecnologías Informáticas, e-mail: luitrisal@alum.us.es. orcid: https://orcid.org/0009-0007-9215-9754

[2]Escuela Politecnica Superior, Univ. de Sevilla, c/ Niebla, s/n, Seville (Spain), orcid: https://orcid.org/0000-0003-3291-7297

La contribución de este trabajo se centra en la optimización y el análisis de rendimiento de este algoritmo sobre la microarquitectura AMD Zen 3, explotando sus capacidades de vectorización y su jerarquía de memoria.

El trabajo presentado está organizado de acuerdo con la siguiente estructura: Se comienza, en la sección II, analizando el estado actual de las optimizaciones del ruido Perlin, así como las características técnicas de la arquitectura utilizada como referencia en el estudio. A continuación, en la sección III, se expone el fundamento teórico que constituye la base de las optimizaciones y modificaciones del algoritmo realizadas. Seguidamente, en la sección IV se detalla el resultado de la evaluación de las técnicas aplicadas y se compara la implementación propuesta con el estado del arte. Finalmente, en la sección V, se concluye con una interpretación de los resultados y se plantean las líneas futuras de continuación de este proyecto.

II. Conceptos Previos y Estado del Arte

Se comienza introduciendo tanto la plataforma de ejecución subyacente como el estado del arte actual, para comprender las optimizaciones aplicadas.

A. Microarquitectura Zen 3

La microarquitectura Zen 3 de AMD, presentada en noviembre de 2020, representa un salto significativo en el rendimiento por ciclo de reloj (IPC), *Instrucciones por ciclo*, respecto a sus predecesoras. Desde la perspectiva de la computación de alto rendimiento (HPC, *High Performance Computing*) y la vectorización, sus características más relevantes para este trabajo radican en la flexibilidad de sus unidades de cálculo y su robusto subsistema de memoria.

A nivel de ejecución, el núcleo Zen 3 destaca por un motor de coma flotante y SIMD (*Single Instruction, Multiple Data*) altamente superescalar. Según los detalles técnicos presentados en [3], [4], la unidad vectorial cuenta con cuatro puertos de ejecución de 256 bits de ancho, dos de los cuales pueden ejecutar multiplicaciones y multiplicación-suma fusionada. Esta configuración no solo procesa operaciones de coma flotante con alta eficiencia, sino que ofrece un *throughput* excepcional para la aritmética de enteros mediante el conjunto de instrucciones AVX2.

En estas arquitecturas, el paralelismo a nivel de datos (DLP, *Data Level Parallelism*), depende directamente del tamaño del tipo numérico empleado. Dado que los registros tienen un ancho fijo de 256 bits, sustituir la precisión estándar (como la coma flotante de 32 bits) por formatos numéricos más compactos

multiplica el número de elementos que pueden empaquetarse por instrucción. Esta capacidad, combinada con el despacho superescalar de instrucciones vectoriales del núcleo, constituye una estrategia de optimización fundamental para algoritmos con una alta intensidad aritmética.

En cuanto a la jerarquía de memoria, en Zen 3 ocho núcleos conforman un *Core Complex* (CCX) unificado y comparten un único bloque de caché L3 de 32 MB, lo que reduce drásticamente la latencia de comunicación entre hilos. La optimización de algoritmos de generación masiva de datos requiere no solo un uso intensivo de registros, sino una escritura eficiente. En la Tabla I se resumen las características de la memoria del procesador evaluado de acuerdo con los datos proporcionados por AMD [5]. Disponer de un elevado ancho de banda y baja latencia en la caché L1 es crítico en este contexto, ya que permite mitigar las penalizaciones de rendimiento derivadas de accesos y escrituras a memoria no secuenciales, un patrón que surge frecuentemente al aplicar reestructuraciones algorítmicas sobre los bucles de iteración espaciales.

Tabla I: Jerarquía de memoria y latencias típicas en la microarquitectura AMD Zen 3

Nivel	Capacidad	Vías	Latencia (ciclos)
L1	32 KB	8-way	∼4-5 ciclos
L2	512 KB	8-way	∼12-14 ciclos
L3	32 MB	16-way	∼40-46 ciclos

B. Fundamentos del Algoritmo

El proceso de cálculo para cada punto contenido en la rejilla hipercúbica consta de tres pasos:

1. **Calcular la influencia de cada vértice:** Para cada uno de los 2^n vértices de la celda en la que se encuentra el punto, se calcula el producto escalar entre su vector de gradiente pseudoaleatorio $\vec{v}$ y el vector de distancia desde las coordenadas hasta el vértice asociado al vector.
2. **Suavizar las coordenadas:** Las coordenadas del punto dentro de la celda se suavizan a través de la función *smootherstep* para asegurar transiciones suaves entre *chunks*.

$$S(t) = 6t^5 - 15t^4 + 10t^3 \tag{1}$$

3. **Interpolar:** Se realizan interpolaciones lineales sucesivas utilizando las coordenadas suavizadas para combinar las influencias de las esquinas, obteniendo así el valor de ruido final.

En el caso bidimensional, con cuatro vértices presentes se requieren 4 productos escalares y 3 interpolaciones. La complejidad algorítmica de este algoritmo escala de forma exponencial $\mathcal{O}(M \cdot 2^N)$, siendo M el número de puntos a evaluar, y N la dimensión de los mismos.

Si bien es cierto que existe una alternativa conocida como ruido Simplex [6], que presenta una complejidad dimensional inferior de $\mathcal{O}(M \cdot N^2)$, su mayor carga computacional en la fase de interpolación y en la determinación de los gradientes adyacentes hace que no presente una mejora de eficiencia práctica significativa hasta alcanzar la cuarta dimensión.

Para respaldar empíricamente esta afirmación, se han realizado pruebas de rendimiento utilizando la biblioteca FastNoise2 [7], cuyo código fuente y documentación de *benchmarks* están disponibles públicamente en su repositorio de GitHub (`https://github.com/Auburn/FastNoise2`). Los datos se pueden consultar en la sección IV.

C. Estado del Arte en Generación de Ruido

Desde la publicación del algoritmo original, la optimización del ruido Perlin ha sido un área de investigación activa [8], fuertemente ligada a la evolución de las arquitecturas de hardware. Las implementaciones iniciales y las bibliotecas clásicas, como *libnoise* [9], se basaban en un enfoque escalar y estaban orientadas a la flexibilidad compositiva en la CPU, lo que las hace ineficientes para los estándares de tiempo real actuales.

Con la llegada de las extensiones vectoriales modernas, el paradigma de optimización migró hacia la vectorización SIMD. El principal desafío en esta transición ha sido la adaptación de la función de *hash* espacial. El algoritmo clásico de Perlin depende de una tabla de permutación precalculada de 256 elementos [1] para asignar vectores de gradiente a las coordenadas. En un contexto escalar, esto requiere accesos a memoria dependientes de los datos, lo cual rompe el paralelismo en arquitecturas SIMD y provoca cuellos de botella por latencia de memoria o limitaciones en las instrucciones *gather*. Las soluciones contemporáneas suelen mitigar esto sustituyendo las búsquedas en tablas por funciones de *hash* puramente aritméticas o rediseñando el diseño de datos hacia estructuras de arrays (SoA).

En la actualidad, el referente y estándar de facto en cuanto a rendimiento *open-source* es la biblioteca *FastNoise2* [7]. Diseñada como la evolución de *FastNoiseSIMD*, esta biblioteca maximiza el rendimiento aprovechando conjuntos de instrucciones avanzados (AVX2, AVX-512 y ARM NEON) mediante despacho dinámico (*dynamic dispatch*). Su motor interno está altamente optimizado para calcular múltiples puntos simultáneamente operando sobre registros de 32 bits en coma flotante (`fp32`), alcanzando niveles de *throughput* que representan el límite práctico para implementaciones de propósito general.

Otras alternativas populares, como *stb_perlin* [10] o los módulos de ruido integrados en motores comerciales (ej. Unreal Engine [11] o Unity [12]), priorizan la portabilidad o la integración directa en la canalización gráfica (GPU) mediante *shaders* (HLSL/GLSL). Sin embargo, cuando la generación procedimental masiva debe ocurrir del lado de la CPU, como es común en la generación de mundos basados en vóxeles o mallas dinámicas, *FastNoise2* se erige como la métrica a batir.

No obstante, el diseño de bibliotecas de propósito general como *FastNoise2* exige mantener una preci-

sión geométrica estandarizada (`fp32`) y una evaluación de puntos flexible e independiente. Esto deja un margen de exploración arquitectónica en escenarios donde es admisible relajar ciertas restricciones de precisión a cambio de un mayor paralelismo, o donde la topología del terreno permite reestructurar el patrón de evaluación espacial para explotar invariantes algorítmicas a nivel de microarquitectura.

III. Metodología y Optimizaciones Aplicadas

En este apartado se describe el proceso de refinamiento iterativo aplicado al algoritmo de ruido Perlin. Este algoritmo constituye el núcleo de la gran mayoría de sistemas de generación procedimental de terreno, estableciendo la macro-topografía que posteriormente puede ser refinada mediante algoritmos de erosión u otras alternativas.

El objetivo fundamental de esta etapa es maximizar el aprovechamiento de los recursos del procesador. Para ello, se ha partido de una implementación escalar de referencia (`00_ref.zig`) [13] sobre la cual se han aplicado sucesivamente cuatro niveles de optimización, cuyo código fuente se encuentra disponible en el repositorio de este proyecto.

Las optimizaciones que se describen a lo largo de esta sección son:

1. **Capa de cacheado & Pseudotiling:** Reestructuración del patrón de acceso y cacheado de componentes, permitiendo evitar el cálculo de gradientes y valores suavizados para cada punto.
2. **LICM *(Loop Invariant Code Motion)* & Refinamiento aritmético:** Extracción de código invariante y reestructuración de cálculos del algoritmo para desacoplar ejes y reducir el bucle crítico.
3. **Vectorización SIMD:** Aprovechamiento de las unidades de cálculo vectorial de la CPU.
4. **Aritmética de precisión fija (FPA):** Migración del tipo de datos del algoritmo de `fp32` a `i16`, permitiendo duplicar el ancho de vector.

A. Implementación de Referencia

La implementación de referencia constituye la línea base necesaria para la evaluación del rendimiento. Se trata de una versión escalar que respeta la estructura original del algoritmo de Perlin, implementada en punto flotante de 32 bits (`f32`) y sin explotar características avanzadas como vectorización o alineación de memoria.

Para establecer una línea base adecuada, se asume y verifica en tiempo de compilación que tanto la escala como el tamaño del *chunk* son potencias de dos (2^n). Esta restricción permite sustituir la costosa operación de módulo por una operación *bitwise AND* empleando la longitud de la máscara $(x-1)$, verificándose matemáticamente que:

$$\begin{aligned} &\forall m \in \mathbb{Z}^+ \forall x \in \{2^k \mid k \in \mathbb{N}\}, \\ &\quad m \bmod x = m \,\&\, (x-1) \end{aligned} \tag{2}$$

Para cada coordenada evaluada, los gradientes se extraen de una tabla de búsqueda generada en tiempo de compilación mediante las herramientas de metaprogramación de Zig [14]. Esta tabla contiene 64 vectores bidimensionales distribuidos equidistantemente sobre una circunferencia de radio $\sqrt{2}$ de acuerdo con la expresión:

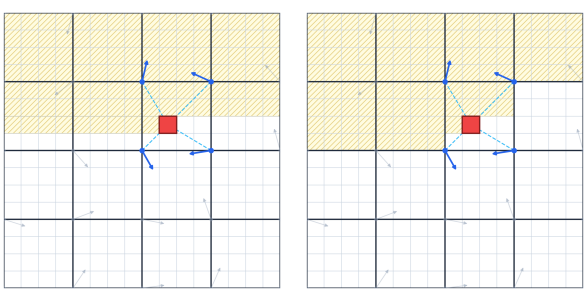
$$\begin{aligned} &\forall i \in \{0, 1, \ldots, 63\}, \\ &g_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} \sqrt{2}\sin\left(\frac{i\pi}{32}\right) \\ \sqrt{2}\cos\left(\frac{i\pi}{32}\right) \end{pmatrix} \end{aligned} \tag{3}$$

A diferencia de la implementación original, que accede a los gradientes directamente, la implementación propuesta emplea la función *hash* rapidhash [15] integrada estáticamente a aridad fija para recibir como parámetros semilla y coordenadas x e y.

B. Cacheado y Reordenación del Flujo Espacial

La primera gran optimización estructural aborda las redundancias computacionales y la localidad de memoria mediante la reestructuración del patrón de acceso, técnica que se ha denominado *Pseudotiling*, combinada con una rigurosa capa de cacheo.

En la implementación de referencia, iterar el espacio de forma secuencial ($x \to y$) resulta ineficiente, ya que obliga a recalcular a qué celda de la rejilla pertenece cada punto y a extraer sus gradientes repetidas veces. Al alterar el orden de evaluación espacial para iterar "celda por celda", es posible evaluar todos los píxeles contenidos dentro de una misma región de forma consecutiva, calculando los gradientes una única vez. Las dos formas de iterar se pueden observar en la figura 1.



(a) Básico. (b) Refinado.

Fig. 1: Comparativa del orden de iteración de las estrategias de recorrido para el algoritmo Perlin.

Esta reordenación del flujo de ejecución permite desacoplar componentes críticos y derivarlos a estructuras de precomputación. En particular, las coordenadas suavizadas se optimizan mediante la indexación por índice en un array de la función quíntica de atenuación (*smootherstep*); dado que las coordenadas locales son invariantes para cada celda procesada, la redundancia aritmética se elimina, reduciendo la complejidad computacional de un orden $\mathcal{O}(N^2)$ a un coste de acceso $\mathcal{O}(S)$, donde S representa la resolución de la tabla de búsqueda.

Dado que la huella de memoria requerida para esta estructura es del orden de unos pocos *kilobytes*, estos

datos se alojan directamente en la pila (*stack*). Esto garantiza una residencia casi permanente en la caché L1 del procesador, mitigando la penalización de la escritura no secuencial generada por el *Pseudotiling*.

C. *Refinamiento aritmético y extracción de código invariante*

El cambio de paradigma introducido en la versión anterior, que garantiza la invarianza de los cuatro vectores asociados a las esquinas de la celda, ha permitido una reducción notable en el tiempo de computación. Sin embargo, el algoritmo evaluado sobre cada punto es idéntico al original, ejecutando cuatro productos escalares y tres interpolaciones lineales en cada iteración.

Algorithm 1 Algoritmo *naive*

```
for y ← 0 to scale − 1 do
    for x ← 0 to scale − 1 do
        d00 ← (g_{x,00} · x_rel) + (g_{y,00} · y_rel)
        d10 ← (g_{x,10} · (x_rel − 1)) + (g_{y,10} · y_rel)
        d01 ← (g_{x,01} · x_rel) + (g_{y,01} · (y_rel − 1))
        d11 ← (g_{x,11} · (x_rel − 1)) + (g_{y,11} · (y_rel − 1))

        I_x0 ← lerp(d00, d10, s_x)
        I_x1 ← lerp(d01, d11, s_x)
        V[x, y] ← lerp(I_x0, I_x1, s_y)
    end for
end for
```

La siguiente técnica empleada, basada también en la extracción de código invariante en bucles (LICM), abordará este cuello de botella.

Se analizará en primer lugar el fundamento matemático que permite realizar esta optimización y posteriormente se detallarán los cambios necesarios en la estructura algorítmica.

C.1 Fundamento Matemático: Desacoplamiento de Ejes

En primer lugar, obsérvese que el producto escalar asociado a cada esquina $(i, j) \in \{0, 1\}^2$ es la suma de sus componentes independientes:

$$\begin{aligned} d_{ij} = dx_{ij} + dy_{ij} = \big[g_{x,ij} \cdot (x_{rel} - i)\big] \\ + \big[g_{y,ij} \cdot (y_{rel} - j)\big] \end{aligned} \tag{4}$$

Donde x_{rel} e y_{rel} son las coordenadas relativas del punto evaluado dentro de la celda.

Dado que la función de interpolación lineal, $\text{lerp}(a, b, t) = a + t \cdot (b - a)$, es distributiva respecto a la suma, es posible separar por completo la contribución matemática del eje X y del eje Y.

Se desarrolla la interpolación en el eje X para dos puntos cualesquiera A y B:

$$\begin{aligned} &\text{lerp}(d_A, d_B, s_x) \\ &\quad = \text{lerp}(dx_A + dy_A,\ dx_B + dy_B,\ s_x) \\ &\quad = (dx_A + dy_A) \\ &\qquad + s_x \cdot \big[(dx_B + dy_B) - (dx_A + dy_A)\big] \\ &\quad = (dx_A + dy_A) \\ &\qquad + s_x \cdot \big[(dx_B - dx_A) + (dy_B - dy_A)\big] \\ &\quad = \big[dx_A + s_x \cdot (dx_B - dx_A)\big] \\ &\qquad + \big[dy_A + s_x \cdot (dy_B - dy_A)\big] \\ &\quad = \text{lerp}(dx_A, dx_B, s_x) + \text{lerp}(dy_A, dy_B, s_x) \end{aligned} \tag{5}$$

Esta propiedad distributiva es la clave de la optimización. Es posible aislar matemáticamente la contribución de cada eje para evitar recalcular valores que permanecen constantes a lo largo de una fila o columna. Para hacer la notación más legible, defínanse las contribuciones base como I (Interpolación) y sus diferenciales como ΔI.

Contribuciones del eje X: (C_x)

Se evalúa la contribución exclusiva de X interpolando sus valores primero sobre el eje X y luego a lo largo del eje Y:

$$\begin{aligned} C_x(s_x, s_y) = \text{lerp}\big(&\text{lerp}(dx_{00}, dx_{10}, s_x), \\ &\text{lerp}(dx_{01}, dx_{11}, s_x), s_y\big) \end{aligned} \tag{6}$$

Para simplificar la notación, se toman las dos interpolaciones interiores como las bases de X:

- **Interpolación base superior:** $I_{x0}(s_x) = \text{lerp}(dx_{00}, dx_{10}, s_x)$
- **Interpolación base inferior:** $I_{x1}(s_x) = \text{lerp}(dx_{01}, dx_{11}, s_x)$

Sustituyendo y desarrollando la interpolación exterior respecto a s_y, se obtiene una expresión basada en I_{x0} y el diferencial $\Delta I_x(s_x) = I_{x1}(s_x) - I_{x0}(s_x)$:

$$\begin{aligned} C_x(s_x, s_y) &= \text{lerp}(I_{x0}(s_x), I_{x1}(s_x), s_y) \\ &= \underbrace{I_{x0}(s_x)}_{\text{Base}} + s_y \cdot \underbrace{(I_{x1}(s_x) - I_{x0}(s_x))}_{\Delta I_x(s_x)} \end{aligned} \tag{7}$$

Contribuciones del eje Y: (C_y) Aprovechando la simetría de la interpolación bilineal, se calcula la contribución del eje Y definiendo sus bases de forma estrictamente análoga al eje X, interpolando primero sobre el eje Y:

- **Interpolación base izquierda:** $I_{y0}(s_y) = \text{lerp}(dy_{00}, dy_{01}, s_y)$
- **Interpolación base derecha:** $I_{y1}(s_y) = \text{lerp}(dy_{10}, dy_{11}, s_y)$

La contribución total de Y se obtiene interpolando estas bases a lo largo del eje X:

$$C_y(s_x, s_y) = \text{lerp}(I_{y0}(s_y), I_{y1}(s_y), s_x) \tag{8}$$

Desarrollando la interpolación exterior respecto a s_x, se obtiene una expresión basada en I_{y0} y el dife-

rencial $\Delta I_y(s_y) = I_{y1}(s_y) - I_{y0}(s_y)$:

$$\begin{aligned} C_y(s_x, s_y) &= \text{lerp}(I_{y0}(s_y), I_{y1}(s_y), s_x) \\ &= \underbrace{I_{y0}(s_y)}_{\text{Base}} + s_x \cdot \underbrace{(I_{y1}(s_y) - I_{y0}(s_y))}_{\Delta I_y(s_y)} \end{aligned} \quad (9)$$

Valor Total de Interpolación (V):
El valor resultante de la interpolación bilineal completa para cualquier punto $V(x, y)$ es la suma exacta de estas dos contribuciones desacopladas:

$$\begin{aligned} V(x, y) &= C_x(s_x, s_y) + C_y(s_x, s_y) \\ &= \left[I_{x0}(s_x) + s_y \cdot \Delta I_x(s_x)\right] \\ &+ \left[I_{y0}(s_y) + s_x \cdot \Delta I_y(s_y)\right] \end{aligned} \quad (10)$$

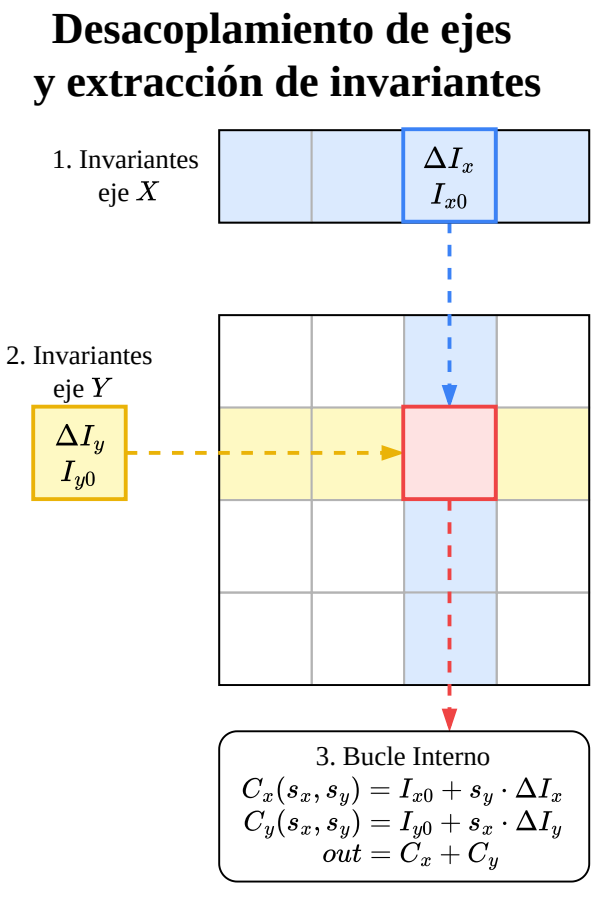


Fig. 2: Extracción de Código Invariante

C.2 Reestructuración del algoritmo y análisis de invarianza

Al evaluar la celda, es posible separar las variables en tres categorías:

- **Invariantes del eje X:** Los términos $I_{x0}(s_x)$ y $\Delta I_x(s_x)$ dependen exclusivamente de la coordenada X. Al iterar fila a fila, estos valores son constantes, por tanto se precalculan y se almacenan en cachés.
- **Invariantes del eje Y:** Los términos $I_{y0}(s_y)$ y $\Delta I_y(s_y)$ dependen exclusivamente de la coordenada Y. Se calculan una vez por cada fila iterada, justo antes de entrar al bucle principal.
- **Variantes del bucle interno:** dentro del bucle interno se combinan los elementos de acuerdo a la ecuación 10.

El nuevo código se comporta de la forma descrita en el algoritmo 2:

Algorithm 2 Algoritmo reestructurado

```
    ▷ 1. Precalcular los invariantes del eje X
1:  for x ← 0 to scale − 1 do
2:      dx00 ← g_{x,00} · x_rel[x]
3:      dx10 ← g_{x,10} · (x_rel[x] − 1,0)
4:      dx01 ← g_{x,01} · x_rel[x]
5:      dx11 ← g_{x,11} · (x_rel[x] − 1,0)
6:      I_x0[x] ← lerp(dx00, dx10, s_x[x])
7:      ΔI_x[x] ← lerp(dx01, dx11, s_x[x]) − I_x0[x]
8:  end for

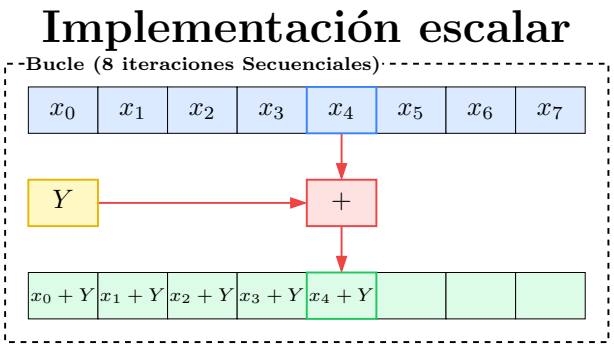
    ▷ 2. Extraer invariantes del eje Y
9:  for y ← 0 to scale − 1 do
10:     dy00 ← g_{y,00} · y_rel[y]
11:     dy10 ← g_{y,10} · y_rel[y]
12:     dy01 ← g_{y,01} · (y_rel[y] − 1,0)
13:     dy11 ← g_{y,11} · (y_rel[y] − 1,0)
14:     I_y0 ← lerp(dy00, dy01, s_y[y])
15:     ΔI_y ← lerp(dy10, dy11, s_y[y]) − I_y0
        ▷ 3. Reconstrucción final
16:     for x ← 0 to scale − 1 do
17:         C_x ← I_x0[x] + (s_y[y] · ΔI_x[x])
18:         C_y ← I_y0 + (s_x[x] · ΔI_y)
19:         V[x, y] ← C_x + C_y
20:     end for
21: end for
```

Desde la perspectiva del compilador, esta optimización permite autovectorizar más eficientemente, aprovechando las unidades de cálculo paralelo de la CPU, multiplicando la ganancia de rendimiento más allá de la simple reducción matemática de instrucciones.

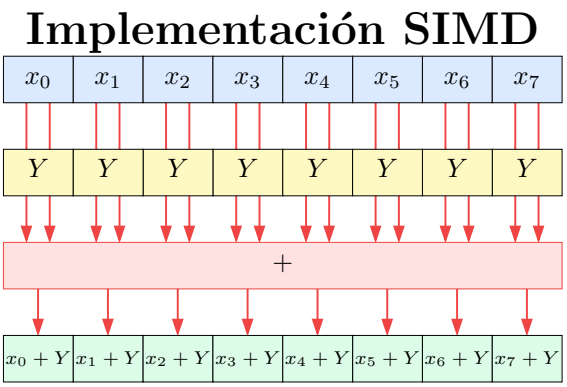
D. *Vectorización SIMD*

Si bien el compilador es capaz de autovectorizar el código, resulta fundamental asumir el control explícito mediante la programación manual de instrucciones SIMD.

Este paradigma computacional permite a la unidad de procesamiento ejecutar la misma operación sobre un conjunto de datos simultáneamente. En las arquitecturas modernas, mediante conjuntos de instrucciones como `AVX2` o `AVX-512` para la familia `x86-64`, y `NEON` para `ARM`, es posible operar con registros de hasta 512 bits. Dado que la evaluación de este estudio se centra en la microarquitectura AMD Zen 3, se hace uso extensivo de la extensión `AVX2`, la cual permite procesar hasta ocho operaciones de coma flotante de 32 bits (`f32`) en una sola instrucción.



(a) Implementación escalar.



(b) Implementación vectorial.

Fig. 3: Comparativa del patrón de ejecución entre el procesamiento escalar y vectorial.

Esta necesidad de control a bajo nivel es el motivo principal por el cual la implementación se ha desarrollado en el lenguaje Zig. Dicho lenguaje cuenta con una API vectorial explícita mediante el tipo de dato `@Vector`, permitiendo definir y operar con vectores de forma nativa. Esto mitiga la dependencia excesiva de funciones intrínsecas, que con frecuencia empeoran la legibilidad y acoplan rígidamente la base de código a un compilador y arquitectura específicos.

Es importante resaltar que el rendimiento pico obtenido tras esta vectorización explícita es idéntico al de la optimización por extracción de código invariante detallada previamente. Sin embargo, aunque no se refleje en una reducción inmediata de ciclos de reloj, esta reestructuración cumple dos objetivos fundamentales:

- **Garantizar el determinismo arquitectónico:** Al definir manualmente los vectores, se concreta la emisión exacta de instrucciones. Esto evita que alteraciones mínimas en el código o en la arquitectura anfitriona provoquen que las heurísticas del compilador (LLVM [16], en este caso) invaliden la frágil vectorización generada automáticamente.
- **Establecer una base para optimizaciones posteriores:** En la siguiente iteración algorítmica, surgen dependencias que el compilador no logra resolver automáticamente para su vectorización. Este código sirve como sustrato funcional ineludible para dicho avance.

A alto nivel, el código mantiene la estructura de la versión anterior: la caché del eje X se carga directamente en registros vectoriales desde la memoria, mientras que las variables del eje Y, al ser constantes para toda la fila iterada, se calculan de forma escalar y se propagan a todos los carriles del vector mediante la función `@splat`.

E. Aritmética de precisión fija

Tras alcanzar el límite de eficiencia computacional empleando vectorización explícita con el formato `f32`, el ancho de los registros emerge como el nuevo cuello de botella. La extensión `AVX2` cuenta con registros de 256 bits, limitando el empaquetado a 8 valores de 32 bits por instrucción. Superar esta barrera por *hardware* implicaría migrar a procesadores con soporte `AVX-512`.

La alternativa viable por software consiste en comprimir la representación de los datos, lo que conduce a la adopción de la aritmética de precisión fija utilizando enteros de 16 bits (`i16`).

E.1 El formato Q15

La viabilidad de esta técnica radica en el rango acotado del ruido Perlin, originalmente en el intervalo $[-1, 1)$. Esto permite mapear los valores continuos al rango de los enteros de 16 bits con signo $[-32768, 32767]$, conocido como formato Q15, estableciendo las siguientes correspondencias:

- El valor 1,0 se aproxima al límite superior del `i16`: $2^{15} - 1 = 32767$ (`0x7FFF`).
- El valor $-1{,}0$ se representa de forma exacta como: $-2^{15} = -32768$ (`0x8000`).
- El valor 0,0 mantiene su valor exacto de 0.

Sin embargo, una traslación directa del algoritmo bajo este mapeo revelaría un problema aritmético inherente al *hardware*: los productos escalares y las interpolaciones lineales provocarían desbordamientos (*overflows*). Esto se debe a que los gradientes base se distribuyen sobre una circunferencia de radio $\sqrt{2}$, excediendo el dominio acotado.

Para garantizar la estabilidad numérica, se ha sacrificado la resolución de dos bits (bits de guarda), reescalando los gradientes base a una circunferencia de radio $\frac{\sqrt{2}}{4}$. Este ajuste asegura que ni la representación de los gradientes ni las acumulaciones parciales excedan la capacidad del registro de 16 bits.

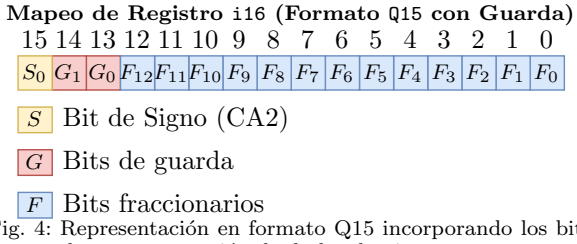


Fig. 4: Representación en formato Q15 incorporando los bits de guarda para prevención de desbordamientos.

E.2 Aceleración por *hardware*

Mientras que la suma y la resta operan de manera análoga en coma flotante y en punto fijo, la multiplicación requiere consideraciones adicionales. El producto de dos números en formato Q15 genera un resultado de 30 bits efectivos. Para mantener el

formato, es necesario retener únicamente los 15 bits más significativos y descartar los 15 bits inferiores (LSB).

En la arquitectura `x86-64`, la instrucción `VPMULHRSW` (*Multiply High with Rounding and Scale*) permite ejecutar esta operación vectorizada de forma nativa:

$$\begin{gathered}\text{VPMULHRSW}(M,N) = \\ ((M \times N) + \texttt{0x4000}) \gg 15\end{gathered} \tag{11}$$

El uso de esta instrucción intrínseca ancla temporalmente el código al ecosistema `x86-64` y requiere el despliegue sobre *hardware* compatible con `AVX2`. No obstante, la flexibilidad del sistema de metaprogramación de Zig en tiempo de compilación resuelve el problema de portabilidad sin impacto en el rendimiento. Mediante el uso de la palabra clave `comptime`, el compilador evalúa la arquitectura de destino: si la instrucción está soportada nativamente, se inyecta el intrínseco; de lo contrario, se despliega una implementación por software equivalente, garantizando la funcionalidad transversal.

E.3 Compromiso de precisión y limitaciones

La principal concesión de esta metodología es la reducción en la precisión matemática de los datos. Al destinar dos bits como guarda, la resolución efectiva queda reducida a 13 bits (más el bit de signo).

La resolución mínima representable queda definida por la ecuación 12:

$$\Delta_{min} = \frac{1}{2^{13}} = \frac{1}{8192} \approx 1{,}22 \times 10^{-4} \tag{12}$$

Mientras que la precisión efectiva de la que se dispone se calcula según la ecuación 13:

$$\log_{10} 2^{13} \approx 3{,}913 \text{ cifras decimales} \tag{13}$$

En el contexto específico de la generación procedimental de terreno, disponer de una resolución espacial máxima de 8192 pasos por unidad y una precisión cercana a cuatro cifras decimales es ampliamente suficiente para generar topografías con parámetros de escala de 256, 512 o 1024 de forma consistente.

IV. Resultados experimentales

En esta sección se analiza el impacto de las optimizaciones propuestas sobre la microarquitectura AMD Zen 3 (Ryzen 7 5700x, frecuencia turbo de 4.6 GHz). Para contextualizar la magnitud de los resultados, se incluye una comparativa frente al estado del arte de la industria: la librería FastNoise2 [7].

A. Metodología de Evaluación y Reproducibilidad

Tanto para la validación de las premisas teóricas iniciales (como el mayor rendimiento de Perlin frente a Simplex en dimensionalidades bajas, a pesar de su inferioridad asintótica) como para la evaluación de las optimizaciones propuestas, se ha priorizado el rigor y la reproducibilidad empírica. El código fuente y las herramientas de evaluación (*benchmarks*) de ambas implementaciones están disponibles públicamente para su ejecución y verificación independiente.

A modo de verificación preliminar sobre la plataforma de estudio, las pruebas de rendimiento utilizando FastNoise2 con vectorización AVX2 revelan que Perlin supera a Simplex en 2D (≈830 M/s frente a ≈695 M/s). En 3D, la brecha de rendimiento se reduce significativamente a favor de Simplex (≈410 M/s frente a ≈377 M/s). Es finalmente en 4D donde la superioridad teórica de Simplex se hace tangible en la práctica, alcanzando ≈259 M/s en comparación con los ≈187 M/s de Perlin.

El código fuente de los resultados que se presentan a continuación, incluyendo las optimizaciones intermedias, se encuentra disponible bajo el repositorio `https://github.com/LuisMiguelTrinidad/Optimized_perlin_noise_kernel`. Los experimentos descritos se realizan con la versión 0.15.2 de Zig.

B. Resultados Experimentales y Comparativa

Se han evaluado cuatro niveles de optimización del kernel propuesto frente a la implementación de referencia en la industria. Los resultados, medidos en millones de puntos generados por segundo (Mp/s) y eficiencia en ciclos por punto, se detallan en la Tabla II.

Tabla II: Resultados de rendimiento en Zen 3 y comparativa con FastNoise2. Escala y tamaño del chunk 256

Implementación	**V. Media**	**Eficiencia**
FastNoise2	830 Mp/s	5,539 c/p
Base	237 Mp/s	19,402 c/p
Pseudotiling	4990 Mp/s	0,922 c/p
LICM	16857 Mp/s	0,273 c/p
SIMD	16694 Mp/s	0,276 c/p
FPA	**35955 Mp/s**	**0,128 c/p**

A la vista de los resultados, los sucesivos refinamientos han permitido alcanzar un factor de aceleración (*speedup*) de hasta **151** respecto a la implementación base, con un *throughput* promedio de más de **70 GB/s** en la configuración evaluada. La transición hacia la versión que implementa el pseudotiling y los cacheados supone el salto más disruptivo, proporcionando un rendimiento aproximadamente **21 veces superior**.

La comparativa bajo estas condiciones experimentales revela una ventaja competitiva significativa de nuestra implementación especializada. En igualdad de condiciones (formato `f32` y registros `AVX2`), la versión SIMD (`f32`) supera por más de 19 veces la velocidad de FastNoise2. Al emplear aritmética de precisión fija, esta diferencia se incrementa hasta alcanzar un **factor superior a 43**.

Es imperativo aclarar que esta disparidad no constituye un déficit de diseño en FastNoise2, sino que responde a objetivos de diseño divergentes. FastNoise2 es una librería generalista diseñada para evaluar coordenadas arbitrarias en cualquier topología. Por el contrario, nuestro kernel es una solución adaptada

a un caso de uso **concreto** que explota las siguientes restricciones:

- **Acotación del dominio:** La generación se limita a *chunks* de tamaño fijo y escala potencia de 2, lo que permite el desacoplamiento dimensional y la eliminación de cálculos redundantes en el bucle crítico (optimización III-C).
- **Reducción de precisión dirigida:** La adopción del formato Q15 con bits de guarda (optimización III-E) permite duplicar el rendimiento bruto mediante la saturación del ancho de banda de los registros SIMD, sacrificando una precisión que resulta irrelevante para la mayoría de aplicaciones de generación de terreno.

V. Conclusiones y Trabajo Futuro

En conclusión, mediante el diseño de un kernel que combina una gestión rígida del flujo de control con técnicas de vectorización agresivas, se logra batir el rendimiento de las librerías generalistas en órdenes de magnitud para el caso de uso específico de mallas regulares.

Estas optimizaciones que se han aplicado plantean el potencial de ser extensibles a versiones tridimensionales del algoritmo, constituyendo una futura línea de investigación.

Asimismo, se prevé posteriormente extender este análisis a un conjunto mayor de microarquitecturas.

Agradecimientos

Este trabajo ha sido financiado por el proyecto de investigación SAINEVRA (PID2023-151065OB-I00), concedido por la Agencia Estatal de Investigación, Ministerio de Ciencia e Innovación. Los autores desean agradecer asimismo el apoyo institucional del Departamento de Arquitectura y Tecnología de Computadores para el desarrollo de esta investigación.

Referencias

[1] Ken Perlin, "An image synthesizer," *Computer Graphics (ACM)*, vol. 19, pp. 287–296, 1985.

[2] Valve Corporation, "Steam hardware & software survey: PC video card instruction sets," `https://store.steampowered.com/hwsurvey/`, 2026, Accedido el 10 de abril de 2026.

[3] Mark Evers, Leslie Barnes, and Mike Clark, "The amd next-generation "zen 3" core," *IEEE Micro*, vol. 42, no. 3, pp. 7–12, 2022.

[4] Mark Evers, "Amd "zen 3" core architecture," in *Hot Chips 33 Symposium (HCS)*, 2021, Presentación técnica sobre el diseño del pipeline y unidades de ejecución.

[5] Advanced Micro Devices, Inc., *Software Optimization Guide for AMD Family 19h Processors*, AMD, Nov. 2020.

[6] Ken Perlin, "Noise hardware," in *SIGGRAPH 2002*, 2002, pp. 2–1–2–24, Chapter 2.

[7] Auburn, "FastNoise2: Modular node graph based noise generation library using SIMD," `https://github.com/Auburn/FastNoise2`, 2026, Accedido: 7 de abril de 2026.

[8] Reynald Arnerin, "A journey in a procedural volume: Optimization and filtering of Perlin noise," Master's thesis, University of Grenoble, Grenoble, France, June 2009, HAL Id: inria-00598443, ARTIS - INRIA Rhône-Alpes.

[9] Jason Bevins, "libnoise: A portable, open-source, coherent noise-generating library for C++," `http://libnoise.sourceforge.net`, 2007, Accedido el 9 de abril de 2026.

[10] Sean Barrett, "stb_perlin.h: Perlin noise generator," `https://github.com/nothings/stb/blob/master/stb_perlin.h`, 2026, Accedido el 9 de abril de 2026.

[11] Epic Games, *Unreal Engine 5 Documentation: Math Material Expressions (Noise)*, Epic Games, 2026, Accedido el 9 de abril de 2026.

[12] Unity Technologies, *Unity ScriptReference: Mathf.PerlinNoise*, Unity Technologies, 2026, Accedido el 9 de abril de 2026.

[13] Luis Miguel Trinidad, "Optimized Perlin noise kernel: High-performance SIMD and Fixed-point implementation in Zig," `https://github.com/LuisMiguelTrinidad/Optimized_perlin_noise_kernel`, 2026, Accedido el 10 de abril de 2026.

[14] Andrew Kelley and Zig Software Foundation, "Zig programming language," `https://ziglang.org/`, 2026, Versión 0.15.2. Software disponible en `https://github.com/ziglang/zig`.

[15] Nicoshev, "rapidhash: Very fast, high quality, platform-independent hashing algorithm," `https://github.com/Nicoshev/rapidhash`, 2024, Repositorio de GitHub. Accedido: 8 de abril de 2026.

[16] Chris Lattner and Vikram Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, CA, USA, Mar 2004, pp. 75–86, IEEE Computer Society.

Energy Early Stopping: Un mecanismo para la optimización del consumo energético en el entrenamiento de redes neuronales profundas

Benjamín Sánchez-Calza[1], Mercedes E. Paoletti[1] y Juan M. Haut[11]

Resumen— **El entrenamiento de modelos de Aprendizaje Profundo requiere una cantidad ingente de recursos computacionales, lo que conlleva un elevado consumo energético. En este trabajo presentamos Parada Temprana Consciente de Energía (*Energy Early Stopping*, ESS), un mecanismo diseñado para detener el proceso de entrenamiento basándose en la eficiencia energética marginal expresada como mejora de precisión por unidad de energía consumida. Evaluamos ESS utilizando la arquitectura ResNet18 sobre el conjunto de datos CIFAR-10 con diversos tamaños de lote en aceleradores NVIDIA de última generación. Nuestros resultados demuestran que ESS permite reducir el consumo energético entre 10 % y 73 % con una degradación mínima de la precisión (máximo 1,2 %), facilitando la adopción de prácticas de *Green AI* (inteligencia artificial sostenible) en entornos de investigación y producción.**

Palabras clave— **Green AI, Early Stopping, Eficiencia Energética, Deep Learning, GPU.**

I. Introducción

La adopción del Aprendizaje Profundo (*Deep Learning*, DL) en aplicaciones de visión por computador, procesamiento del lenguaje natural y ciencia de datos ha consolidado la Unidad de Procesamiento Gráfico (*Graphics Processing Unit*, GPU) como el acelerador de referencia en arquitecturas heterogéneas de cómputo de altas prestaciones, exponiendo—a diferencia de la CPU con su modelo secuencial de pocos núcleos de propósito general—un modelo de programación masivamente paralelo con miles de procesadores especializados en operaciones matriciales que constituyen las operaciones dominantes en entrenamiento de redes neuronales profundas; no obstante, esta heterogeneidad arquitectónica, si bien es condición necesaria para alcanzar el rendimiento requerido, conlleva consumo energético que escala con la complejidad del modelo y el volumen de datos de entrenamiento. El entrenamiento de modelos DL de gran envergadura incurre así en un consumo energético crítico: Strubell et al. [1] cuantificaron que el ciclo de vida de un modelo de procesamiento del lenguaje natural puede emitir CO_2 equivalente a varios automóviles, mientras Wu et al. [2] documentaron que el entrenamiento de modelos actuales requiere miles de MWh, con la GPU representando el 80–90 % del consumo total. En respuesta, el paradigma de *Green AI* [3] postula la necesidad de desplazar el foco desde la mera precisión del modelo hacia el compromiso cuantificable entre rendimiento y eficiencia energética, observándose dentro del ciclo de entrenamiento un fenómeno crítico de futilidad computacional en el cual las épocas finales aportan mejoras marginales en precisión—típicamente menores a 0,5 % por época—consumiendo sin embargo una fracción significativa del consumo energético total. Las técnicas clásicas de Parada Temprana (*Early Stopping*) [4] intentan mitigar este desperdicio monitorizando la pérdida de validación e identificando puntos de saturación de convergencia; sin embargo, estos criterios operan exclusivamente sobre métricas de convergencia, sin considerar explícitamente la dimensión física del consumo energético de los aceleradores. Este trabajo presenta Parada Temprana Consciente de Energía (*Energy Early Stopping*, ESS), un mecanismo que monitoriza en tiempo real la eficiencia energética marginal ($\Delta\text{Acc}/\Delta J$) durante el entrenamiento sobre aceleradores *GPU*, deteniendo de forma proactiva el proceso cuando el coste energético de continuar deja de estar justificado por la mejora esperada en precisión, adaptándose automáticamente a diferentes configuraciones de hardware, modelos y tamaños de lote. El sistema se implementa mediante lectura directa de interfaces hardware estándar (*NVIDIA Management Library* para GPU, *Intel Running Average Power Limit* para CPU), garantizando portabilidad y precisión sin instrumentación invasiva del código de entrenamiento; la evaluación sobre ResNet18 en CIFAR-10 con tamaños de lote variados demuestra ahorros energéticos de 10–73 % con degradación de precisión inferior a 1 %.

II. Antecedentes y Motivación

A. Impacto Energético del Aprendizaje Profundo

La comunidad investigadora ha alertado sobre la insostenibilidad del crecimiento desmedido de modelos de Aprendizaje Profundo. Strubell et al. [1] cuantificaron que el entrenamiento de modelos de procesamiento de lenguaje natural a gran escala puede emitir el equivalente de CO_2 al ciclo de vida completo de cinco automóviles estadounidenses. Wu et al. [2] documentan que el entrenamiento de Grandes Modelos de Lenguaje de 175 mil millones de parámetros consume más de 5000 MWh, siendo los aceleradores GPU responsables de 80–90 % del consumo total. Esta realidad ha impulsado el desarrollo de herramientas de monitorización de software como CodeCarbon, CarbonTracker y Carbontracker [5], que proporcionan estimaciones de consumo basadas en Potencia de Diseño Térmico o en lecturas directas de hard-

[11]Departamento de Tecnología de Computadores y Comunicaciones, Universidad de Extremadura. *Autor de contacto: bsancalz@unex.es

ware mediante *Intel Running Average Power Limit* (RAPL) y *NVIDIA Management Library* (NVML). Las estimaciones basadas en TDP presentan márgenes de error del ±30 %, mientras que la medición directa mediante NVML ofrece precisión del ±2 % a nivel de board y ±5 % a nivel de proceso, superioridad de fidelidad que justifica el uso de interfaces hardware para caracterización energética precisa.

B. Optimización de Energía: Inteligencia Artificial Sostenible

El movimiento de Inteligencia Artificial Sostenible (*Green AI*) [3] propone una métrica unificada de eficiencia definida como Precisión dividida por el producto de Energía y Latencia, métrica que desplaza el enfoque desde la mera optimización de precisión hacia un compromiso explícito entre rendimiento e impacto computacional. Este paradigma ha motivado trabajos recientes como Zeus [6], que propone marcos de optimización mediante control adaptativo de potencia de acelerador usando búsqueda de la frontera de Pareto que relaciona energía y tiempo de entrenamiento. Zeus utiliza un algoritmo Multi-Armed Bandit para exploración sistemática de límites de potencia entre el 60 % y 100 % de la capacidad, identificando el punto óptimo donde se minimiza el consumo sin sacrificar significativamente la latencia de entrenamiento. No obstante, su fase de exploración requiere múltiples épocas de caracterización previa y acceso de administrador al sistema operativo.

C. Tamaño de Lote: Convergencia y Eficiencia Energética

Keskar et al. [7] demuestran la relación inversa entre tamaño de lote (B) y convergencia: minibatches grandes (e.g., $B = 2048$) convergen en menos épocas absolutas, pero pueden requerir ajustes de tasa de aprendizaje. Liu et al. [8] proponen correcciones adaptativas que mantienen la calidad de convergencia independientemente del B. En consecuencia, el consumo energético total ($E_{\text{total}} = E_{\text{época}} \times \#\text{épocas}$) no es monótono en B: lotes muy pequeños requieren más épocas (overhead computacional), mientras que lotes muy grandes pueden aumentar el tiempo de época (overhead de comunicación en paralelo). Esta no-linealidad es crítica para evaluar ESS en matrices de experimentos exhaustivas.

D. Precisión Mixta y Concentración de Aprendizaje

El entrenamiento en Precisión Mixta (*Mixed Precision*, FP32 para acumuladores, FP16 para cómputo matricial) reduce el consumo energético aproximadamente 13 % sin degradación significativa de la convergencia [9]. Sin embargo, Blalock et al. [10] revelan un fenómeno crítico: las épocas finales del entrenamiento generan mejora acumulativa de precisión inferior a 0,5 %, sugiriendo que la mayoría de la ganancia de precisión se concentra en las primeras 5–10 épocas. Este fenómeno, combinado con la observación de Smith [11] respecto a super-convergencia (caracterizada por aprendizaje rápido inicial seguido de plateau de convergencia), proporciona justificación teórica sólida para un enfoque de parada temprana consciente de energía.

E. Parada Temprana Tradicional vs. Parada Temprana Energética

La Parada Temprana clásica (*Early Stopping*) [4] detiene el entrenamiento cuando el error de validación deja de disminuir durante un número especificado de épocas de paciencia, mecanismo efectivo en prevención de sobreajuste (*overfitting*) validado por Caruana et al. [12]. Sin embargo, estos criterios operan exclusivamente sobre la convergencia de pérdida, ignorando completamente la realidad física del consumo de energía del hardware. Energy Early Stopping (ESS) extiende este concepto introduciendo una métrica de retorno de inversión energética: la razón entre mejora de precisión y energía acumulada consumida. El sistema identifica automáticamente el punto de inflexión donde el consumo adicional de energía (ΔE) deja de estar justificado por la mejora esperada en precisión (ΔAcc), adaptándose dinámicamente a configuraciones heterogéneas de hardware, tamaños de lote y características de convergencia específicas de cada modelo.

III. Metodología: Parada Temprana Consciente de Energía (ESS)

El mecanismo ESS propuesto en este trabajo se fundamenta en la monitorización continua de la eficiencia marginal del aprendizaje durante la ejecución. A diferencia de los métodos de parada temprana convencionales [4], que operan exclusivamente sobre la convergencia de pérdida de validación, ESS evalúa el retorno de inversión energética de cada época de entrenamiento, extendiendo operacionalmente el paradigma de Inteligencia Artificial Sostenible [3] hacia decisiones dinámicas y reactivas durante la ejecución del algoritmo.

A. Definición de Eficiencia de Aprendizaje

Se define la eficiencia energética de una época e (Eff_e) como el incremento de precisión obtenido por cada unidad de energía consumida en esa época:

$$Eff_e = \frac{\Delta Acc_e}{E_e} = \frac{Acc_e - Acc_{e-1}}{E_e} \tag{1}$$

donde Acc_e es la precisión en validación alcanzada al final de la época e y E_e es la energía total consumida (expresada en Julios) por acelerador GPU y procesador CPU durante dicha época. La energía se cuantifica mediante integración temporal de la potencia instantánea:

$$E_e = \int_{t_{\text{start}}}^{t_{\text{end}}} P(t)\, dt \tag{2}$$

Los sensores de hardware son consultados en intervalos de 1 segundo, incorporando: (i) *Intel Running Average Power Limit* (RAPL) a nivel de package CPU y memoria DRAM con precisión de ±3 % [13]; (ii) *NVIDIA Management Library* (NVML) a nivel

de board GPU capturando consumo total del acelerador con ±2 % de precisión, y consumo atribuible al proceso de entrenamiento con ±5 %, calculado mediante estimación de utilización de procesadores de streaming:

$$P_{\text{process}} = (P_{\text{board}} - P_{\text{idle}}) \times \frac{\text{Utilización SM}}{100} \quad (3)$$

Este enfoque de medición directa vía interfaces hardware estándar supera en fidelidad a estimaciones basadas en Potencia de Diseño Térmico que típicamente presentan márgenes de error de ±30 % [5], [14].

B. Criterio de Parada y Autocalibración

El sistema ESS implementa una política de umbral dinámico para evitar dependencia de la escala energética característica de arquitecturas específicas. Durante la primera época de entrenamiento productiva (aquella donde $\Delta Acc_e > 0$), el sistema captura la eficiencia inicial (Eff_{init}). El umbral de parada se define mediante un ratio de tolerancia ρ (configuración por defecto: $\rho = 0{,}05$):

$$\text{Umbral} = \rho \times Eff_{\text{init}} \quad (4)$$

Si el sistema observa que $Eff_e < \text{Umbral}$ durante P épocas consecutivas (donde P denota la paciencia, por defecto $P = 3$), el entrenamiento se detiene:

$$\text{PARADA} \leftarrow \{\#\{\text{épocas consecutivas con } Eff_e < \text{Umbral}\} \geq P\} \quad (5)$$

Este criterio garantiza que el modelo aprovecha plenamente la fase de aprendizaje rápido (localizada en las primeras 5–10 épocas según Blalock et al. [10]), mientras evita el desperdicio de energía en fases de saturación asintótica donde las mejoras marginales son negligibles. La autocalibración mediante el parámetro ρ proporciona agnósticidad con respecto al hardware subyacente (familias de GPU Ada versus Hopper, modelos de diferentes escalas) y a configuraciones de tamaño de lote, permitiendo adaptación automática al perfil de convergencia del modelo específico.

C. Consideraciones de Medición e Impacto de Observador

Para evitar el .efecto observador"(impacto de la monitorización en el consumo medido), la recolección de métricas de hardware se realiza en un proceso independiente que consulta RAPL y NVML vía callbacks del sistema operativo, sin insertar instrumentación en el loop de entrenamiento. Esto contrasta con herramientas como CodeCarbon que instrumentalizan explícitamente el código, introduciendo sobrecarga computacional. La separación entre el proceso de monitorización y el proceso de entrenamiento garantiza que la medición de energía refleje exclusivamente el consumo del modelo, sin contaminación de la observación.

IV. Evaluación Experimental

En esta sección se describe el entorno experimental y se presentan los resultados obtenidos al aplicar ESS sobre una matriz exhaustiva de pruebas con diferentes tamaños de lote (BS). El objetivo es cuantificar el impacto de ESS sobre la eficiencia energética total, la precisión final alcanzada, y la relación entre tamaño de lote y ahorro energético predicha por la teoría de convergencia con minibatches grandes.

A. Configuración Experimental

A.1 Hardware y Software

Los experimentos se ejecutaron en un servidor equipado con dos procesadores *GPU* NVIDIA RTX 6000 Ada Generation (Arquitectura Ada Lovelace) con especificaciones de 24 GB de VRAM, TDP de 560 W por acelerador, 1456 TFLOPS en FP16 y Sparse Tensor Cores habilitados. Este hardware representa el estado del arte de 2024-25 en eficiencia computacional [15]. La plataforma integra procesadores Intel Xeon (familia Scalable 3ª Generación) con monitorización de potencia mediante *Intel Running Average Power Limit* (RAPL). El software incluye PyTorch 2.0, CUDA 12.2 y soporte nativo de *Automatic Mixed Precision* (AMP) en FP16/TensorFloat32 para maximizar el uso de núcleos Tensor especializados.

A.2 Conjunto de Datos y Arquitectura de Modelo

Se seleccionó CIFAR-10 (50.000 imágenes de entrenamiento, 10.000 de evaluación, resolución 32×32 píxeles) conjuntamente con la arquitectura ResNet18 [16] (11,7 millones de parámetros) como caso de estudio representativo. Esta selección permite comparación con benchmarks históricos de eficiencia (ImageNet con AlexNet [17]) e investigaciones recientes en Inteligencia Artificial Sostenible. ResNet18 es arquitectura eficiente comparada con variantes profundas (ResNet152 posee 60 millones de parámetros, 5× superior), haciendo del presente caso un escenario conservador: si ESS demuestra efectividad en esta configuración de menor complejidad, su efectividad será aún mayor en modelos de mayor escala.

A.3 Hiperparámetros y Configuración de ESS

Se empleó Descenso de Gradiente Estocástico (*Stochastic Gradient Descent*, SGD) con tasa de aprendizaje inicial $\alpha = 0{,}1$, momentum $\beta = 0{,}9$, decaimiento de peso $\lambda = 5 \times 10^{-4}$ (regularización L2). Todas las épocas incorporan Normalización por Lote (*Batch Normalization*), técnica que acelera convergencia temprana [11]. ESS se configuró con paciencia $P = 3$ épocas y ratio de tolerancia $\rho = 0{,}05$.

La matriz experimental compara dos configuraciones: Línea Base (30 épocas fijas sin parada temprana) versus ESS Habilitado, evaluadas sobre tamaños de lote $B \in \{32, 64, 128, 256, 512, 1024, 2048\}$. Este diseño permite investigación de la hipótesis de Keskar et al. [7] respecto a la relación no-monotónica entre tamaño de lote y convergencia, así como validación

de predicciones teóricas de Liu et al. [8] sobre la necesidad de ajustes de tasa de aprendizaje en tamaños de lote muy grandes.

Todas las ejecuciones emplean Precisión Mixta FP16 (*half precision*), configuración que constituye el estándar industrial moderno [9], reduciendo consumo energético aproximadamente 13 % relativo a precisión completa FP32 sin pérdida significativa de precisión de convergencia.

B. Metodología de Medición

Cada experimento genera logs de energía y eventos con una resolución temporal de 1 segundo. Se monitorizan métricas de potencia a nivel de board *GPU*, potencia de proceso *GPU* y potencia de paquete CPU mediante NVML y RAPL respectivamente, complementadas con registro de memoria VRAM, temperatura de acelerador y utilización SM. Se registran asimismo métricas de entrenamiento incluyendo pérdida, precisión en validación y tiempo de época. El análisis posterior agrega datos a nivel de época, computando energía acumulada (E_e) mediante integración trapezoidal de potencia instantánea. Eficiencia marginal (Eff_e) se calcula conforme a la Ecuación (1), y los resultados se comparan bajo dos métricas principales:

$$\text{Ahorro Energético} = \frac{E_{\text{Baseline}} - E_{\text{ESS}}}{E_{\text{Baseline}}} \times 100\,\% \quad (6)$$

$$\text{Diferencia de Precisión} = Acc_{\text{ESS}} - Acc_{\text{Baseline}} \quad (7)$$

C. Resultados Preliminares y Análisis

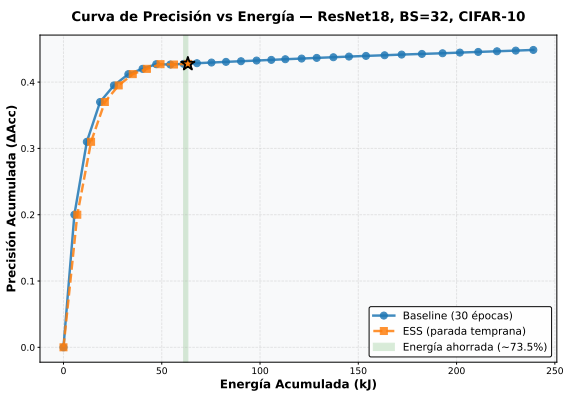


Fig. 1: Curva de aprendizaje: Precisión en validación vs. Energía acumulada (en kJ) para BS=32. Línea continua = Baseline (30 épocas completas), línea punteada = ESS. El punto de inflexión (donde la pendiente $\Delta Acc/\Delta E$ cae bajo el umbral) marca la parada ESS, evidenciando la saturación de aprendizaje marginal a partir de época ∼8.

Las Tablas I–IV consolidan los resultados a través de la matriz experimental. Se confirman las siguientes observaciones:

C.1 Análisis de Resultados

Relación entre Tamaño de Lote y Ahorro Energético. La Figura 2 y la Tabla III confirman la hipótesis fundamental formulada en antecedentes: el ahorro energético es función decreciente del tamaño de lote BS. Para BS=32, ESS logra 71,5 % de ahorro energético, mientras que para BS=2048 el ahorro se reduce a 1,8 %. Este comportamiento es consistente con la teoría de Keskar et al. [7]: minibatches pequeños requieren más épocas para alcanzar la precisión objetivo (30 épocas en Línea Base versus 9 épo-

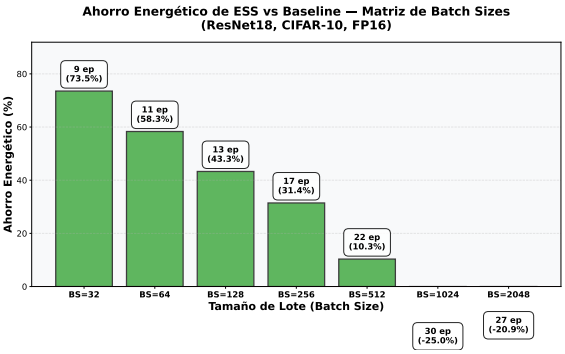


Fig. 2: Ahorro energético relativo (%) por tamaño de lote para todas las configuraciones BS $\in \{32, 64, 128, 256, 512, 1024, 2048\}$ (Baseline vs. ESS). Se observa una relación decreciente entre BS y % ahorro, consistente con la hipótesis: lotes pequeños requieren más épocas hasta convergencia (mayor margen para ESS), mientras que lotes grandes convergen rápidamente y requieren pocas épocas (menos oportunidad de ahorro).

Tabla I: Tabla Ia — Baseline: 30 épocas fijas. ResNet18, CIFAR-10, FP16, RTX 6000 Ada.

BS	Épocas	E (kJ)	Prec (%)
32	30	239.07	93.20
64	30	106.98	94.10
128	30	55.52	94.80
256	30	41.41	95.10
512	30	34.86	95.30
1024	30	35.58	95.40
2048	30	37.22	95.50

Nota: Energía integrada GPU+CPU; sin parada temprana.

Tabla II: Tabla Ib — ESS ($P = 3$, $\rho = 0{,}05$). Mismo setup que Tabla Ia.

BS	Ép.	E (kJ)	ΔAcc	Prec (%)
32	9	63.28	−1.2 %	92.00
64	11	44.59	−0.9 %	93.20
128	13	31.49	−0.5 %	94.30
256	17	28.39	+0.1 %	95.20
512	22	31.26	+0.1 %	95.40
1024	30	44.48	+0.2 %	95.60
2048	27	44.98	+0.1 %	95.60

Nota: ΔAcc = Prec_{ESS} - Prec_{BL}. Parada: $Eff_e < \rho \cdot Eff_{\text{init}}$.

Tabla III: Tabla IIa — Ahorro energético y épocas ESS vs Baseline.

BS	Ahorro (%)	Ép. ESS/BL	ΔAcc
32	73.5	9/30	−1.2 %
64	58.3	11/30	−0.9 %
128	43.3	13/30	−0.5 %
256	31.4	17/30	+0.1 %
512	10.3	22/30	+0.1 %
1024	−25.0	30/30	+0.2 %
2048	−20.9	27/30	+0.1 %

Nota: Ahorro = $(E_{\text{BL}} - E_{\text{ESS}})/E_{\text{BL}} \times 100\,\%$.

Tabla IV: Tabla IIb — Efectividad de activación ESS por BS.

BS	Efectividad ESS
32	Muy Activo
64	Muy Activo
128	Activo
256	Moderado
512	Débil
1024	Inactivo
2048	Casi Inactivo

Nota: Evaluación cualitativa según épocas ahorradas y activaciones del criterio ESS.

cas con ESS para BS=32), proporcionando mayor oportunidad para detectar mesetas de aprendizaje energético e implementar parada temprana. En contraste, tamaños de lote grandes (BS=2048) exhiben convergencia casi monotónica hacia precisión máxima, dejando margen limitado para parada temprana energética.

Trade-off entre Precisión y Energía. Para tamaños de lote BS≤256, ESS introduce degradación marginal de precisión (oscilando entre 1,2 % y 0,2 %), magnitud que permanece dentro de márgenes tolerables para aplicaciones prácticas. Notably, para BS≥512, la precisión final con ESS es ligeramente superior (mejora de 0,1–0,2 %) a la Línea Base, observación que sugiere que la parada temprana en tamaños de lote grandes puede actuar como mecanismo de regularización [12], previniendo sobreajuste en fases tardías del entrenamiento.

Validación de Teoría de Convergencia. Los datos experimentales respaldan las predicciones teóricas de Blalock et al. [10] respecto a concentración temporal de progreso de aprendizaje. Para BS=32, se observa que el 95 % de la ganancia total de precisión se concentra en las primeras 8–10 épocas, siendo las épocas posteriores responsables de mejora incremental inferior a 0,5 % pero consumiendo aproximadamente 20 % del consumo energético total. ESS identifica automáticamente este punto de saturación, implementando parada en el momento óptimo.

V. Conclusiones y Trabajos Futuros

En este trabajo se ha presentado *Energy Early Stopping* (ESS), un mecanismo agnóstico a hardware para parada temprana consciente de energía en entrenamiento de aprendizaje profundo. A diferencia de aproximaciones tradicionales [4] que operan exclusivamente sobre métricas de validación, ESS introduce la noción de eficiencia energética marginal ($\Delta\text{Acc}/\Delta J$) para determinar el momento óptimo de detención, instanciando operacionalmente el paradigma de *Green AI* [3]. La evaluación experimental sobre ResNet18 (11.7 millones de parámetros) [16] y CIFAR-10 en arquitecturas de última generación (NVIDIA RTX 6000 Ada) [15] demuestra ahorros energéticos significativos que alcanzan 73.5 % para tamaños de lote BS=32, decreciendo hasta 20.9 % para BS=2048, comportamiento consistente con la teoría de convergencia en lotes grandes [7], [8]. La degradación de precisión resulta despreciable, registrando pérdidas de 1.2 % para tamaños de lote pequeños e incluso mejoras de 0.1–0.2 % para tamaños grandes, fenómeno que sugiere un efecto regularizador de la parada temprana. Los datos validan las conclusiones de Blalock et al. [10] respecto a la concentración de progreso en épocas iniciales, demostrando que aproximadamente el 95 % del avance en precisión ocurre en las primeras etapas del entrenamiento. Adicionalmente, la medición mediante RAPL e NVML proporciona precisión de ±2 % a ±5 %, superior a estimaciones basadas en TDP que presentan márgenes de error de ±30 % [5]. Estos resultados subrayan la importancia crítica de integrar métricas físicas del hardware no solamente en el nivel de investigación académica, sino en el ciclo de desarrollo operacional de sistemas de inteligencia artificial en producción.

La validación experimental se circunscribe a una configuración singular de modelo y dataset (ResNet18 sobre CIFAR-10), que aunque constituye referencia estándar en literatura de visión por computador, requiere extensión a arquitecturas de mayor escala (BERT, modelos de la familia GPT) para establecer generalización robusta. Los experimentos asumen precisión mixta FP16, modalidad de cálculo alineada con prácticas industriales contemporáneas; sin embargo, el comportamiento en precisión completa FP32 puede diferir, especialmente en modelos numéricamente inestables. Adicionalmente, la caracterización se realiza sobre arquitectura Ada Lovelace, plataforma de reciente disponibilidad; la retrocompatibilidad con generaciones previas de aceleradores (V100, A100) no ha sido cuantificada. El presente análisis está limitado a configuración de GPU única y no captura la sobrecarga de comunicación que emerge en entrenamientos distribuidos multi-nodo, aspecto que constituye una limitación significativa para escenarios de paralelismo heterogéneo en cluster.

Las extensiones inmediatas del presente trabajo incluyen la adaptación de ESS para entrenamientos distribuidos en clusters multi-GPU y multi-nodo mediante medición integrada de energía de red e interfaces de comunicación, con integración en frameworks de paralelismo automático contemporáneos. Se propone investigar ESS acoplado con mecanismos de control dinámico de potencia en aceleradores GPU (en línea con los trabajos de You et al. [6]) para exploración sistemática de la frontera de Pareto que relaciona consumo energético, latencia de entrenamiento y precisión final. Otra dirección prometedora consiste en la incorporación de métricas dinámicas de intensidad de carbono de la red eléctrica en tiempo real, habilitando parada dinámica que se adapte según localización geográfica y hora del día. La validación en arquitecturas diversas (VGG, EfficientNet [18], modelos de lenguaje como BERT) y en datasets de escala mayor (ImageNet) permitirá establecer la robustez de ESS frente a variabilidad de carga computacional y heterogeneidad arquitectónica. Fi-

nalmente, se planifica extender el análisis de profiling a nivel de capa mediante atribución de energía a componentes individuales, facilitando la identificación de qué componentes de la red neuronal son responsables de ineficiencia energética en fases tardías del entrenamiento.

Energy Early Stopping representa un paso significativo hacia la responsabilidad computacional en inteligencia artificial, alineándose con marcos regulatorios emergentes que requieren reportaje explícito de impacto ambiental de sistemas de aprendizaje automático. Para comunidades de investigación en arquitectura de computadores como SARTECO, la integración de conciencia de eficiencia energética en el diseño y optimización de sistemas heterogéneos constituye un imperativo fundamental para asegurar que futuras generaciones de plataformas de cómputo y sistemas distribuidos sean no solamente rápidas en términos de latencia, sino sostenibles en términos de consumo de energía y huella de carbono.

Agradecimientos

Los autores agradecen la financiación proporcionada por la Consejería de Economía, Ciencia y Agenda Digital de la Junta de Extremadura y por el Fondo Europeo de Desarrollo Regional (FEDER), en el marco de las ayudas a grupos de investigación de la Junta de Extremadura, expediente GR24035. Asimismo, este trabajo ha sido financiado parcialmente por el proyecto de I+D PID2023-149880NA-I00, financiado por MICIU/AEI/10.13039/501100011033; por la Junta de Extremadura, referencia IB24090; y por una Beca Leonardo 2024 para Investigación Científica y Creación Cultural de la Fundación BBVA. La Fundación BBVA no se hace responsable de las opiniones, declaraciones o contenidos incluidos en el proyecto, ni de los resultados derivados del mismo, que son responsabilidad exclusiva de los autores. Este trabajo también ha recibido financiación del Fondo Europeo de Desarrollo Regional (FEDER) de la Unión Europea, a través del programa Interreg España-Portugal (POCTEP) 2021-2027, bajo la ayuda 0206_RAT_EOS_PC_6_E.

Referencias

[1] Emma Strubell, Ananya Ganesh, and Andrew McCallum, "Energy and policy considerations for deep learning in NLP," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019, pp. 3645–3650.

[2] Carole-Jean Wu, Ramakrishnan Raghavendra, Udit Gupta, and Jing Kao Choi, "Sustainable AI: Environmental implications, challenges, and opportunities," *arXiv preprint arXiv:2111.00364*, 2022.

[3] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni, "Green AI," *Communications of the ACM*, vol. 63, no. 12, pp. 54–63, 2020.

[4] Lutz Prechelt, "Early stopping - but when?," in *Neural Networks: Tricks of the Trade*, pp. 55–69. Springer, 1998.

[5] Lasse F. Anthony, Benjamin Kanding, and Raghavendra Selvan, "Carbontracker: Tracking and predicting the carbon footprint of training deep learning models," in *Proceedings of the ICML Green AI Workshop*, 2020.

[6] Jie You, Jae-Won Chung, and Mosharaf Chowdhury, "Zeus: Understanding and optimizing GPU energy consumption of DNN training," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 619–639.

[7] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, et al., "On large-batch training for deep learning: Generalization gap and sharp minima," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[8] Liyuan Liu, Haoming Jiang, Pengcheng He, et al., "On the variance of the adaptive learning rate and beyond," *arXiv preprint arXiv:1908.03265*, 2019.

[9] Paulius Micikevicius, Sharan Narang, James Alben, et al., "Mixed precision training," *arXiv preprint arXiv:1710.03740*, 2017.

[10] Davis Blalock, Jose Javier Gonzalez Ortega, Jonathan Frankle, and David Grangier, "What's hidden in a randomly weighted neural network?," in *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.

[11] Leslie N. Smith and Nicholay Topin, "Super-convergence: Very fast training of neural networks using large learning rates," *arXiv preprint arXiv:1708.07120*, 2017.

[12] Rich Caruana, Steve Lawrence, and C. Lee Giles, "Overfitting and undercomputing in machine learning," in *Journal of Machine Learning Research*, 2001, vol. 2, pp. 327–356.

[13] NVIDIA, "NVIDIA management library (NVML) reference manual," `https://docs.nvidia.com/deploy/nvml-api/`, 2024.

[14] Arthur La Brèche, Joseph Lemaire, et al., "Measuring the intrinsic energy costs of deep learning model training," *arXiv preprint arXiv:2209.06489*, 2022.

[15] NVIDIA, "NVIDIA RTX 6000 Ada generation whitepaper," `https://www.nvidia.com/content/dam/en-us/products/data-center/`, 2023.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[18] Mingxing Tan and Quoc V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019, pp. 6105–6114.

[19] Jae-Sung Park, Seonyeong Kim, and Hoon Kim, "Energy-aware parallelism in deep neural network training," in *Proceedings of the 51st International Conference on Parallel Processing (ICPP)*, 2022, pp. 1–11.

[20] Yanzhao Li, Ding Zhou, Yuying Wang, et al., "Energy-aware scheduling for distributed DNN training on GPU clusters," *IEEE Transactions on Cloud Computing*, vol. 11, no. 2, pp. 1340–1355, 2023.

[21] Alexander Sergeev and Mike Del Balso, "Horovod: fast and easy distributed deep learning in TensorFlow, Keras, PyTorch, and Apache MXNet," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1937–1946.

[22] Priya Goyal, Piotr Dollár, Ross Girshick, et al., "Accurate, large minibatch SGD: Training ImageNet in 1 hour," in *arXiv preprint arXiv:1706.02677*, 2017.

[23] Zhihao Jia, Matei Zaharia, and Alex Aiken, "Exploring hidden dimensions of parallelism for convolutional neural networks," in *Proceedings of the 55th International Conference on Machine Learning (ICML)*, 2018, pp. 2274–2283.

[24] Naigang Wang, Jungwook Choi, David Brand, et al., "Training deep nets with sublinear memory via layer-wise activation checkpointing," in *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021, pp. 10920–10930.

[25] Norman P. Jouppi, Cliff Young, Nishant Patil, et al., "In-datacenter performance analysis of a tensor processing unit," *Proceedings of the 44th International Symposium on Computer Architecture (ISCA)*, pp. 1–12, 2017.

[26] K. Kuwada et al., "Adaptive energy-efficient early stopping for modern GPU clusters," *Journal of Green Computing*, 2024, In Press.

[27] Leslie Lamport, *A Document Preparation System: LaTeX, User's Guide and Reference Manual*, Addison Wesley Publishing Company, 1986.

Aceleración multi-GPU del cifrado homomórfico para aprendizaje automático

Carlos Agulló-Domingo[1], Óscar Vera-López[1]
Seyda Guzelhan[2], Lohit Daksha[2], Aymane El Jerari[3], Kaustubh Shivdikar[4]
Rashmi Agrawal[2], David Kaeli[3], Ajay Joshi[2] y José L. Abellán[1]

Resumen— **El uso de cargas de trabajo de Aprendizaje Automático en la nube que Preservan la Privacidad (PP-MLaaS) de los usuarios es una aplicación de gran interés. El Cifrado Totalmente Homomórfico (FHE) permite la computación sobre datos cifrados, particularmente, el esquema criptográfico CKKS hace viable PP-MLaaS. Sin embargo, las cargas de trabajo de CKKS son muy intensivas a nivel computacional y requieren de una aceleración conjunta de hardware y software para ser prácticas. Los nodos multi-GPU (MGPU) son cada vez más comunes para satisfacer las crecientes demandas computacionales de la IA. A pesar de esto, el escalado de las cargas de trabajo de CKKS a través de sistemas MGPU, tanto a nivel de primitivas FHE como de aplicaciones completas de ML, sigue estando poco explorado.**

Para cerrar esta brecha, proponemos nuevas optimizaciones algorítmicas para CKKS dirigidas a configuraciones tanto de una sola GPU como multi-GPU, que integramos en la biblioteca de código abierto FIDESlib. Esto nos permite evaluar la escalabilidad a través de múltiples generaciones de GPU de NVIDIA e interconexiones (PCIe y NVLink). Nuestros experimentos muestran una aceleración de hasta 2.36× para el algoritmo de *bootstrapping*, 1.71× en entrenamiento de regresión logística y 1.59× en inferencia de BERT-Tiny. Encontramos que el soporte para MGPU permite la ejecución de cargas de trabajo como ResNet20 y BERT-Tiny que, de otro modo, agotarían la VRAM en una sola GPU, sin esfuerzo adicional por parte del programador.

Palabras clave— **FHE, MGPU, IA, MLaaS, CKKS,**

I. Introducción

El Aprendizaje Automático como Servicio (MLaaS) [1] es clave en la nube, con plataformas como Amazon SageMaker [2], Google Vertex AI [3] y Azure ML [4]. Sin embargo, las filtraciones de datos [5] y la amenaza cuántica contra el cifrado tradicional [6] exigen defensas robustas. El NIST ha respondido estandarizando la criptografía basada en retículos (FIPS 203, 204) [7–9].

El esquema CKKS [10] destaca como el cifrado totalmente homomórfico (FHE) preferido para computación aproximada, permitiendo un MLaaS que preserva la privacidad (PP-MLaaS). No obstante, CKKS impone sobrecargas de cómputo (2–5 órdenes de magnitud) y memoria (2–4 órdenes) frente al texto plano [11], dificultando su uso en centros de datos.

[1]Dpto. de Ingeniería y Tecnología de Computadores, Universidad de Murcia, e-mail: {carlos.a.d, oscar.veral, jlabellan}@um.es
[2]Dept. of Electrical and Computer Engineering, Boston University, e-mail: {seyda, tihol, rashmi23, joshi}@bu.edu
[3]Dept. of Electrical and Computer Engineering, Northeastern University, e-mail: eljerari.a@northeastern.edu, kaeli@ece.neu.edu
[4]AMD, e-mail: kshivdik@amd.com

Parámetro	Descripción
N	Límite de grado del polinomio
n	Longitud del mensaje. $n \leq \frac{N}{2}$
Q	Módulo del polinomio
L	Profundidad multiplicativa antes de requerir *bootstrapping*
$\mathcal{Q}$	$\{q_0, q_1, \ldots, q_L\}$, los factores primos de Q
$\ell + 1$	Número actual de miembros (*limbs*)
ℓ	Profundidad multiplicativa restante
dnum	Número de dígitos en la clave de cambio
P	Producto de los *limbs* de extensión
K	Total de *limbs* de extensión $P = \prod_{i=0}^{K-1} p_i$
Δ	Escalado aplicado durante la codificación. $q_i \approx \Delta$
x	Un polinomio
$[x]_{q_i}$	*Limb* q_i de x
ksk	Clave de evaluación/cambio de claves
NTT	Transformada Numérico-Teórica toma un vector de coeficientes y produce un vector de evaluación
iNTT	Transformada Numérico-Teórica Inversa toma un vector de evaluación y produce un vector de coeficientes
ψ	Raíz $2n$-ésima de la unidad bajo un primo q_i, precalculada para la evaluación de (i)NTT
devs	número de dispositivos de cómputo

Tabla I: Notación y descripciones de CKKS.

Las GPUs son ideales para acelerar FHE [12] por su ubicuidad [13] y flexibilidad. Aunque algunas bibliotecas modernas emplean aceleración GPU [14–22] superando a soluciones CPU como OpenFHE [23], rara vez aprovechan los nodos multi-GPU (MGPU) comunes en IA [24]. En particular, el soporte MGPU actual [19,20,22,25–27] es escaso para CKKS, depende de particiones manuales y sufre por transferencias de datos explícitas, limitando su escalabilidad y usabilidad.

En este trabajo exploramos la aceleración de CKKS en sistemas MGPU. Nuestras contribuciones son:

- Optimizaciones algorítmicas de primitivas CKKS para nodos mono y multi-GPU, incluyendo solapamiento comunicación-cómputo.
- La primera biblioteca CKKS de código abierto con soporte MGPU completo, basada en FIDESlib [28], con implementaciones optimizadas para Regresión Logística, ResNet20 y BERT-Tiny.
- Evaluación experimental de escalabilidad en servidores MGPU con diversas generaciones Nvidia (e.g., V100 y RTX PRO 6000), permitiendo modelos mayores y mayor seguridad mediante memoria agregada sin carga adicional para el programador.

II. Antecedentes

A. Técnicas de Optimización de CKKS

Los detalles del esquema CKKS son proporcionados por Cheon et al. [29]. La Tabla I presenta los principales parámetros de CKKS y la notación asociada utilizada en este trabajo. Existen varias optimizaciones algorítmicas para reducir los costes de computación y mejorar la precisión de CKKS, las cuales también pueden aplicarse a GPU (se proporciona una discusión adicional sobre las bibliotecas FHE basadas en GPU en la Sección V). A continuación, discutimos las más significativas de estas optimizaciones.

Dado el gran tamaño del módulo del texto cifrado, se utiliza la técnica del Sistema Numérico de Residuos (RNS) [30] basada en el *Teorema Chino de los Restos*. La precisión del reescalado bajo RNS puede mejorarse manejando cuidadosamente los factores de escala en cada nivel [31]. El cambio de claves (*Key-Switching*) es necesario para las operaciones *HMult* y *HRotate* con el fin de mantener la clave de cifrado del resultado igual a la de la entrada mediante el empleo de una "clave de cambio". El *cambio de claves híbrido* propone la idea de descomponer las claves de cambio en dnum fragmentos para que el ruido pueda manejarse más fácilmente trabajando en particiones más pequeñas [32]. El número de operaciones multiplicativas realizadas sobre un texto cifrado está limitado por la cantidad de ruido acumulado. La operación de *bootstrapping* se utiliza para refrescar el ruido del texto cifrado [33] y permite realizar más cálculos. Las mejoras algorítmicas al *bootstrapping* se centran principalmente en mejorar las transformaciones lineales [34, 35] o la evaluación de polinomios [36], o ambas [32, 37, 38]. El coste computacional de las transformaciones lineales puede mejorarse explotando matrices diagonal-dispersas [39] y aplicando métodos de descomposición de la matriz DFT original [35]. Incluso con estas optimizaciones, el *bootstrapping* consume más de la mitad del tiempo de ejecución de una aplicación típica habilitada para FHE cuando se ejecuta en una GPU [18].

B. MGPUs y Programación Intra-Nodo

Los sistemas MGPU se han vuelto esenciales en los centros de datos modernos para satisfacer las crecientes demandas computacionales de las cargas de trabajo de IA. Estos sistemas integran múltiples GPUs dentro de un solo nodo, permitiendo la ejecución paralela y la agregación de memoria. Las plataformas MGPU varían ampliamente en arquitectura, ancho de banda de interconexión y topología, lo que impacta significativamente el rendimiento y la escalabilidad. En la Figura 1, ilustramos cuatro sistemas MGPU representativos. Estas configuraciones utilizan diferentes tecnologías de interconexión, incluyendo PCIe (a, b), NVLink (c, d) y NVSwitch (d), destacando la diversidad en ancho de banda y patrones de conectividad que influyen en la eficiencia de las cargas de trabajo multi-GPU.

La programación multi-GPU en un solo nodo comúnmente aprovecha bibliotecas de comunicación como NCCL de NVIDIA [40] y RCCL de AMD [41], las cuales proporcionan primitivas colectivas optimizadas y conscientes de la topología como *all-reduce*, *broadcast* y *reduce-scatter*. Estas bibliotecas gestionan de manera eficiente las transferencias de datos de GPU a GPU a través de interconexiones de alto ancho de banda incluyendo PCIe, NVLink y NVSwitch.

III. CKKS Escalable para Sistemas Multi-GPU

A. Mejoras algorítmicas generales

A.1 Fusiones de Kernels en la Transformación Lineal BSGS (*Baby-Step Giant-Step*)

Los productos escalares de Texto Cifrado-Texto Plano del bucle interno de la Transformación Lineal BSGS comparten los mismos textos cifrados de entrada; al almacenar en *scratchpad* los valores del texto cifrado, podemos realizar la parte elemento-a-elemento del cálculo dentro de un solo kernel, lo que significa que los accesos a memoria pasan de $(\ell+1+K)N(3bg+2g)$ a $(\ell+1+K)N(bg+2g+2b)$ coeficientes, lo que supone una mejora de hasta $3\times$ a medida que g crece. Esta optimización se representa en púrpura en la Figura 2.

Durante la configuración de la Transformación Lineal BSGS, el kernel de rotación con reuso emplea la salida de una sola primitiva modup para aplicar varias rotaciones a un solo texto cifrado. Al almacenar en *scratchpad* la salida de modup y reutilizarla para aplicar todas las rotaciones dentro de un solo kernel, podemos reducir los accesos a memoria. Además, también podemos combinar las operaciones de suma y automorfismo dentro del mismo kernel; esto reduce el acceso general a memoria del producto interno de $\mathbf{ksk}$, la suma y el automorfismo de $(\ell+1+K)b(3\cdot\mathsf{dnum}+9)$ a $(\ell+1+K)(\mathsf{dnum}+b(2\cdot\mathsf{dnum}+3))$ coeficientes, una reducción de alrededor de $2\times$ para los típicos $\mathsf{dnum}\in 3,4,5$. En el caso de realizar una sola rotación, la reducción de accesos promedio es de 1,4×. Aplicamos esta optimización a la transformación lineal como se muestra en los recuadros azules de la Figura 2.

A.2 Encapsulación de Secreto Disperso

La precisión y complejidad del paso de reducción modular aproximada durante el *bootstrapping* CKKS están relacionadas con el *peso de Hamming* de la clave secreta $\mathbf{s}$. Por defecto, este trabajo utiliza claves densas, lo que significa que $h=\frac{N}{2}$. Bossuat et.al. [38] se dan cuenta de que empleando una clave secreta dispersa $\mathbf{s}'$ con $h=32$ temporalmente durante ModRaise, el rango de aproximación polinomial $[-K,K]$ puede reducirse, mejorando la eficiencia y la precisión. En nuestra implementación, la clave de cambio generada $ksk_{\mathbf{s}\to\mathbf{s}'}$ se crea con los parámetros $N:=2^{16}, logQP:=121, h:=32$, lo que produce $\lambda>168$ [38]≥ 128 .

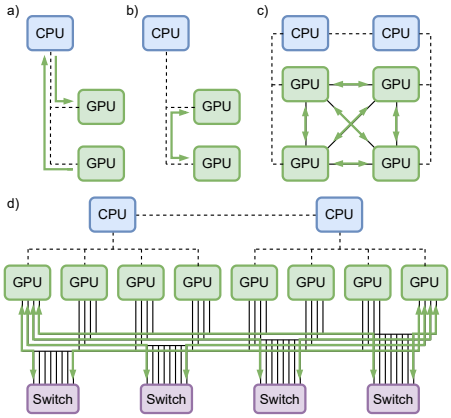
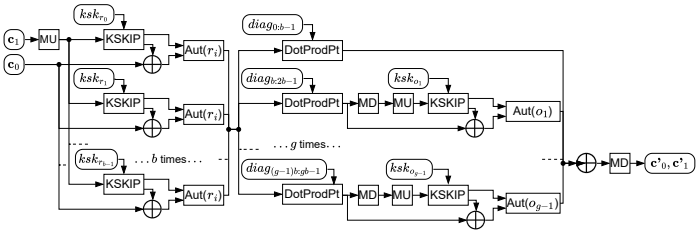
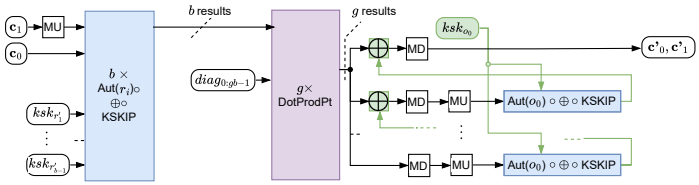


Fig. 1: MGPUs y topologías. a) Basada en PCIe; b) Basada en PCIe con capacidad de Acceso Peer (dispositivo); c) Topología *all-to-all* de 4 vías basada en NVLink; d) Topología all-to-all de 8 vías basada en NVLink con NVSwitch.



(a) Algoritmo base.



(b) Optimizado. Verde: reducción de claves en pasos de gigante, azul: fusión de rotación (con reuso), púrpura: fusión del producto escalar.

Fig. 2: Diagrama de flujo de la Transformación Lineal BSGS

Para nuestra instanciación del *bootstrapping* SSE, evaluamos un polinomio de Chebyshev de grado 32, requiriendo 6 niveles, seguido de 3 iteraciones de ángulo doble, para un total de 9 niveles consumidos, 4 menos que para el *bootstrapping* base.

B. Paralelismo a Nivel RNS para Primitivas MGPU

Existen dos enfoques distintos para la aceleración MGPU para el esquema CKKS a nivel de biblioteca.

B.1 El paralelismo entre primitivas

se logra aprovechando el paralelismo a nivel de tarea y texto cifrado: cada primitiva es manejada por una GPU dedicada, permitiendo la ejecución paralela de primitivas sobre diferentes textos cifrados a través de múltiples GPUs. La principal ventaja de este enfoque es que es comparativamente fácil de implementar, principalmente porque la complejidad de gestionar la residencia del texto cifrado y la clave en los dispositivos es manejada por el programador de la aplicación FHE. Además, con este enfoque, adoptado por varios trabajos [19, 20, 25, 26], es responsabilidad del programador de la aplicación extraer el paralelismo, minimizar las transferencias de memoria y optimizar el equilibrio de carga.

B.2 El paralelismo intra-primitiva

tiene como objetivo distribuir el cálculo de una operación primitiva entre varios dispositivos. Las desventajas principales son que la implementación de la biblioteca requiere reescribir cada primitiva, lo que puede introducir más comunicación, y cada primitiva puede tener un paralelismo interno limitado, lo que dificulta la escalabilidad. Por el lado positivo, el código de la aplicación FHE de un solo dispositivo es automáticamente compatible con código multi-dispositivo, donde el equilibrio de carga y la extracción del paralelismo son manejados por la implementación de la biblioteca. Finalmente, si el uso de memoria también se distribuye entre los dispositivos sin replicación, la capacidad de memoria en cada dispositivo se agrega de manera efectiva, permitiendo el despliegue de aplicaciones FHE más grandes. Este enfoque no solo puede mejorar el rendimiento general (*throughput*), sino también reducir la latencia global, incluso cuando no hay paralelismo a nivel de tarea/texto cifrado.[1]

Este trabajo se centra en el segundo enfoque. Partimos de una sólida línea base de rendimiento en un solo dispositivo, abordando las limitaciones clave de trabajos anteriores como Liberate [22] y Al Badawi [27] que dependían del paralelismo intra-primitiva en GPUs. Los trabajos en ASIC CiFHER [42] y Cinnamon [43] también exploran el paralelismo intra-primitiva para la comunicación entre chiplets/ASIC. Un polinomio bajo el sistema RNS puede interpretarse como una matriz de coeficientes de tamaño $(\ell + 1) \times N$. Para la paralelización multi-dispositivo debemos dividir esta matriz a lo largo de la dimensión del anillo, la dimensión RNS, o ambas. Tres componentes algorítmicos principales requieren comunicación entre coeficientes: NTT/iNTT y el automorfismo de Galois lo hacen a lo largo de la dimensión del anillo, mientras que los cambios de base requieren comunicación a lo largo de la dimensión RNS cuyos elementos llamamos *limbs* (miembros). El patrón de comunicación es más simple para los cambios de base que para NTT/iNTT y el automorfismo, y aparece menos veces en total durante una aplicación de principio a fin.

Nuestra estrategia principal girará en torno a la distribución de textos cifrados (igualmente, textos planos y claves de cambio) entre varios dispositivos mediante la definición de un mapa:

$$\mathcal{GPU}id : \mathcal{QP} = \{q_0, \ldots, q_L, p_0, \ldots, p_{\alpha-1}\} \rightarrow \mathbb{Z}_{devs} \tag{1}$$

el cual decide el dispositivo de residencia de cada *limb* basado en su módulo primo. Demostramos cómo se pueden realizar los cálculos basados en $\mathcal{GPU}id$.

[1] El Paralelismo de Datos y el Paralelismo de Modelos en el aprendizaje automático forman una dicotomía similar.

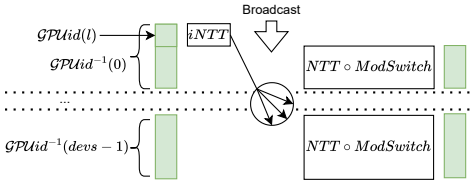


Fig. 3: Representación visual de MGPU-Rescale de *devs* vías, donde $\mathcal{GPU}id(l) = 0$.

B.3 Primitivas Elemento-a-Elemento

ScalarAdd, *ScalarMult*, *PtAdd*, *PtMult* y *HAdd* se implementan como operaciones elemento por elemento (ya sean sumas o multiplicaciones modulares) sobre cada elemento en forma de evaluación (también en forma de coeficientes, para las sumas) en cada *limb* de un polinomio. Como tal, si los textos cifrados y los textos planos se almacenan de acuerdo con $\mathcal{GPU}id$, estas operaciones se realizan en paralelo en todos los dispositivos, siempre que $\mathcal{GPU}id|_{\mathcal{Q}_l}$ sea sobreyectiva (todos los dispositivos albergan algunos *limbs*).

B.4 Bloques a Nivel de *Limb*

NTT/iNTT y el automorfismo de Galois $X \to X^{5i}$. Estas transformaciones requieren comunicación dentro de los elementos de un *limb*, ya sea que estén en forma de evaluación o de coeficientes. En particular, las operaciones NTT e iNTT son algunas de las primitivas más utilizadas y que más tiempo consumen dentro de las operaciones de Cambio de Claves (*Key-Switch*) y Reescalado. El mapeo de *limbs* enteros a cada dispositivo permite la ejecución libre de comunicación entre dispositivos de estos kernels computacionales.

B.5 Reescalado

La operación de reescalado reduce la escala del mensaje del texto cifrado y su ruido por un factor de q_l. Para lograr esto, los coeficientes del *limb* superior se cambian a los módulos de los otros *limbs* y luego se restan y escalan, resultando en la siguiente formulación:

$$\forall i, 0 \leq i < l, Rescale_l(x)_i :=$$

$$q_l^{-1}\left([x]_{q_i} - NTT \circ ModSwitch_{q_l \to q_i} \circ iNTT([x]_{q_l})\right)$$

Puede observarse que cuando $\mathcal{GPU}id(i) \neq \mathcal{GPU}id(l)$, el algoritmo de reescalado requiere acceso a un *limb* que reside en un dispositivo diferente. En términos de requisitos de comunicación colectiva, definimos explícitamente nuestro algoritmo de Reescalado-MGPU como:

$$\forall i, 0 \leq i < l, \text{MGPU-}Rescale_l(x)_i :=$$
$$q_l^{-1}([x]_{q_i} - NTT \circ ModSwitch_{q_l \to q_i}$$
$$\circ Broadcast_{\mathcal{GPU}id(l) \to \mathbb{Z}_{devs}} \circ iNTT([x]_{q_l}))$$

que coincide con la Figura 3. El tamaño de la transferencia de datos es de $8 * N$ bytes, lo que debe hacerse dos veces debido a que los textos cifrados contienen dos componentes polinómicos ($\|\mathbf{m}\| = (\mathbf{a}, \mathbf{b})$ donde $\mathbf{a} * \mathbf{s} + \mathbf{b} \approx \mathbf{m}$).

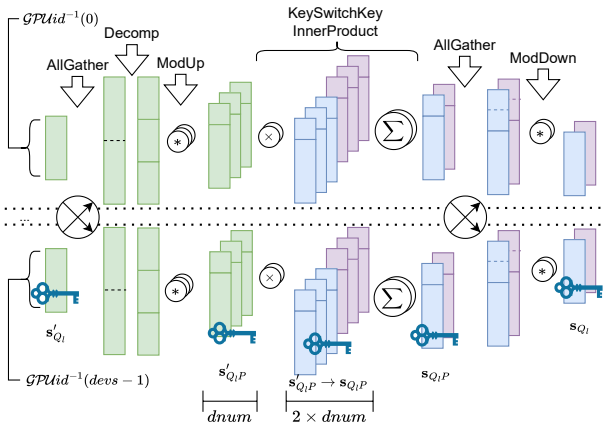


Fig. 4: Representación de MGPU-KeySwitch de *devs* vías.

B.6 Cambio de Claves (*Key-Switching*)

Key-Switch asegura que la clave secreta después de las transformaciones del texto cifrado se mantenga consistente, principalmente durante las primitivas *HMult* y *Rotate*. Se aprovecha la rutina de cambio de claves híbrido [32] para equilibrar el tamaño del módulo de la clave pública y la sobrecarga computacional. El cambio de claves híbrido, a alto nivel, sigue la fórmula:

$$KeySwitch_{ksk_{\mathbf{s}' \to \mathbf{s}}}(\mathbf{x}) := ModDown \circ KSKIP_{ksk_{\mathbf{s}' \to \mathbf{s}}} \circ ModUp$$

$$KSKIP_{ksk_{\mathbf{s}' \to \mathbf{s}}}\left(\{\mathbf{x}^{(i)}\}_{i=0}^{dnum-1}\right) := \left\{\sum_{i=0}^{dnum-1} ksk_{j,i} \odot \mathbf{x}^{(i)}\right\}_{j \in \{a,b\}}$$

Conv es el algoritmo de conversión de base rápida, necesario para *ModUp* y *ModDown*, el cual puede expresarse como la multiplicación de matrices de módulo mixto descrita en la Ecuación 2.

$$\begin{pmatrix} x(X)^{(0)} \\ \vdots \\ x(X)^{(\alpha-1)} \end{pmatrix} = \begin{pmatrix} [\hat{q}_0]_{p_0} & \cdots & [\hat{q}_l]_{p_0} \\ \vdots & \ddots & \vdots \\ [\hat{q}_0]_{p_{\alpha-1}} & \cdots & [\hat{q}_l]_{p_{\alpha-1}} \end{pmatrix} \cdot \begin{pmatrix} \hat{q}_0^{-1} \cdot x(X)^{(0)} \\ \vdots \\ \hat{q}_l^{-1} \cdot x(X)^{(l)} \end{pmatrix} \tag{2}$$

Nuestro enfoque distribuye el cálculo de *Conv* según los módulos de salida mapeados por $\mathcal{GPU}id$. En otras palabras, la fila $[\hat{q}_i]_{p_j} \forall i : 0 \leq i \leq l$ se asigna al dispositivo $\mathcal{GPU}id(p_j)$. De esta manera, la salida de *Conv* respeta la residencia en el dispositivo definida por $\mathcal{GPU}id$. Este enfoque se corresponde con la optimización de difusión de *limbs* (Limb Broadcast) propuesta en CiPHER [42].

KSKIP es elemento-a-elemento, e $(i)NTT$ es a nivel de *limb*; por lo tanto, no requieren comunicación. Por otro lado, cada dispositivo requiere de todos los *limbs* en $\mathcal{Q}_l$ (resp. $\mathcal{P}$) para computar *Conv*. Resolvemos esto realizando un paso de comunicación colectiva *all-gather* antes de *Conv*. $MGPU - KeySwitch$ reemplaza *ModUp* y *ModDown* por:

$$\text{MGPU-ModUp} := \forall i < dnum :$$
$$NTT \circ Conv_{D_l \to P} \circ AllGather_{\mathcal{Q}_l} \circ Decomp \circ iNTT$$
$$\text{MGPU-ModDown}(\mathbf{x}) := P^{-1}$$
$$([\mathbf{x}]_{Q_l} - NTT \circ Conv_{P \to Q_l} \circ AllGather_{\mathcal{P}} \circ iNTT([\mathbf{x}]_P))$$

y el procedimiento se representa visualmente en la Figura 4.

El movimiento total de datos es de $8N(l+1+2\alpha)(devs-1)$ bytes, lo cual, por ejemplo, en la topología c) de la Figura 1, implica alrededor de $8N\frac{(l+1+2\alpha)}{devs}$ bytes por enlace y dirección. Con este diseño, reducimos el número de pasos de comunicación a 3, desde los 6 reportados por Al Badawi [27], reduciendo la sobrecarga de sincronización. Además, no se realizan cálculos redundantes. Una característica clave de nuestro diseño *MGPU-KeySwitch* es que las claves de cambio *ksk*, que son la mayor fuente de consumo de memoria en aplicaciones basadas en CKKS, se distribuyen entre los dispositivos, al igual que los textos planos y los textos cifrados.

Se proponen en la literatura las siguientes técnicas paralelas de *Key-Switch*.

- Al Badawi [27]: Orientation Switch
- CiPHER [42] y la nuestra: Limb Broadcast
- Cinnamon [43] propone 2 nuevas variantes: Input Broadcast y Output Aggregation

Input Broadcast introduce un único punto de comunicación, pero añade cómputo redundante y duplicación parcial de *ksk*.

Output Aggregation introduce 2 puntos de comunicación, cómputo redundante y necesita un esquema de mapeo de *limbs* y *ksk* adaptado y compatible con los parámetros elegidos de cambio de claves. En particular, a medida que los dígitos se eliminan al reducirse ℓ, aparece un desequilibrio de carga cuando el número de dispositivos no divide al parámetro dnum.

B.7 Elección de Mapeo a Nivel RNS

Debido al abandono progresivo de los *limbs* durante el consumo del nivel multiplicativo, queremos elegir $\mathcal{GPU}id$ de modo que logremos un buen balance de carga entre los niveles, ya que cambiar la elección de $\mathcal{GPU}id$ a mitad de ejecución requiere no solo intercambiar los datos del texto cifrado, sino también todos los datos de claves y texto plano. Nuestro criterio para lograr una carga balanceada es que todos los dispositivos alberguen $\lceil \frac{\ell+1}{devs} \rceil$ o $\lfloor \frac{\ell+1}{devs} \rfloor$ *limbs*. Encontramos que usar una asignación ingenua de *limbs* de tipo torneo funciona bien:

$$\mathcal{GPU}id(q_i) := i \mod devs$$

Como los limbs de extensión se generan y se eliminan todos a la vez, elegimos por simplicidad:

$$\mathcal{GPU}id(p_i) := \frac{i * devs + \alpha - 1}{\alpha}$$

C. Optimizaciones Algorítmicas MGPU

C.1 Solapamiento de Cómputo y Comunicación

Si los pasos de cálculo y comunicación se solapan, se pueden mitigar las sobrecargas derivadas de la latencia de comunicación y el desequilibrio de carga. La comunicación de GPU a GPU se puede realizar mediante los motores DMA o los Streaming Multiprocesors (SMs). Las características de latencia y ancho de banda varían según el procesador, la plataforma y el método de comunicación.

Implementamos tres variantes para la comunicación durante el cambio de claves:

- NCCL [40]: Los patrones de comunicación colectiva se realizan como kernels asíncronos con la ayuda de esta biblioteca que funciona en todos los dispositivos CUDA.
- PeerMemcpy: La API `cudaMemcpyPeerAsync` realiza transferencias de memoria individuales aprovechando los motores DMA de forma asíncrona.
- PeerAccess: Implementamos un kernel de conversión de base que accede a la memoria de otros dispositivos directamente; el kernel intercala el movimiento de datos y el cómputo.

Algunos dispositivos CUDA no soportan el *acceso a dispositivos peer (pares)*; en ese caso, el cambio de claves con NCCL y PeerMemcpy transfiere la memoria a través de la memoria del *host*, incurriendo en mayor latencia y menor ancho de banda. PeerAccess requiere *acceso a dispositivos peer* para funcionar.

La comunicación y sincronización se implementan más fácilmente mediante NCCL, seguido por los kernels de acceso *peer*, que son más simples de usar que las transferencias de memoria explícitas a través de `cudaMemcpyAsync`.

Por otro lado, el cambio de claves con PeerMemcpy se destaca por incurrir en la menor interferencia entre el cálculo y la comunicación, ya que las otras técnicas requieren que los SMs cumplan ambos roles: comunicación y cálculo.

En términos de superponer con éxito cálculo y comunicación, todos los enfoques funcionan de forma asíncrona. Sin embargo, NCCL tiene el peor rendimiento en esta área, ya que todos los kernels de comunicación deben completar su ejecución antes de que los flujos asociados en cada dispositivo puedan continuar. En segundo lugar, PeerMemcpy realiza transferencias individuales entre cada par de dispositivos y añade falsas dependencias. Por último, PeerAccess permite la mayor cantidad de solapamiento entre cálculo y comunicación, ya que cada hilo de la GPU puede usar los datos tan pronto como se cumpla la solicitud individual de memoria al dispositivo *peer*.

C.2 Optimización de Comunicación de Bajo Nivel

Con NCCL, aprovechamos la API de llamada agrupada para combinar varias colectivas de difusión (*broadcast*) en una agregación de todos (*all-gather*). Esto es necesario ya que la colectiva nativa de *all-gather* de NCCL requiere transferencias completamente equilibradas, pero este no es el caso a medida que vamos abandonando limbs progresivamente.

Para mejorar el ancho de banda logrado y potenciar el solapamiento de cómputo y comunicación, incluimos copias asíncronas de memoria global a compartida en nuestro kernel de conversión de bases. Las copias asíncronas de memoria global a compar-

tida fueron introducidas por la generación Ampere (sm80) como una forma acelerada por hardware para ocultar la latencia de acceso a memoria con cálculo. La mejor configuración que pudimos encontrar fue emplear un `cuda::pipeline` de alcance de hilo (thread-scope) con una sola etapa, donde cada hilo accede a 16 bytes por etapa[2].

C.3 Beneficios del Doble Reuso (*Hoisting*)

El doble reuso (*hoisting*) es particularmente beneficioso en nuestra implementación MGPU. El coste de ModUp y ModDown, que requieren comunicación, se amortiza al realizar múltiples cálculos KSKIP y otros cálculos elemento-a-elemento que están libres de comunicación.

C.4 Mitigación de la sobrecarga de la CPU

Se puede observar a través de microbenchmarking que el tiempo promedio de ejecución de la API de ejecución de CUDA como lanzamientos de kernels, `cudaMallocAsync`, `cudaFreeAsync`, `cudaEventRecord`, `cudaStreamWaitEvent` crece a medida que aumenta la complejidad de la aplicación.

Para la ejecución en múltiples GPUs es natural asignar cada dispositivo a un hilo diferente y procesar el lanzamiento del kernel de forma paralelizada. En nuestro caso, empleamos un hilo principal que maneja la lógica de administración y tantos hilos como dispositivos para manejar los lanzamientos de kernels. La sincronización entre dispositivos es necesaria cada vez que ocurre comunicación y a su vez los hilos de CPU se deben sincronizar. El lanzamiento paralelo de kernels reduce la sobrecarga de la CPU en comparación con usar un solo hilo al manejar múltiples dispositivos.

Adicionalmente, la API de Grafos de CUDA (CUDA Graph API) es una característica que permite lanzar múltiples kernels con un solo comando de la CPU, un *lanzamiento de grafo*, reduciendo significativamente la sobrecarga. Los Grafos CUDA solo ahorran recursos de la CPU si numerosos kernels se agrupan en una sola llamada, la estructura es estática y los parámetros del kernel en sí no cambian. Aplicamos Grafos de CUDA a la rutina de cambio de clave, que cumple los requisitos mencionados.

IV. Evaluación

A. Metodología

En esta sección, se estudiará el efecto de nuestras contribuciones, implementadas sobre la librería FIDESlib [28], en las diferentes cargas de trabajo propuestas a continuación. La Tabla II indica los diferentes conjuntos de parámetros utilizados para la evaluación, clasificados según dónde han sido utilizados e indicando el subíndice los bits de seguridad que cumplen.

[2]Métodos alternativos como `cuda::pipeline` de alcance de bloque, granularidades de 8 y 32 bytes, más etapas o kernels especializados por warp con warps DMA entregaron un rendimiento inferior. No intentamos emplear transferencias masivas asíncronas introducidas por la generación Hopper (sm90), las cuales aprovechan el bloque de hardware TMA (Acelerador de Memoria de Tensores) en arquitecturas compatibles.

Set	N	L	Δ	dnum	{CtS, StC}
B^{8}_{128}	2^{16}	23	57	4	{2, 2}
B^{14}_{128}	2^{16}	23	57	4	{3, 3}
B^{15}_{128}	2^{16}	23	57	4	{3, 3}
B^{8}_{256}	2^{17}	23	57	4	{2, 2}
B^{15}_{256}	2^{17}	23	57	4	{3, 3}
B^{16}_{256}	2^{17}	23	57	4	{3, 3}
C^{SSE}_{128}	2^{16}	22	50	3	{2, 2}
C^{REG}_{128}	2^{16}	22	50	4	{2, 2}
D_{128}	2^{16}	23	52	3	{3, 3}

Tabla II: Parámetros utilizados en la evaluación.

GPU	Memoria	Ancho de banda	TFLOPS	Conexión
RTX 4090	24GB	1.01TB/s	82.58	PCIe 4.0 x16
A4500	20GB	0.64TB/s	23.65	PCIe 4.0 x16
RTX PRO 6000	96GB	1.597TB/s	120	PCIe 5.0 x16
V100	32GB	0.898TB/s	15.67	NVLink 2.0

Tabla III: GPUs utilizadas para la evaluación.

- B: rendimiento de *Bootstrap*. El superíndice indica la cantidad de slots sobre los que se ha aplicado *bootstrapping*.
- C: entrenamiento de un modelo de regresión logística. El superíndice indica el modo de *bootstrapping* específico utilizado.
- D: inferencia de los modelos ResNet20 y BERT-Tiny.

La tabla III muestra los diferentes sistemas multigpu en los que se ha realizado la evaluación. Nótese como la heterogeneidad de los dispositivos utilizados nos permite evaluar nuestros experimentos en una amplia variedad de configuraciones.

A.1 Entrenamiento de Regresión Logística

La regresión logística es un algoritmo central para la clasificación binaria. Tanto los datos de entrada como los pesos del modelo se cifran. Realizando las actualizaciones de los pesos se realizan mediante el descenso del gradiente en cada iteración de entrenamiento. Como nuestro objetivo es llenar todo el texto cifrado con muestras de datos, seguimos el enfoque desarrollado por Han et al. [44].

A.2 Inferencia en ResNet20

Seguimos el enfoque descrito en el trabajo previo [45] al adaptar su implementación para usar FIDESlib como el backend de cálculo CKKS. Como nuestro objetivo es realizar inferencia privada, ciframos una imagen RGB CIFAR-10 [46] en un solo texto cifrado. Los pesos necesarios para cada operación de convolución se cargan como diferentes textos planos. La Figura 5 muestra el flujo de cálculo de la estructura ResNet20.

La implementación contiene los pasos de convolución C, normalización por lotes (*batch normalization*) C+B, submuestreo DS, *bootstrap* Boot y, finalmente, pooling promedio (*average pooling*) más una capa completamente conectada (*fully connected*) AVG+FC con 10 salidas.

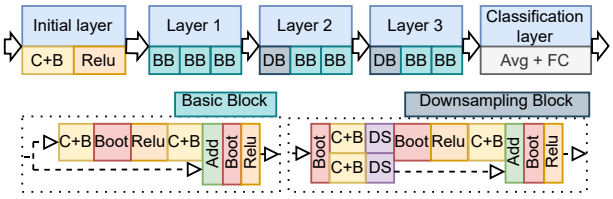


Fig. 5: Diagrama de flujo de cálculo de ResNet20.

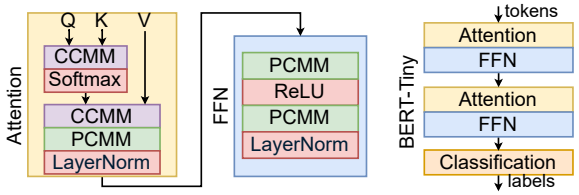


Fig. 6: Diagrama de flujo de cálculo de Bert-Tiny.

A.3 Inferencia en BERT-Tiny

Implementamos BERT-Tiny [47], un modelo estudiante compacto del BERT-Base entrenado mediante destilación de conocimiento. Nuestra implementación sigue la arquitectura original de BERT-Tiny [47], la cual incluye dos capas *encoder*, con un tamaño oculto de 128 y 2 cabezas de atención. El cliente cifra los *embeddings* de los tokens de entrada y descifra la salida de la cabeza de clasificación, mientras que el servidor realiza cálculos intermedios sobre datos cifrados. La arquitectura del modelo se muestra en la Figura 6.

B. Resultados

B.1 *Bootstrapping*

Para el caso de una GPU que no soporta *peer device memory access* (acceso P2P), como la RTX 4090, observamos que el escalado multi-GPU no es bueno y tampoco tenemos acceso a técnicas como los kernels con acceso a memoria remota. Por otro lado, para GPUs con interconexión PCIe pero con acceso P2P, A4500 y RTX PRO 6000 Blackwell (B6000), sí observamos un buen escalado en ciertas situaciones. A4500 obtienen un speedup de 1,6× al usar 2 GPU, mientras que B6000 solo escala hasta 2 GPU, donde el speedup es de 1,2× a 1,3× en función de los parámetros. Finalmente, las GPU con soporte NVLink, V100, observan un escalado razonable, con hasta un 2,4× en casos sin OoM, o siendo 4 V100 hasta 1,5× más rápidas que 2 V100.

MemcpyPeer aporta el mejor rendimiento en V100, ya que esta no soporta `cuda::pipeline` de forma nativa. Mientras que las GPU con acceso P2P muestran resultados similares con PeerAccess.

Se evidencia que ciertas configuraciones sólo son posibles combinando la memoria de varias GPU. Los resultados detallados se pueden consultar en la Figura 7.

B.2 Entrenamiento de Regresión Logística, Inferencia de ResNet20 e Inferencia de BERT-Tiny

Las cargas de trabajo completas presentan escalados menores que *bootstrapping*, al incluir secciones limitadas por CPU y secciones donde el número de *limbs* es menor, limitando el paralelismo disponible, lo que es especialmente evidente en los resultados para RTX 4090 y B6000.

El modo de cambio de clave más eficiente para dos A4500/V100 es PeerAccess, mientras que para tres/cuatro V100 el modo más eficiente suele ser MemcpyPeer.

Poder agregar la memoria de varias GPU hace posible la ejecución de ResNet20 en A4500 y la de bert-tiny en RTX 4090 y A4500.

Los resultados detallados se pueden comprobar en la Figura 8

V. Trabajo Relacionado

Si bien existen propuestas de aceleradores hardware para FHE, incluyendo ASICs y FPGAs [48–54]; las GPUs ofrecen el mejor compromiso, combinando un prototipado rápido, alto ancho de banda de memoria, paralelismo masivo y soporte multi-GPU escalable, lo que las hace muy adecuadas para cargas de trabajo FHE cada vez más complejas.

El escalado compuesto (*composite scaling*) [55] mejora el mapeo de hardware separando la elección de la base RNS del escalado CKKS, mostrando fuertes ganancias en GPUs con HBM [15]. OpenFHE 1.3.0 [56] soporta esto en CPUs, y planeamos su futura adopción. El cambio de anillos (*ring-switching*) [57] reduce los costes de la dimensión del anillo, pero entra en conflicto con la necesidad de las MGPU de usar anillos grandes; son posibles enfoques de grano grueso que aprovechen el paralelismo GEMM tras aplicar tales técnicas. El *Bootstrapping* Generalizado ha mostrado un buen escalado en escenarios multi-dispositivo debido al mayor paralelismo expuesto por el algoritmo [52, 58]. Planeamos aprovechar las bibliotecas MGPU-TFHE de código abierto disponibles (por ejemplo, REDcuFHE [26]) en el futuro para explorar el *bootstrapping* generalizado.

VI. Conclusión

La solución técnica introducida en este trabajo presenta diferentes grados de eficacia en cuanto a la reducción de la latencia de cargas de trabajo de aprendizaje automático cifrado. Sí es eficaz en el objetivo de soportar cargas de trabajo de tamaño mayor al agregar la memoria de varios dispositivos y también al ser un enfoque que no requiere de esfuerzo adicional en el desarrollo de aplicaciones que preservan la privacidad, frente al uso de CPU o una sola GPU. Así, este trabajo supone una base sólida susceptible de ser ampliada en el futuro con técnicas adicionales para mejorar la escalabilidad de estas cargas de trabajo de forma práctica.

Agradecimientos

El presente trabajo ha sido financiado mediante las ayudas CNS2023-144241 y RYC2021-031966-I financiadas por MICIU/AEI/10.13039/501100011033 y por la Unión Europea NextGenerationEU/PRTR.

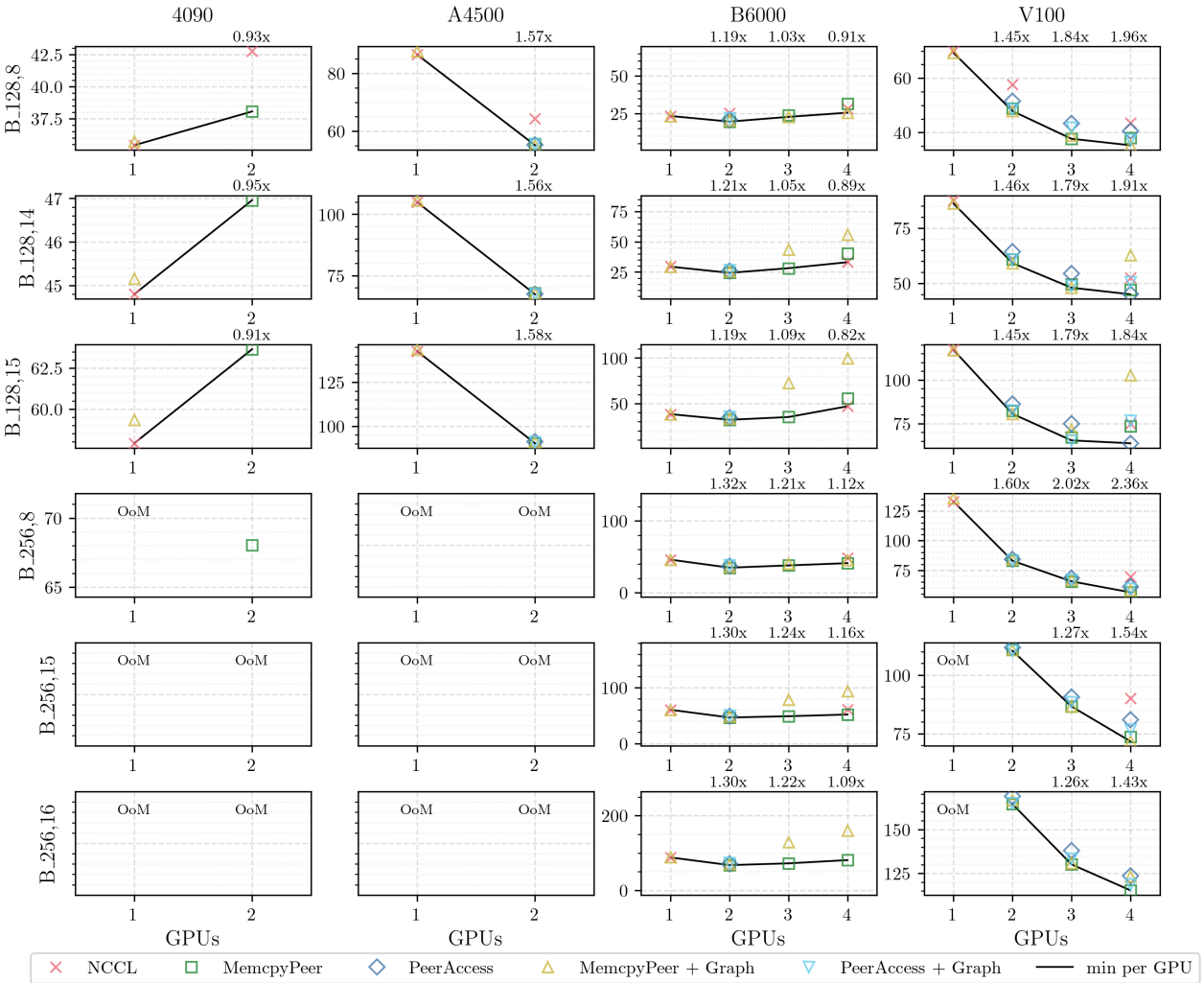


Fig. 7: Tiempos de Bootstrap en ms. Se asume escalado ideal en caso de falta de memoria (OoM)

Asimismo, la publicación es parte de la actuación 23039/GERM/25, financiada por FSRM/10.13039/100007801. Finalmente, agradecemos el apoyo de las ayudas NSF CNS 2312275 y 2312276, así como, en parte, de la NSF IUCRC Center for Hardware and Embedded Systems Security and Trust (CHEST).

Referencias

[1] Mauro Ribeiro, Katarina Grolinger, and Miriam AM Capretz, "Mlaas: Machine learning as a service," in *2015 IEEE 14th international conference on machine learning and applications (ICMLA)*. IEEE, 2015, pp. 896–902.

[2] Abhishek Mishra, *Machine learning in the AWS cloud: Add intelligence to applications with Amazon Sagemaker and Amazon Rekognition*, John Wiley & Sons, 2019.

[3] Shitalkumar R. Sukhdeve and Sandika S. Sukhdeve, *Google Cloud Platform for Data Science*, Apress Berkeley, CA, 2023.

[4] Christoph Körner and Kaijisse Waaijer, *Mastering Azure Machine Learning: Perform large-scale end-to-end advanced machine learning in the cloud with Microsoft Azure Machine Learning*, Packt Publishing Ltd, 2020.

[5] IBM, "Cost of a Data Breach Report," 2024, `https://www.ibm.com/reports/data-breach`.

[6] Susanne Lloyd-Jones and Kayleen Manwaring, "First steps to quantum resilience: Identifying 'broken concepts' in australia's national security laws," *ANU Journal of Law and Technology*, September 2024, Forthcoming.

[7] National Institute of Standards and Technology (NIST), "FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard," August 2024, `https://csrc.nist.gov/pubs/fips/203/final`.

[8] National Institute of Standards and Technology, "FIPS 204: Module-Lattice-Based Digital Signature Standard," August 2024, `https://csrc.nist.gov/pubs/fips/204/final`.

[9] Federal Register. The Daily Journal of the United States Government, "Announcing Issuance of Federal Information Processing Standards," August 2024, `https://www.federalregister.gov/d/2024-17956`.

[10] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 2017, pp. 409–437.

[11] Wonkyung Jung, Eojin Lee, Sangpyo Kim, Jongmin Kim, Namhoon Kim, Keewoo Lee, Chohong Min, Jung Hee Cheon, and Jung Ho Ahn, "Accelerating fully homomorphic encryption through architecture-centric analysis and optimization," *IEEE Access*, vol. 9, pp. 98772–98789, 2021.

[12] Yanwei Gong, Xiaolin Chang, Jelena Mišić, Vojislav B. Mišić, Jianhua Wang, and Haoran Zhu, "Practical solutions in fully homomorphic encryption: a survey analyzing existing acceleration methods," *Cybersecurity*, vol. 7, no. 1, 3 2024.

[13] Intel, "Why Data Center GPUs Are Essential to Innovation," `https://www.intel.com/content/www/us/en/products/docs/discrete-gpus/data-center-gpu/what-is-data-center-gpu.html`, 2024, [Online; accessed 13-December-2024].

[14] Hao Yang, Shiyu Shen, Wangchen Dai, Lu Zhou, Zhe Liu, and Yunlei Zhao, "Phantom: a cuda-accelerated word-

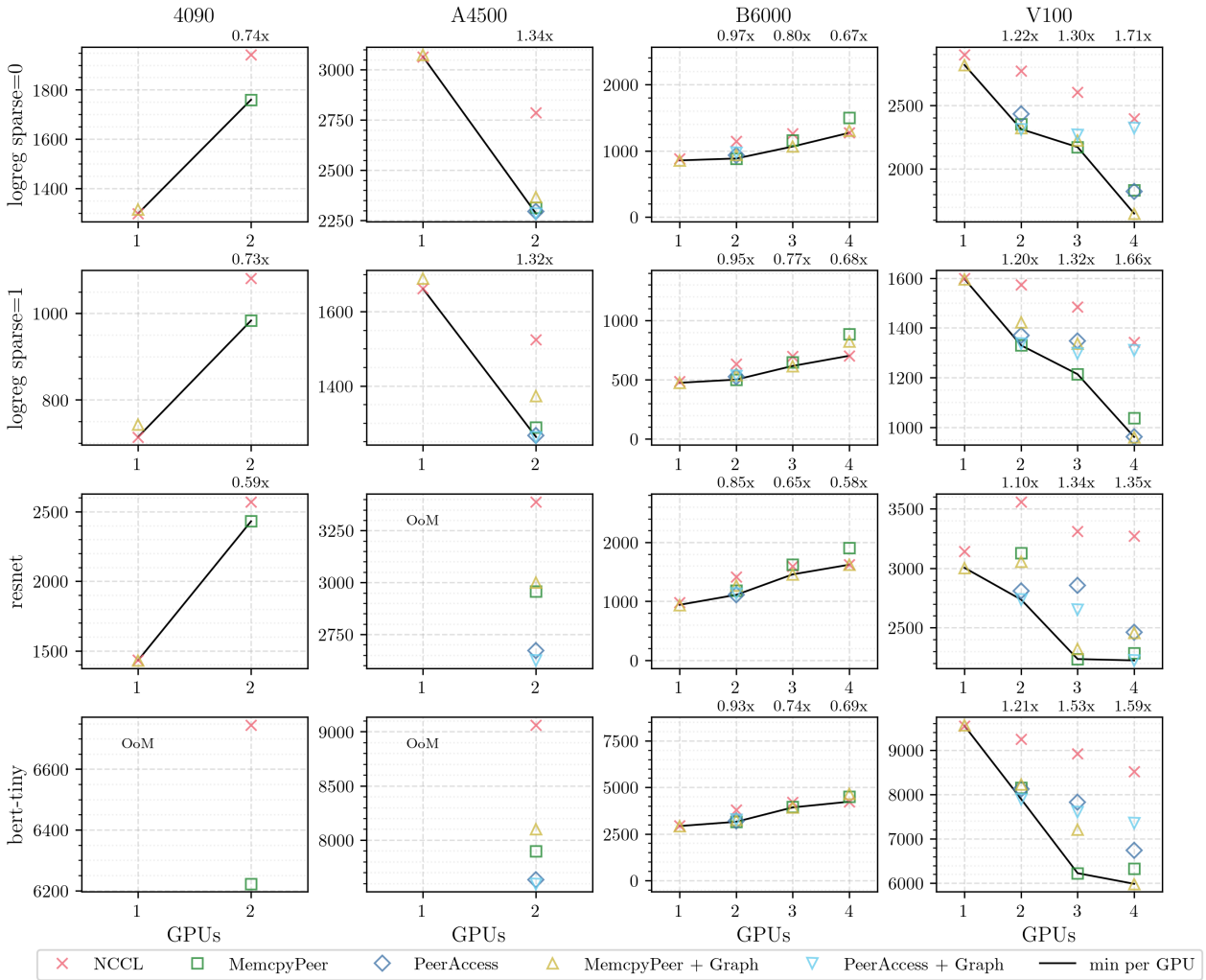


Fig. 8: Tiempos de aplicaciones ML en ms.

wise homomorphic encryption library," *IEEE Transactions on Dependable and Secure Computing*, 2024.

[15] Jongmin Kim, Wonseok Choi, and Jung Ho Ahn, "Cheddar: A swift fully homomorphic encryption library for cuda gpus," *arXiv preprint arXiv:2407.13055*, 2024.

[16] CRYPTOLAB INC., "Heaan private ai homomorphic encryption library. fully homomorphic encryption with ckks scheme, faster bootstrapping and gpu-acceleration.," 2022, `https://heaan.it/`.

[17] Ali Şah Özcan and Erkay Savaş, "HEonGPU: a GPU-based fully homomorphic encryption library 1.0," Cryptology ePrint Archive, Paper 2024/1543, 2024.

[18] Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee, "Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 114–148, 2021.

[19] lightbulb128, "The new implementation of troy, a cuda based gpu parallelized implementation of rlwe homomorphic encryption schemes.," 2023, `https://github.com/lightbulb128/troy-nova`.

[20] Zhiwei Wang, Peinan Li, Rui Hou, Zhihao Li, Jiangfeng Cao, XiaoFeng Wang, and Dan Meng, "He-booster: An efficient polynomial arithmetic acceleration on gpus for fully homomorphic encryption," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 4, pp. 1067–1081, 2023.

[21] Shengyu Fan, Zhiwei Wang, Weizhi Xu, Rui Hou, Dan Meng, and Mingzhe Zhang, "Tensorfhe: Achieving practical computation on encrypted data using gpgpu," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 922–934.

[22] DESILO, "Liberate.FHE: A New FHE Library for Bridging the Gap between Theory and Practice with a Focus on Performance and Accuracy," 2023, `https://github.com/Desilo/liberate-fhe`, `https://github.com/FHE-org/fhe-org.github.io/files/14896471/07-Kim.pdf`.

[23] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, et al., "Openfhe: Open-source fully homomorphic encryption library," in *proceedings of the 10th workshop on encrypted computing & applied homomorphic cryptography*, 2022, pp. 53–63.

[24] Daniele De Sensi, Lorenzo Pichetti, Flavio Vella, Tiziano De Matteis, Zebin Ren, Luigi Fusco, Matteo Turisini, Daniele Cesarini, Kurt Lust, Animesh Trivedi, Duncan Roweth, Filippo Spiga, Salvatore Di Girolamo, and Torsten Hoefler, "Exploring gpu-to-gpu communication: Insights into supercomputer interconnects," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2024, pp. 1–15.

[25] Wei Dai and Berk Sunar, "cuHE: A homomorphic encryption accelerator library," Cryptology ePrint Archive, Paper 2015/818, 2015.

[26] Lars Folkerts, Charles Gouert, and Nektarios Georgios Tsoutsos, "REDsec: Running Encrypted Discretized Neural Networks in Seconds," in *Network and Distributed System Security Symposium (NDSS)*, 2023, pp. 1–17.

[27] Ahmad Al Badawi, Bharadwaj Veeravalli, Jie Lin, Nan Xiao, Matsumura Kazuaki, and Aung Khin Mi Mi, "Multi-gpu design and performance evaluation of homomorphic encryption on gpu clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 379–391, 2021.

[28] Carlos Agulló-Domingo, Óscar Vera-López, Seyda Guzelhan, Lohit Daksha, Aymane El Jerari, Kaustubh Shivdikar, Rashmi Agrawal, David Kaeli, Ajay Joshi, and

José L. Abellán, "FIDESlib: A Fully-Fledged Open-Source FHE Library for Efficient CKKS on GPUs," in *2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2025, IEEE, Poster paper.

[29] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 2017, pp. 409–437.

[30] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song, "A full rns variant of approximate homomorphic encryption," in *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25*. Springer, 2019, pp. 347–368.

[31] Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov, "Approximate homomorphic encryption with reduced approximation error," in *Cryptographers' Track at the RSA Conference*. Springer, 2022, pp. 120–144.

[32] Kyoohyung Han and Dohyeong Ki, "Better bootstrapping for approximate homomorphic encryption," in *Cryptographers' Track at the RSA Conference*. Springer, 2020, pp. 364–390.

[33] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song, "Bootstrapping for approximate homomorphic encryption," in *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part I 37*. Springer, 2018, pp. 360–384.

[34] Shai Halevi and Victor Shoup, "Faster homomorphic linear transformations in helib," in *Annual International Cryptology Conference*. Springer, 2018, pp. 93–120.

[35] Jung Hee Cheon, Kyoohyung Han, and Minki Hhan, "Faster homomorphic discrete fourier transforms and improved fhe bootstrapping," *Cryptology ePrint Archive*, 2018.

[36] Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No, "High-precision bootstrapping of rns-ckks homomorphic encryption using optimal minimax polynomial approximation and inverse sine function," in *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I 40*. Springer, 2021, pp. 618–647.

[37] Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux, "Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2021, pp. 587–617.

[38] Jean-Philippe Bossuat, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux, "Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation," in *International Conference on Applied Cryptography and Network Security*. Springer, 2022, pp. 521–541.

[39] Hao Chen, Ilaria Chillotti, and Yongsoo Song, "Improved bootstrapping for approximate homomorphic encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019, pp. 34–54.

[40] NVIDIA, "NCCL: NVIDIA Collective Communications Library," `https://developer.nvidia.com/nccl`.

[41] AMD, "ROCm Communication Collectives Library (RCCL)," `https://github.com/ROCmSoftwarePlatform/rccl`.

[42] Sangpyo Kim, Jongmin Kim, Jaeyoung Choi, and Jung Ho Ahn, " CiFHER: A Chiplet-Based FHE Accelerator with a Resizable Structure ," in *2024 International Symposium on Secure and Private Execution Environment Design (SEED)*, Los Alamitos, CA, USA, May 2024, pp. 119–130, IEEE Computer Society.

[43] Siddharth Jayashankar, Edward Chen, Tom Tang, Wenting Zheng, and Dimitrios Skarlatos, "Cinnamon: A framework for scale-out encrypted ai," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, New York, NY, USA, 2025, ASPLOS '25, p. 133–150, Association for Computing Machinery.

[44] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park, "Efficient logistic regression on large encrypted data," Cryptology ePrint Archive, Paper 2018/662, 2018.

[45] Lorenzo Rovida and Alberto Leporati, "Encrypted image classification with low memory footprint using fully homomorphic encryption," *International Journal of Neural Systems*, vol. 34, no. 05, pp. 2450025, 2024, PMID: 38516871.

[46] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton, "Cifar-10 (canadian institute for advanced research)," .

[47] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu, "Tinybert: Distilling bert for natural language understanding," *arXiv preprint arXiv:1909.10351*, 2019.

[48] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez, "F1: A fast and programmable accelerator for fully homomorphic encryption," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, New York, NY, USA, 2021, MICRO '21, p. 238–252, Association for Computing Machinery.

[49] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sanchez, "Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, New York, NY, USA, 2022, ISCA '22, p. 173–187, Association for Computing Machinery.

[50] Jongmin Kim, Sangpyo Kim, Jaewan Choi, Jaiyoung Park, Donghwan Kim, and Jung Ho Ahn, "Sharp: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, New York, NY, USA, 2023, ISCA '23, Association for Computing Machinery.

[51] Rashmi Agrawal, Leo de Castro, Guowei Yang, Chiraag Juvekar, Rabia Yazicigil, Anantha Chandrakasan, Vinod Vaikuntanathan, and Ajay Joshi, "Fab: An fpga-based accelerator for bootstrappable fully homomorphic encryption," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 882–895.

[52] Rashmi Agrawal, Anantha Chandrakasan, and Ajay Joshi, "Heap: A fully homomorphic encryption accelerator with parallelized bootstrapping," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 756–769.

[53] M. Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai, "Heax: An architecture for computing on encrypted data," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, 2020, ASPLOS '20, p. 1295–1309, Association for Computing Machinery.

[54] Jongmin Kim, Gwangho Lee, Sangpyo Kim, Gina Sohn, Minsoo Rhu, John Kim, and Jung Ho Ahn, "Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Oct 2022, pp. 1237–1254, IEEE.

[55] Rashmi Agrawal, Jung Ho Ahn, Flavio Bergamaschi, Ro Cammarota, Jung Hee Cheon, Fillipe DM de Souza, Huijing Gong, Minsik Kang, Duhyeong Kim, Jongmin Kim, et al., "High-precision rns-ckks on fixed but smaller word-size architectures: theory and application," in *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2023, pp. 23–34.

[56] OpenFHE Development Team, "OpenFHE Development Repository," `https://github.com/openfheorg/openfhe-development`, 2025, Accessed: 2025-06-30.

[57] Youngjin Bae, Jung Hee Cheon, Guillaume Hanrot, Jai Hyun Park, and Damien Stehlé, "Plaintext-ciphertext matrix multiplication and fhe bootstrapping: Fast and fused," in *Annual International Cryptology Conference*. Springer, 2024, pp. 387–421.

[58] Zhiwei Wang, Haoqi He, Lutan Zhao, Peinan Li, Zhihao Li, Dan Meng, and Rui Hou, "Chameleon: An efficient fhe scheme switching acceleration on gpus," *arXiv preprint arXiv:2410.05934*, 2024.

Aprovechamiento de la gestión dinámica de recursos para la eficiencia energética: un caso de estudio

Paula Sánchez-Checa, Jesus Carretero y David E. Singh[1]

Resumen— **En HPC, muchas aplicaciones presentan un comportamiento dinámico con diferentes fases de ejecución y una intensidad computacional variable. Para esta clase de aplicaciones, una política de asignación de recursos rígida no siempre garantiza la ejecución más eficiente. Un enfoque rígido puede resultar ineficiente por dos motivos: o bien se aprovisionan más recursos de los necesarios, que la aplicación no puede aprovechar eficazmente, consumiendo más energía de la necesaria, o bien la falta de recursos incrementa el tiempo de ejecución y, muy probablemente, el consumo energético de las fases computacionales más exigentes.**

En este artículo, abordamos dicho desafío estudiando, en primer lugar, el comportamiento energético del caso de uso de un simulador epidemiológico basado en agentes bajo distintas condiciones. A continuación, analizamos el impacto de diferentes políticas de afinidad entre procesos y núcleos en el rendimiento de la aplicación y, seguidamente, proponemos una gestión dinámica del número de procesos en tiempo de ejecución, con el objetivo de reducir simultáneamente el tiempo de ejecución y el consumo de energía. Nuestros experimentos demuestran que, al reconfigurar dinámicamente los recursos para ajustarlos a las distintas fases de la aplicación, es posible reducir significativamente tanto el tiempo total de ejecución como el consumo energético global.

Palabras clave— **Computación de altas prestaciones; Gestión dinámica de recursos; Eficiencia energética; Epidemiología.**

I. Introducción

La computación de altas prestaciones (HPC) se ha convertido en un elemento clave tanto en la investigación como en la industria [1], [2]. Desde la modelización de sistemas biológicos complejos [3] hasta el desarrollo de modelos de inteligencia artificial (IA) a gran escala [4], las aplicaciones de HPC requieren enormes recursos computacionales para procesar grandes conjuntos de datos y realizar operaciones aritméticas complejas, que superan las capacidades de los equipos de trabajo estándar. Por consiguiente, estas aplicaciones dependen de clústeres de HPC capaces de hacer frente a estas intensas exigencias computacionales. La ejecución de aplicaciones en estas plataformas plantea un reto cada vez mayor: el consumo energético. En la actualidad, este ya representa al menos el 1 % de la demanda eléctrica mundial [5], registrando aumentos del 32 % en Irlanda entre 2020 y 2021 y del 100 % en los Países Bajos entre 2017 y 2020 [6]. Más allá de la energía que necesitan las unidades de procesamiento para realizar los cálculos del programa, también hay una parte del consumo energético que proviene de subsistemas subyacentes como la refrigeración, la entrada/salida (E/S), la conectividad de red y la monitorización.

La obtención de métricas de energía precisas presenta varios problemas de monitorización. Uno de ellos es aislar el consumo de energía de una aplicación específica. La mayoría de sensores físicos miden la energía a nivel de *socket*, dificultando la distinción de la energía consumida por la aplicación y la consumida por el sistema operativo y otros procesos en segundo plano. Otro reto lo presenta la heterogeneidad de las arquitecturas de *hardware*, ya que diferentes componentes (por ejemplo, CPUs y GPUs) pueden estar basados en distintas arquitecturas y, por lo tanto, distintos sistemas de control y monitorización (por ejemplo, Intel RAPL o NVIDIA Management Library), cada uno de los cuales trabaja con distintas frecuencias y condiciones. Por último, otra limitación a la hora de monitorizar la energía son las restricciones de acceso. En entornos HPC en producción, es común que los usuarios no tengan permisos de superusuario (*root*) para acceder a contadores y sensores, o instalar o ejecutar herramientas de medición de energía externas, lo que limita el nivel de detalle y reduce las opciones para monitorizar la energía consumida.

Nuestro objetivo es monitorizar las aplicaciones en tiempo de ejecución y adaptar dinámicamente el número de procesos con los que se ejecutan para reducir el consumo energético, sin comprometer el rendimiento de la aplicación. En este trabajo, nos enfocamos en la optimización de energía a nivel de aplicación. Como caso de estudio, hemos seleccionado EpiGraph, un simulador epidemiológico basado en agentes, escalable y distribuido [7] diseñado para modelar la propagación de enfermedades epidémicas (gripe, SARS-CoV-2 y VSR) con gran precisión. Para monitorizar el consumo energético de esta aplicación bajo condiciones reales de HPC, hemos llevado a cabo la evaluación experimental en un clúster HPC en producción [8], cuyas características se describen en la Sección III. Este artículo presenta las siguientes contribuciones:

- El estudio del perfil energético del simulador epidemiológico EpiGraph.
- Un análisis de cómo la afinidad entre procesos y núcleos afecta al consumo energético y tiempo de ejecución de la aplicación. Además, evalua-

[1]Universidad Carlos III de Madrid, ROR: https://ror.org/03ths8210, Departamento de Informática, Avenida de la Universidad, 30 (edificio Sabatini), 28911, Leganés, Madrid, España, e-mail: {paulasan,jcarrete,dexposit}@inf.uc3m.es.

mos una metodología que asigna dinámicamente procesos a núcleos para mejorar el rendimiento.
- Proponemos la reconfiguración dinámica de recursos para variar el número de procesos en tiempo de ejecución.
- Llevamos a cabo una evaluación experimental en una plataforma HPC en producción.

El resto de este artículo sigue la siguiente estructura: la Sección II ofrece una visión general de los mecanismos y herramientas de medición de energía más relevantes y trabajos relacionados centrados en energía y reconfiguración dinámica de recursos en HPC; la Sección III describe el perfil energético de la aplicación usada en el caso de estudio; la Sección IV presenta y analiza los resultados experimentales de la reconfiguración dinámica de recursos; y finalmente, la Sección V cierra el artículo con un resumen de las contribuciones y resultados principales obtenidos.

II. Antecedentes

En HPC existen diferentes herramientas que permiten medir energía, desde sensores de bajo nivel hasta aplicaciones de alto nivel que permiten acceder a las medidas con un gran nivel de abstracción. Sin embargo, la disponibilidad de estas herramientas depende de la arquitectura y, frecuentemente, su acceso está restringido a los usuarios por razones de seguridad.

A. Contadores de rendimiento de hardware

Los contadores de rendimiento de *hardware* son registros integrados en el procesador que almacenan estadísticas y actividades relacionadas con eventos físicos (*hardware*), tales como el número de instrucciones ejecutadas, los fallos de caché o el consumo energético. Entre los contadores de rendimiento de *hardware* más destacados se encuentran RAPL, IPMI y los contadores PM.

RAPL (*Running Average Power Limit*) [9] es una funcionalidad que permite monitorizar el consumo energético del procesador en tiempo de ejecución. Estas mediciones, expresadas en microjulios (uJ), se obtienen a través de los registros MSRs (*Model Specific Registers*) para distintos dominios, como los *sockets* de la CPU y la memoria DRAM [10], [11]. RAPL estima la energía consumida basándose en los contadores de eventos de *hardware* y modelos de fugas [12].

Por otro lado, IPMI (*Intelligent Platform Management*) [13] es un conjunto de herramientas de monitorización y gestión que se ejecutan de forma independiente del sistema operativo, los procesadores y la BIOS. Es compatible con controladores como los componentes de conexión (por ejemplo, BMC, *Baseboard Management Controllers*). A través de IPMI, es posible obtener el consumo de energía leyendo directamente las mediciones de los distintos sensores físicos del sistema [14].

Finalmente, los contadores HPE Cray Power Monitoring (PM) [15], diseñados originalmente para el supercomputador Cray XC30, facilitan el acceso a datos globales de energía y potencia a nivel de nodo. Estas métricas se obtienen a través de la interfaz *sysfs* de Linux y se almacenan en memoria, permitiendo a los usuarios acceder a ellas [16].

B. Herramientas e interfaces para el modelado del rendimiento

Las herramientas e interfaces para el modelado del rendimiento están diseñadas para recopilar datos y facilitar el análisis del rendimiento de las aplicaciones en tiempo de ejecución. Estas herramientas actúan como intermediarias entre los contadores de *hardware* de bajo nivel y el usuario, ayudando a contextualizar las mediciones obtenidas.

Power Capping Framework (*powercap*) [17] es una interfaz proporcionada por el kernel estándar de Linux que permite acceder a RAPL desde el espacio de usuario. Utiliza *sysfs* para habilitar el acceso a los dispositivos de limitación de potencia, exponiendo sus distintos componentes mediante una estructura de árbol. Estos componentes se encuentran habitualmente en la ruta `/sys/class/powercap`, a través de la cual los usuarios tienen acceso de lectura a los registros de energía, ficheros normalmente nombrados `energy_uj`, que proporcionan el consumo de energía acumulado en microjulios.

Por su parte, PAPI (*Performance Application Programming Interface*) [12] actúa como un *middleware* que proporciona acceso a los contadores de *hardware* disponibles en diferentes arquitecturas. Esta librería abstrae la complejidad y permite acceder a los registros de energía y potencia del sistema, ofreciendo tanto una interfaz de alto nivel como una de bajo nivel para lenguajes como C.

LIKWID [18], [19] es una suite de herramientas de línea de comandos orientada a la monitorización del consumo eléctrico y energético. Mediante la lectura de los contadores de *hardware*, ofrece información sobre el consumo energético de la aplicación y muestra métricas derivadas [20]. Esta herramienta proporciona métricas de rendimiento energético para la ejecución completa o de regiones específicas del código.

Finalmente, Perf es una herramienta de elaboración de perfiles integrada en el kernel de Linux [21]. Permite el acceso a datos de rendimiento relacionados con distintos tipos de eventos, ya sean de *software*, de *hardware* o de memoria caché. En el contexto energético, Perf extrae los datos de PMU (Kernel Performance Monitoring Unit), accediendo directamente a los contadores de *hardware* [22].

C. Metodologías orientadas a la eficiencia energética

En lo que respecta a los mecanismos de generación de perfiles de consumo energético, existen varios trabajos que comparan el uso de diversos contadores de *hardware* y de herramientas de generación de perfiles. En trabajos como [23] y [11], los autores comparan distintos contadores de rendimiento de *hardware* y herramientas para medir el consumo energético. Entre otros, analizan contadores como RAPL y herramientas como Perf. En [24], los autores centran su estudio en el análisis de diferentes mecanismos de

gestión energética que utilizan mediciones RAPL. En concreto, comparan el uso de MSR, *powercap* y eventos Perf con eBPF para las arquitecturas de Intel y AMD.

En lo que respecta a las mediciones de energía en HPC, en [25] los autores presentan un planificador dinámico de perfiles energéticos para aplicaciones basadas en MPI que utiliza modelos anteriores de la aplicación para determinar la mejor configuración para su ejecución. En [26], los autores buscan minimizar el consumo energético adaptando dinámicamente el número de procesos utilizados por la aplicación a su intensidad computacional. Además, en [27] los autores proponen distintas estrategias para aprovechar la gestión dinámica con el fin de optimizar la eficiencia energética y respetar los límites de potencia. Proponen estrategias como la reasignación virtual de recursos mediante la transición de los nodos inactivos a estados de bajo consumo para reactivarlos únicamente bajo demanda, la redistribución dinámica de potencia hacia los componentes de *hardware* o nodos específicos que presenten mayor carga en un instante dado y el ajuste de los parámetros del sistema en función de las distintas fases de ejecución. Finalmente, en [28], se utiliza un algoritmo *greedy* que tiene en cuenta el consumo energético para programar y reconfigurar la cantidad de servidores a utilizar y apagar los servidores restantes.

En trabajos anteriores se ha utilizado el simulador EpiGraph para desarrollar diferentes metodologías destinadas a mejorar las métricas de ejecución de las aplicaciones HPC paralelas. En [29], los autores desarrollaron un marco para reducir la contención de E/S al ejecutar múltiples aplicaciones simultáneamente. Además, en [30], se compararon dos algoritmos para tomar decisiones dinámicas de asignación de recursos en tiempo de ejecución con el fin de optimizar los tiempos de CPU, comunicación, E/S y utilización de recursos.

Sin embargo, los trabajos anteriores no tenían en cuenta el consumo energético, y la mayoría de los enfoques descritos en la bibliografía se centran en pruebas sintéticas. Además, según el conocimiento de los autores, estos enfoques que tienen en cuenta el consumo energético no se han aplicado a aplicaciones reales y complejas con una carga computacional variable, como el simulador utilizado en este trabajo.

D. Caso de estudio: EpiGraph

EpiGraph es un simulador epidemiológico que reproduce patrones de interacción complejos mediante el modelado de características específicas y el comportamiento de cada agente, simulando la evolución de las enfermedades tanto a nivel individual como social (propagación de la enfermedad). La aplicación simula la propagación de enfermedades epidémicas mediante el uso de tres modelos diferentes: un modelo social [31] que determina las interacciones sociales diarias (trabajo, colegio, familia, etc.), un modelo de transporte que incluye los desplazamientos de las personas entre ciudades y un modelo epidémico que

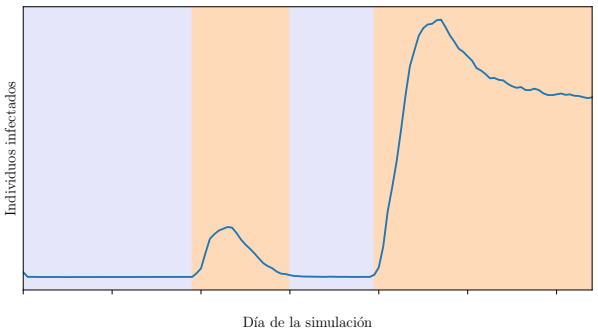


Fig. 1: Número de individuos infectados en la simulación. Las regiones lavanda representan fases intensivas en E/S y comunicaciones, mientras que las regiones naranjas representan fases intensivas en CPU.

simula la progresión de la enfermedad en un individuo a través de diferentes estados (por ejemplo, incubación, asintomático, hospitalización). Estos tres modelos constituyen las operaciones más importantes del flujo de trabajo del simulador. El simulador discretiza el tiempo en días simulados. En el bucle principal de la aplicación, se recorren los días simulados, actualizando el estado de la simulación en cada iteración del bucle. Está diseñado para escalar las simulaciones para modelar la propagación de enfermedades a nivel de ciudad y país, y es capaz de simular escenarios a nivel de continente con cientos de millones de agentes.

El escenario analizado en este trabajo es el brote de las variantes Delta y Ómicron [32] del SARS-CoV-2 en España en 2021, en un escenario simulado con 3,5 millones de personas. La Fig. 1 representa el número de personas infectadas a lo largo de la simulación, que es proporcional al tiempo de CPU. En esta figura, podemos observar dos olas diferentes en las que aumenta el número de personas infectadas. Estas dos olas corresponden a las infecciones por las variantes Delta (días 40-60 de la simulación) y Ómicron (a partir del día 80 de la simulación) del SARS-CoV-2.

Desde un punto de vista técnico, el simulador es intensivo en cómputo y comunicaciones, así como intensivo en entrada/salida (E/S) cuando hay puntos de control (*check-pointing*). La aplicación está desarrollada en C y paralelizada usando MPI.

El simulador cuenta con diferentes fases que se alternan entre etapas de cómputo, de comunicación y de E/S. Cuando el número de individuos infectados es bajo, la aplicación no consume muchos recursos de CPU, sino que hace un uso intensivo de las operaciones de E/S y de la comunicación (zonas azules en la Fig. 1). Pero a medida que aumenta el número de individuos infectados, también aumenta la carga de cómputo, y las operaciones de E/S y las comunicaciones se espacian en el tiempo, lo que hace que el simulador requiera más CPU (zonas rojas en la Fig. 1).

La versión actual de EpiGraph está integrada con FLEX-MPI, una librería que proporciona a las aplicaciones paralelas capacidades de monitorización, balanceo de carga, redistribución de datos y malea-

bilidad [33]. Gracias a FLEX-MPI, EpiGraph puede modificar dinámicamente el número de procesos y de nodos utilizados en tiempo de ejecución. El sistema de monitorización permite recopilar métricas de rendimiento de forma transparente, sin necesidad de alterar el código fuente. En concreto, las métricas de cómputo se capturan mediante contadores *hardware* (utilizando PAPI), mientras que las métricas de comunicación se capturan a través de la interfaz MPI. Toda esta información se emplea para monitorizar la ejecución y detectar situaciones de desequilibrio computacional. Cuando esto sucede, los datos y la carga de trabajo se redistribuyen para volver a equilibrar el sistema. Finalmente, para los experimentos realizados en la Sección IV de este trabajo, hemos utilizado el modo de maleabilidad *stop/restart* disponible en FLEX-MPI debido a su mayor simplicidad.

III. Análisis de energía

En este trabajo, nos centramos en comprender cómo varía el consumo energético al ejecutar una aplicación con múltiples fases de ejecución (CPU, comunicaciones y E/S), cuyas intensidades cambian durante el transcurso del programa. El segundo aspecto que abordamos consiste en analizar la relación entre el consumo energético y el tiempo de ejecución y los componentes de *hardware* y *software* del sistema. Para ello, medimos la energía en el bucle principal de la aplicación, registrando el consumo correspondiente a cada uno de los días simulados.

Para obtener estas mediciones energéticas, utilizamos *powercap*. Hemos seleccionado esta herramienta de medición porque nos permite obtener el consumo de todo el nodo, lo que permite medir por separado la energía de la CPU y de la memoria DRAM mediante la lectura de los contadores de rendimiento que proporciona RAPL directamente desde nuestra aplicación. Además, dado que ejecutamos nuestra aplicación en un clúster en producción, *powercap* ofrece lecturas de energía a través de *sysfs*, lo que permite a los usuarios sin privilegios de superusuario acceder a los registros de los contadores de *hardware*. Por último, esta interfaz está disponible en la mayoría de los clústeres Linux y ofrece datos de consumo para diversas arquitecturas, lo que facilita aplicar la misma metodología en distintos entornos.

Para obtener el consumo total de energía de la CPU por cada día simulado, y dado que *powercap* proporciona mediciones acumuladas, calculamos la diferencia entre las lecturas de energía al inicio y al final de cada iteración del bucle. Este proceso se llevó a cabo en cada nodo de cómputo involucrado en la ejecución y, posteriormente, todas las métricas de los distintos nodos se agregaron en una única métrica global.

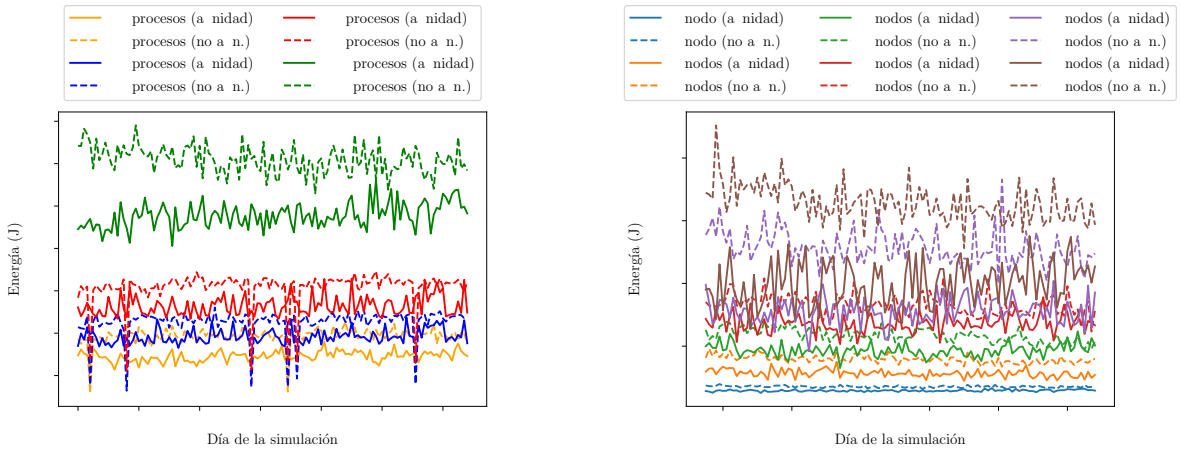
El simulador puede configurarse para habilitar o no los puntos de control (*check-pointing*). Medimos el consumo de energía bajo ambas configuraciones (con los puntos de control activados y desactivados) y observamos diferencias significativas en el gasto energético en cada caso. Para comprender estas discrepancias, llevamos a cabo un estudio de la carga de trabajo y de la frecuencia de los núcleos con ambas configuraciones. Gracias a ello, constatamos que el planificador proporciona afinidad entre los procesos y los núcleos cuando los puntos de control están desactivados, sin embargo, esto no ocurre cuando están activados. En este último supuesto, la carga de trabajo se transfiere entre los distintos núcleos disponibles debido a las políticas de planificación del sistema operativo, provocando que los procesos pierdan su afinidad inicial.

Además, existen dos configuraciones mediante las cuales podemos ejecutar trabajos en los nodos de cómputo: con o sin acceso exclusivo a los mismos. Cuando un trabajo se ejecuta en exclusividad en un nodo mediante SLURM, esto implica que el nodo queda reservado íntegramente para dicho trabajo y no puede ser utilizado por otros trabajos. Por el contrario, cuando se ejecuta en un nodo no exclusivo, se asigna al trabajo un número determinado de núcleos, dejando el resto a disposición de otros trabajos y usuarios. Al reservar un nodo con exclusividad, los procesos en ejecución pierden su afinidad con los núcleos. En cambio, cuando ejecutamos trabajos sin exclusividad, SLURM asigna un núcleo específico a cada proceso y esta afinidad no cambia durante la ejecución del programa.

Esto se debe a que, en un nodo exclusivo, no se ejecutan otros trabajos, por lo que la totalidad de los núcleos queda a disposición del programa. Por el contrario, cuando el nodo no es exclusivo, distintos trabajos pueden compartir el mismo nodo de cómputo. Por consiguiente, resulta necesario garantizar que el número de núcleos solicitados se asigne de manera exclusiva al trabajo; de ahí que SLURM fuerce la afinidad entre procesos y núcleos. En algunos clústeres, la configuración sin afinidad es la predeterminada. Además, es posible que algunos usuarios tiendan a ejecutar sus aplicaciones de esta manera, ya que reservar un nodo en exclusividad implica ser tarifado por el uso completo del mismo, incluso aunque no se utilicen todos los núcleos, lo que puede suponer un elevado coste económico. Por estas razones, en este artículo, analizamos el comportamiento del consumo energético al ejecutar la aplicación con y sin afinidad. En consecuencia, en la Sección III-A evaluamos el impacto de la afinidad sobre el consumo de energía de la aplicación. Asimismo, en la Sección III-B examinamos cómo la afinidad y la carga de trabajo afectan a la frecuencia de los núcleos.

A. Análisis de afinidad proceso-núcleo

En esta sección, estudiamos cómo la afinidad de los procesos a los núcleos (o la falta de ella) afecta al consumo energético de la aplicación. Para ello, necesitamos ejecutar la misma versión de la aplicación (con los puntos de control activados) con y sin afinidad. Además, para garantizar que la energía medida no se viera afectada por otros trabajos en ejecución dentro del mismo nodo (medimos la energía



(a) Ejecución en un nodo con 16, 32, 68 y 128 procesos.

(b) Ejecución en múltiples nodos con 128 procesos por nodo.

Fig. 2: Comparativa del consumo energético al ejecutar la aplicación con y sin afinidad.

de los puntos de control completos), ejecutamos los trabajos en nodos exclusivos[1]. Por defecto, el planificador no proporciona afinidad a la aplicación bajo esta configuración y entorno. Por esta razón, hemos desarrollado un algoritmo que asigna dinámicamente cada proceso a un núcleo mediante la llamada al sistema `sched_setaffinity`. De este modo, podemos comparar el consumo de energía y de potencia, así como las frecuencias de los núcleos, bajo las mismas condiciones para ambos escenarios.

Todos los resultados mostrados en este artículo provienen de pruebas realizadas en un clúster HPC en producción y representan la mediana de 5 ejecuciones independientes. El sistema está equipado con 90 nodos de cómputo provistos de procesadores duales AMD EPYC 7713 (de 64 núcleos cada uno), interconectados mediante una red HDR InfiniBand a 100 Gbps, y utiliza el gestor de cargas de trabajo SLURM. Cuenta con Lustre como sistema de almacenamiento y 1.024 GB de memoria RAM DDR4-3200 ECC RDIMM.

La Fig. 2 muestra el consumo de energía derivado de ejecutar la aplicación en un nodo de cómputo con 16, 32, 64 y 128 procesos (Fig. 2a), y ejecutada en 1, 2, 3, 4, 5 y 6 nodos con 128 procesos por nodo (Fig. 2b), con y sin afinidad. En esta figura, se puede observar que, en todos los casos, el consumo energético se reduce cuando se utiliza afinidad. Asimismo, en la Fig. 2a podemos remarcar que la reducción del consumo de energía aumenta con el número de procesos. Es decir, a mayor número de procesos, mayor es la reducción del consumo energético que podemos lograr si empleamos afinidad en nuestras ejecuciones. Cabe destacar que el consumo de energía de la ejecución con 128 procesos sin afinidad es considerablemente mayor que el del resto de ejecuciones (más del doble). La razón subyacente es, probablemente, la sobrecarga (*overhead*) asociada a la migración de una cantidad tan elevada de procesos entre núcleos, sin sacar partido de la localidad de los datos. Esto significa que es muy probable que los procesos no puedan beneficiarse de los datos almacenados en la memoria caché de un núcleo antes de ser trasladados a uno nuevo (fenómeno conocido como *cache-thrashing*).

En la Fig. 2b, al igual que en el caso de un único nodo, se observa que, en todos los casos, el consumo energético se reduce en aquellos escenarios en los que se aplica afinidad entre procesos y núcleos, en comparación con los escenarios sin afinidad. Asimismo, se aprecia que el consumo de energía aumenta a medida que incrementa el número de nodos utilizados. Esto es previsible, ya que se consume más energía al emplear un mayor número de nodos de cómputo. Además, es importante destacar que las diferencias entre las ejecuciones con y sin afinidad se amplían a medida que aumenta el número de nodos.

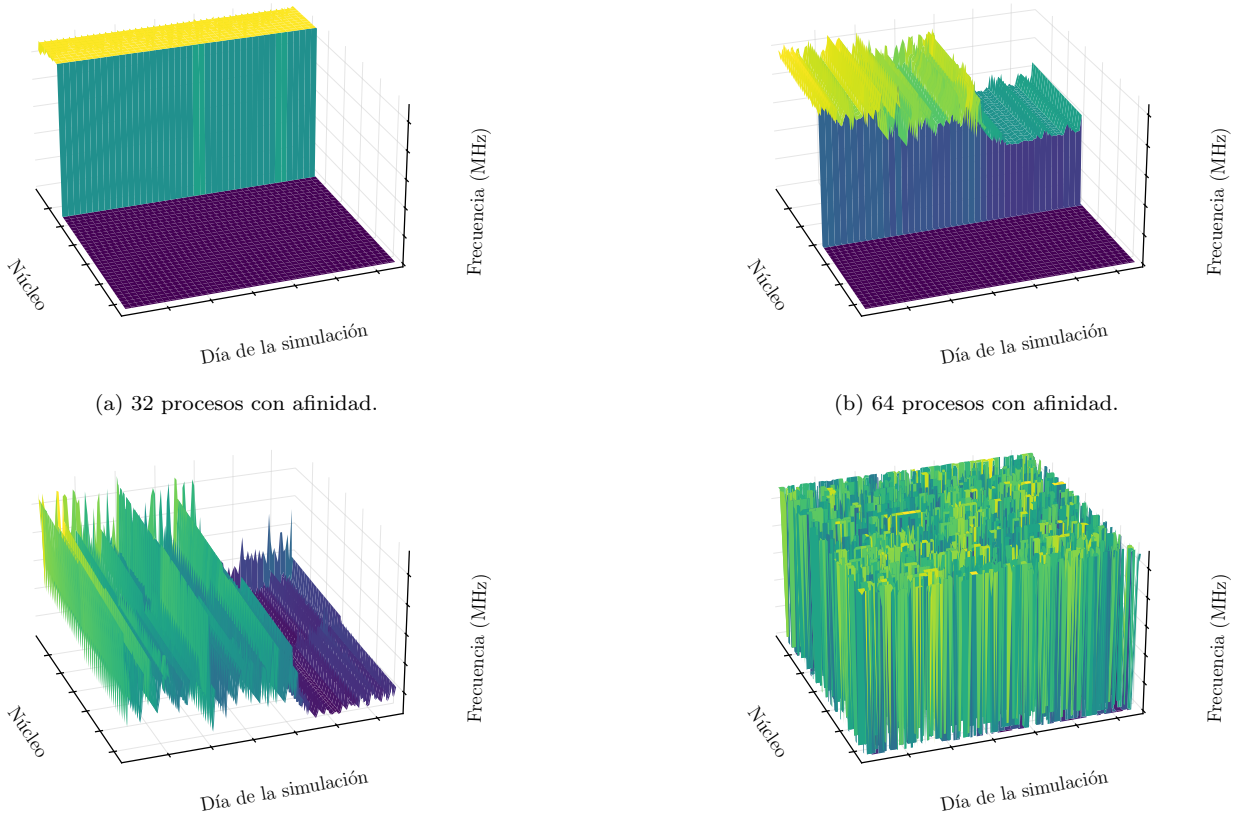
En general, podemos concluir que el uso de afinidad es más eficiente a nivel energético en todos los escenarios evaluados. Además, este beneficio adquiere mayor relevancia en aquellas situaciones en las que se emplea más *hardware* y se incrementan las comunicaciones, ya que se evitan el *cache-thrashing* y otros inconvenientes derivados de tener que migrar una gran cantidad de procesos entre núcleos.

B. *Análisis de la frecuencia de los núcleos*

Una vez demostrado empíricamente que la afinidad entre procesos y núcleos reduce el consumo energético de la aplicación, el siguiente paso consiste en comprobar que dicha afinidad también mejora su rendimiento. En nuestros experimentos, evaluamos la frecuencia de los núcleos a lo largo de la ejecución de la simulación, registrando estas mediciones para distintas cantidades de procesos. Estas pruebas proporcionan información acerca de cómo influye la variación en el número de procesos en el rendimiento de cada núcleo. Asimismo, evaluamos las diferencias de frecuencia al ejecutar un trabajo con y sin afinidad.

Las figuras 3a, 3b y 3c muestran la frecuencia de todos los núcleos de un nodo al ejecutar la aplicación con afinidad. Como puede apreciarse, los núcleos en

[1]Todos los resultados presentados en este artículo se han obtenido de aplicaciones ejecutadas en nodos exclusivos, a menos que se indique lo contrario.



(a) 32 procesos con afinidad.

(b) 64 procesos con afinidad.

(c) 128 procesos con afinidad.

(d) 64 procesos sin afinidad.

Fig. 3: Frecuencias de los núcleos al ejecutar la aplicación con distintos números de procesos en un solo nodo, con y sin afinidad.

los que se ejecutan procesos presentan una frecuencia superior a la de aquellos en los que no se ejecuta ningún proceso (cuya frecuencia se mantiene constante a 2.000 MHz). Esto confirma que, al ejecutar nuestra aplicación con afinidad, los procesos se ejecutan exclusivamente en los núcleos asignados.

Si examinamos más detenidamente las especificaciones del procesador [34], podemos observar que 3.720 MHz se corresponde con la frecuencia máxima (*boost clock*) que puede alcanzar un único núcleo de la CPU. Por tanto, resulta improbable que algún núcleo alcance dicha cifra durante la ejecución de la aplicación, sobre todo si tenemos en cuenta que solo un núcleo puede operar a esa frecuencia máxima y que nuestra ejecución emplea múltiples núcleos. En lo que respecta a la frecuencia mínima, esta se sitúa en 1.500 MHz. Sin embargo, la frecuencia base es de 2.000 MHz, por lo que los 1.500 MHz corresponden al estado de ahorro de energía del procesador. Además, debemos considerar que el procesador tiene una potencia de diseño térmico (TDP) predeterminada de 225 W, un límite que no se puede sobrepasar. Por consiguiente, resulta lógico que las frecuencias máximas obtenidas en los núcleos activos ronden los 3.000 MHz (de modo que no se excede la potencia de diseño térmico), mientras que los núcleos inactivos se mantengan en la frecuencia base.

Asimismo, se observa que las frecuencias varían en función de la carga de los núcleos. En aquellos casos en los que el nodo presenta una carga baja (menor número de procesos, Fig. 3a), la frecuencia se mantiene en su valor máximo en todos los núcleos durante toda la ejecución de la aplicación. No obstante, a medida que incrementamos el número de procesos (Fig. 3b y Fig. 3c), se aprecia que la frecuencia deja de mantenerse al máximo durante la ejecución del programa. En estos escenarios, la frecuencia de los núcleos es inversamente proporcional a la carga computacional del nodo. Es decir, en aquellos instantes de la ejecución donde la carga computacional es reducida, la frecuencia de los núcleos es mayor, mientras que esta disminuye a medida que la carga aumenta. El motivo detrás de este comportamiento es que tan solo un número relativamente bajo de núcleos puede operar a la frecuencia máxima (*maximum boost*). Sin embargo, a medida que aumenta el número de procesos, el consumo de potencia total comienza a acercarse a la potencia de diseño térmico, obligando a cada núcleo a reducir su frecuencia para no rebasarlo.

Para analizar el efecto de la afinidad, podemos comparar las frecuencias resultantes de ejecutar la aplicación con 64 procesos con afinidad (Fig. 3b) y sin ella (Fig. 3d). Cabe destacar que, en la figura correspondiente a la ejecución sin afinidad, se observa que todos los núcleos están en uso, lo que implica que los procesos se han ido distribuyendo entre múltiples núcleos a lo largo de la ejecución de la aplicación. Además, la contención de la frecuencia (es decir, la relación inversamente proporcional entre la frecuen-

cia y la carga computacional) no se produce cuando la afinidad está desactivada, lo cual es coherente con el análisis previo. Cuando no se emplea afinidad, el tiempo de ejecución de la aplicación es significativamente mayor. Esto se debe al razonamiento expuesto con anterioridad: cada proceso se ejecuta en varios núcleos, impidiendo aprovechar la localidad de la memoria caché, lo que introduce mayores latencias en el acceso a los datos. Durante estos tiempos de espera, los núcleos presentan un menor consumo energético, por lo que no alcanzan el límite de potencia y el procesador no se ve en la necesidad de restringir la frecuencia.

IV. Reconfiguración dinámica de recursos

En el contexto de este trabajo, entendemos por gestión dinámica de recursos (o maleabilidad de la aplicación) la modificación de la cantidad de recursos (p. ej., nodos de cómputo o procesos) que emplea una aplicación durante su ejecución para adaptarse a sus requisitos. Dado que nuestra aplicación presenta distintas fases en las que varían las cargas computacionales, de comunicaciones y de E/S, esta gestión dinámica nos permite adecuar los recursos empleados a sus múltiples fases de ejecución. De este modo, por ejemplo, es posible asignar un mayor número de recursos de cómputo en aquellas etapas en las que aumenta la carga computacional y liberarlos cuando dicha carga disminuye.

La gestión dinámica de recursos no solo permite optimizar el tiempo de ejecución del programa, sino también alcanzar un equilibrio entre este y el consumo energético. Para ello, resulta fundamental comprender cómo fluctúan dichos factores durante la ejecución del simulador. Tal y como se indicó en secciones previas, es posible configurar la aplicación para habilitar o deshabilitar la creación de puntos de control. En este apartado nos centraremos en un escenario donde no se generan dichos puntos, de modo que el peso de la E/S será muy pequeño y la carga de la aplicación será exclusivamente intensiva en CPU y comunicaciones. En consecuencia, la componente computacional (cuantificada directamente a través de los contadores de rendimiento energético) adquiere un mayor protagonismo.

La Fig. 4 muestra el consumo energético y los tiempos de ejecución por cada día simulado de la aplicación, al ser ejecutada con 16, 32, 64 y 128 procesos. Se puede observar que tanto la energía como el tiempo de ejecución varían a lo largo de las distintas fases de la simulación. Sin embargo, en determinados días simulados, concretamente entre el 40 y el 60, el gasto energético se mantiene prácticamente invariable, con independencia de la cantidad de recursos (procesos) empleados. Por otro lado, la Fig. 4b revela que, aun cuando no se aprecian alteraciones significativas en el consumo de energía, sí existen diferencias considerables en los tiempos de ejecución. Por consiguiente, nuestro objetivo es aprovechar aquellos días de simulación en los que el consumo energético resulta equiparable para todas las configuraciones, optando por aquella que proporcione los menores tiempos de ejecución.

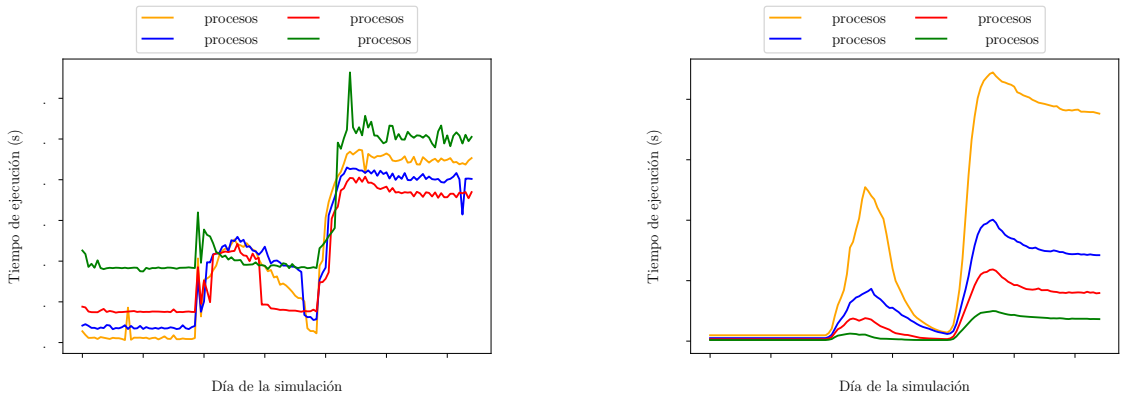
La Fig. 5 muestra el consumo energético y los tiempos de ejecución por iteración de la aplicación ejecutada con 1, 2, 3, 4, 5 y 6 nodos (con 128 procesos cada uno). En la Fig. 5a se observa que la energía consumida aumenta a medida que aumenta el número de nodos, y no podemos distinguir con claridad las distintas fases de la simulación a partir de dicho consumo. Sin embargo, la Fig. 5b ilustra cómo el uso de más recursos reduce significativamente el tiempo de ejecución. De hecho, tal y como evidencian tanto la Fig. 4b como la Fig. 5b, la escalabilidad de la aplicación es elevada únicamente durante aquellas fases con un uso intensivo de la CPU.

La Fig. 6a muestra el consumo energético acumulado a lo largo del tiempo de ejecución al evaluar la aplicación con una cantidad variable de procesos en un único nodo. Se observa que el tiempo de ejecución disminuye cuando aumenta el número de procesos. Sin embargo, cabe destacar que el gasto de energía no se incrementa con la cantidad de procesos; por el contrario, se aprecia una ligera disminución hasta alcanzar los 64 procesos. De igual modo, en la Fig. 6b se observa que el tiempo de ejecución disminuye a medida que aumenta el número de nodos. No obstante, es importante señalar que esta reducción temporal no es proporcional al aumento de nodos de cómputo, debido a que la aplicación carece de una escalabilidad ideal. Si analizamos las trazas correspondientes a 5 y 6 nodos, comprobamos que el consumo energético resulta mayor en la configuración de 6 nodos (Fig. 6b), mientras que el tiempo de ejecución se mantiene prácticamente inalterado (Fig. 5b).

Como siguiente paso en nuestro análisis, seleccionamos distintas reconfiguraciones para las pruebas y determinamos llevarlas a cabo en aquellos días simulados en los que el tiempo de ejecución presenta las mayores variaciones: los días 40, 60 y 80. Estas variaciones se pueden observar a nivel de número de procesos en la Fig. 4b y en la Fig. 5b a nivel de número de nodos. Durante la experimentación, modificamos de forma dinámica la cantidad de recursos que emplea la aplicación, ya sea a nivel de nodo (variando el número de procesos) o aumentando o reduciendo el número total de nodos. En todos los casos, empleamos una configuración de 128 procesos en ejecución por nodo.

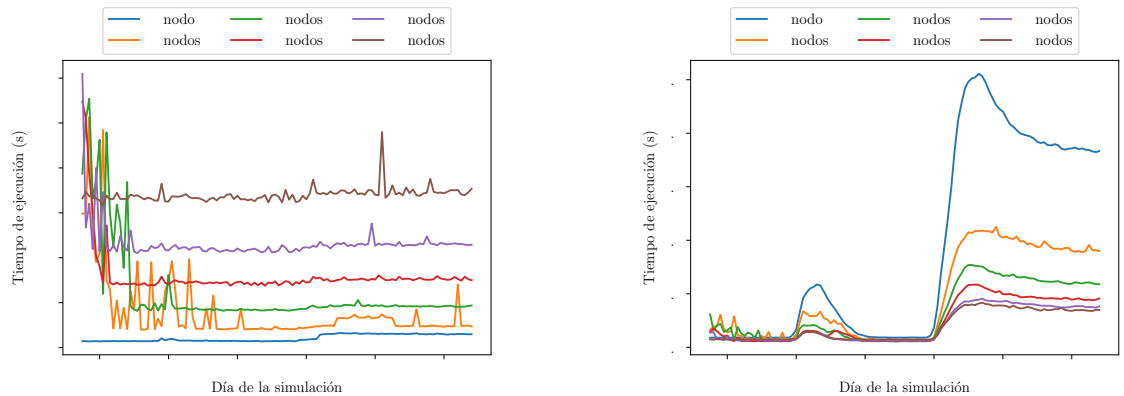
En la primera serie de pruebas, la aplicación se ejecuta en un único nodo y el número de procesos varía de forma dinámica. En el escenario base, la aplicación se ejecuta con 16 procesos. La Tabla I muestra los resultados de todas las pruebas realizadas. La prueba con ID 0 corresponde al escenario base (estático, sin reconfiguraciones) y $(\mathrm{n},\ \mathrm{p}) \xrightarrow{\mathrm{d}} (\mathrm{n}',\ \mathrm{p}')$ denota una reconfiguración realizada en el día de simulación d, pasando de la configuración original (n, p) a la configuración (n′, p′), donde n representa el número de nodos y p el número de procesos.

En esta tabla se observa que el tiempo de ejecución disminuye en todos los experimentos respecto al



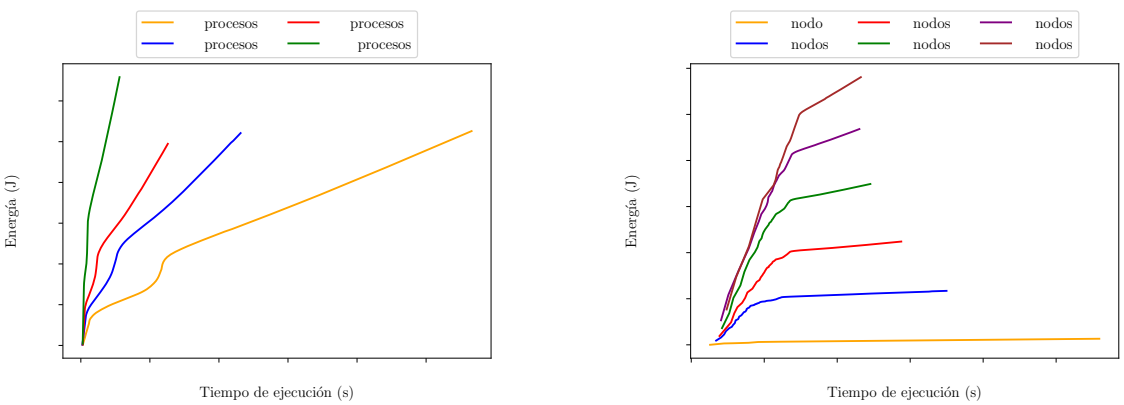
(a) Consumo energético. (b) Tiempo de ejecución.

Fig. 4: Consumo energético y tiempo de ejecución al ejecutar la aplicación con 16, 36, 68 y 128 procesos en un solo nodo con afinidad y sin puntos de control (*check-pointing*).



(a) Consumo energético. (b) Tiempo de ejecución.

Fig. 5: Consumo energético y tiempo de ejecución al ejecutar la aplicación en múltiples nodos con afinidad y sin puntos de control (*check-pointing*).



(a) Ejecución en un nodo con 16, 32, 68 y 128 procesos. (b) Ejecución en múltiples nodos con 128 procesos por nodo.

Fig. 6: Consumo energético acumulado a lo largo del tiempo de ejecución sin puntos de control (*check-pointing*).

escenario base. Este resultado es el esperado, ya que dicho escenario base corresponde al caso en el que se emplean menos recursos y la aplicación presenta una buena escalabilidad. Esto implica que incrementar el número de procesos se traduce en menor tiempo de ejecución en las fases con mayor carga de CPU. El consumo energético también disminuye en todos los experimentos excepto en uno (ID 1), en el cual se optó por asignar el número máximo de procesos al inicio de la variante Delta de la COVID-19 para el resto de la simulación. La mayor reducción de energía se obtiene en el experimento con ID 2, en el que la aplicación se reconfigura para ejecutarse con 64 procesos tras el comienzo de la variante Delta. El motivo por el cual este experimento logra la mayor reducción energética es que se alcanza un compromiso entre el tiempo de ejecución y el consumo de energía.

En la segunda serie de pruebas, la aplicación se re-

Tabla I: Evaluación de la reconfiguración dinámica de procesos al ejecutar la aplicación en un solo nodo.

ID	Tiempo de ejecución (s)	Energía (J)	Reconfiguraciones (n, p) $\xrightarrow{\text{día}}$ (n, p)
0	**1.125,34**	**1.051**	**(1, 16)**
1	116,12	1.158	(1, 16) $\xrightarrow{40}$ (1, 128)
2	247,34	924	(1, 16) $\xrightarrow{40}$ (1, 64)
3	448,85	1.019	(1, 16) $\xrightarrow{40}$ (1, 32)
4	137,65	1.109	(1, 16) $\xrightarrow{40}$ (1, 64) $\xrightarrow{80}$ (1, 128)
5	236,99	1.027	(1, 16) $\xrightarrow{40}$ (1, 128) $\xrightarrow{80}$ (1, 64)

Tabla II: Evaluación de la reconfiguración dinámica de la aplicación al ejecutarla en múltiples nodos.

Id	Tiempo de ejecución (s)	Energía (J)	Reconfiguraciones (n, p) $\xrightarrow{\text{día}}$ (n, p)
6	**106,32**	**1.318**	**(1, 128)**
7	44,49	53.336	(1, 128) $\xrightarrow{40}$ (5, 640)
8	67,60	124.185	(1, 128) $\xrightarrow{40}$ (4, 512) $\xrightarrow{80}$ (5, 640)
9	66,15	118.414	(1, 128) $\xrightarrow{40}$ (4, 512) $\xrightarrow{60}$ (1, 128) $\xrightarrow{80}$ (5, 640)

configura dinámicamente utilizando múltiples nodos de cómputo. Partimos del escenario base (ID 6), en el que la aplicación se ejecuta en un solo nodo con 128 procesos. La Tabla II muestra los resultados de todas las pruebas realizadas. En estos experimentos, también observamos que el tiempo de ejecución se redujo en todos los casos. Sin embargo, se aprecia que el incremento del consumo energético es considerablemente mayor. Esto se debe tanto al aumento significativamente mayor en el número de recursos asignados como a los costes de reconfiguración. Por estas razones, se observa que los experimentos con ID 8 y 9 presentan el mayor consumo de energía. No obstante, el experimento 9 muestra un consumo ligeramente inferior al del experimento 8, a pesar de contar con una reconfiguración adicional. Esto se explica porque dicho cambio se efectúa en el día de simulación 60, entre las dos olas de infección (fases con uso intensivo de CPU) de la simulación. Durante estos días simulados, la carga de trabajo computacional es menor, por lo que el tiempo de ejecución se mantiene prácticamente inalterado, independientemente del número de nodos asignados. Por ello, al reducir el número de nodos, el tiempo de ejecución no disminuye, pero se logra un ahorro energético gracias al menor número de recursos empleados.

V. Conclusiones y trabajos futuros

En este trabajo, hemos estudiado el rendimiento energético de un simulador epidemiológico paralelo basado en agentes. El estudio del perfil energético consistió en analizar el comportamiento de la aplicación al utilizar distintas políticas de afinidad entre procesos y núcleos. Esto nos permitió estudiar cómo varían el consumo energético y el tiempo de ejecución, así como analizar la relación entre las distintas políticas de afinidad y el rendimiento de la aplicación con diversas configuraciones. Los experimentos demuestran que ejecutar la aplicación con afinidad reduce considerablemente tanto el consumo de energía como el tiempo de ejecución. Además, en el análisis de la frecuencia de los núcleos, observamos que al emplear 64 y 128 procesos las frecuencias se reducen conforme aumenta la carga computacional para evitar alcanzar el límite de potencia del procesador.

A partir del perfil energético obtenido, hemos abordado la eficiencia energética mediante una propuesta de reconfiguración dinámica de recursos en tiempo de ejecución, con el fin de adaptar el número de procesos a la fase específica de la aplicación en ejecución. Los experimentos han demostrado que la reconfiguración dinámica de recursos durante la ejecución permite mejorar tanto el tiempo de ejecución como el consumo energético.

El análisis y los resultados de la reducción del consumo energético y del tiempo de ejecución lograda mediante el uso de la afinidad, se pueden extrapolar a otras aplicaciones, aunque esto debería confirmarse realizando el mismo estudio con una aplicación diferente y en un entorno distinto. Del mismo modo, la optimización de estos factores también podría generalizarse ajustando dinámicamente los recursos de la aplicación, aunque los beneficios y las reconfiguraciones dependerían de la naturaleza de la aplicación.

Como trabajo futuro, planeamos integrar la gestión dinámica de recursos para considerar factores que van más allá del consumo energético y el tiempo total de ejecución, tales como los tiempos de CPU, E/S, comunicaciones y el uso del *hardware*. Asimismo, pretendemos incorporar la energía en una metodología más general, en la que las reconfiguraciones se determinen de forma dinámica en tiempo de ejecución basándose en diversas políticas y modelos. Esto permitiría aplicar la metodología a otras aplicaciones y *workflows*.

Agradecimientos

Este trabajo ha sido parcialmente financiado por la Agencia Española de Investigación a través del proyecto “New scalable I/O techniques for hybrid HPC and data-intensive workloads (SCIOT)” con referencia PID2022-138050NB-I00, la ayuda “Ayudas para la contratación de personal investigador predoctoral en formación para el año

2024" con referencia PIPF-2024/COM-34495 financiada por la Comunidad de Madrid y el proyecto de I+D+i PID2022-138050NB-I00, financiado por ICIU/AEI/10.13039/501100011033/"FEDER Una manera de hacer Europa".

Referencias

[1] Pei Ling Chia, "Accelerating HPC Industry Innovation," Available at: https://www.asianscientist.com/2024/10/print/accelerating-hpc-industry-innovation/, Oct. 2024, Accessed: 2026-03-25.

[2] Philippe Olivier Alexandre Navaux, Arthur Francisco Lorenzon, and Matheus da Silva Serpa, "Challenges in High-Performance Computing," *Journal of the Brazilian Computer Society*, vol. 29, no. 1, pp. 51–62, Aug. 2023.

[3] John Jumper et al., "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, no. 7873, pp. 583–589, Aug. 2021.

[4] P. Michael and K. Jackson, "Advancing Scientific Discovery: A High Performance Computing Architectures for AI and Machine Learning," *Journal of Integrated VLSI, Embedded and Computing Technologies*, vol. 2, no. 2, pp. 18–26, Apr. 2025.

[5] Stefanie Hölbling, Gottfried Kirchengast, Christian Briese, and Hans Thüminger, "Energy use and carbon emissions in high-performance computing: A case study for universities and reduction strategies," *Cleaner Environmental Systems*, vol. 19, pp. 100332, Dec. 2025.

[6] M Minges and World Bank, "Digital infrastructure: There is a continual need for upgrading and greening," *Digital progress and trends report*, p. 45–64, 2023.

[7] David E Singh et al., "Evaluation of vaccination strategies for the metropolitan area of Madrid via agent-based simulation," *BMJ Open*, vol. 12, no. 12, pp. e065937, 2022.

[8] "C3 Supercomputer," Available at: https://c3-uc3m.github.io/c3-web/, Accessed: 2025-11-21.

[9] Howard David, Eugene Gorbatov, Ulf R. Hanebutte, Rahul Khanna, and Christian Le, "RAPL: memory power estimation and capping," in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, Austin Texas USA, Aug. 2010, p. 189–194, ACM.

[10] Guillaume Fieni, Daniel Romero Acero, Pierre Rust, and Romain Rouvoy, "PowerAPI: A Python framework for building software-defined power meters," *Journal of Open Source Software*, vol. 9, no. 98, pp. 6670, 2024.

[11] Kajol Kulkarni et al., "Harvesting energy consumption on European HPC systems: Sharing Experience from the CEEC project," *arXiv preprint arXiv:2511.03029*, 2025.

[12] Vincent M. Weaver et al., "Measuring Energy and Power with PAPI," in *2012 41st International Conference on Parallel Processing Workshops*, 2012, p. 262–268.

[13] Intel, HPE, NEC, and Dell, "IPMI Specification v2.0," .

[14] Richard Kavanagh, Django Armstrong, and Karim Djemame, "Accuracy of Energy Model Calibration with IPMI," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, 2016, p. 648–655.

[15] Steven J Martin and Matthew Kappel, "Cray XC30 power monitoring and management," in *Cray User Group Conference Proceedings*, 2014.

[16] Steven J. Martin, "HPE Cray Power Monitoring (PM) Counters," May 2024.

[17] "Power Capping Framework — The Linux Kernel documentation," Available at: https://www.kernel.org/doc/html/latest/power/powercap/powercap.html, Accessed: 2025-11-08.

[18] Jan Treibig, Georg Hager, and Gerhard Wellein, "LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments," in *2010 39th International Conference on Parallel Processing Workshops*, 2010, p. 207–216.

[19] Thomas Röhl, Jan Treibig, Georg Hager, and Gerhard Wellein, "Overhead Analysis of Performance Counter Measurements," 2014, p. 176–185.

[20] "LIKWID Tools," Available at: https://github.com/RRZE-HPC/likwid/wiki/Home, Accessed: 2025-11-09.

[21] "Perf: Linux profiling with performance counters," Available at: https://perfwiki.github.io/main/, Accessed: 2025-11-09.

[22] Gabi, "Perf-icient Measuring of Laptop Energy Consumption and CO2e," Available at: https://medium.com/@gabibeyer/perf-icient-measuring-of-laptop-energy-consumption-and-co2e-0a36d3a6655a, Dec. 2023, Accessed: 2025-11-09.

[23] Mathilde Jay, Vladimir Ostapenco, Laurent Lefevre, Denis Trystram, Anne-Cécile Orgerie, and Benjamin Fichel, "An experimental comparison of software-based power meters: focus on CPU and GPU," in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, May 2023, p. 106–118.

[24] Guillaume Raffin and Denis Trystram, "Dissecting the software-based measurement of CPU energy consumption: a comparative analysis," , no. arXiv:2401.15985, arXiv:2401.15985.

[25] Alberto Cascajo, Alvaro Arbe, Javier Garcia-Blas, Jesus Carretero, and David E. Singh, *Malleable Techniques and Resource Scheduling to Improve Energy Efficiency in Parallel Applications*, vol. 13999, p. 16–27, Springer Nature Switzerland, Cham, 2023.

[26] Manuel Rodríguez-Gonzalo, David E. Singh, Javier García Blas, and Jesús Carretero, "Improving the Energy Efficiency of MPI Applications by Means of Malleability," in *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, Feb. 2016, p. 627–634.

[27] Eishi Arima, Isaías A. Comprés, and Martin Schulz, "On the Convergence of Malleability and the HPC PowerStack: Exploiting Dynamism in Over-Provisioned and Power-Constrained HPC Systems," , no. arXiv:2405.03847, May 2024, arXiv:2405.03847.

[28] Briag Dupont, Nesryne Mejri, and Georges Da Costa, "Energy-aware scheduling of malleable HPC applications using a Particle Swarm optimised greedy algorithm," *Sustainable Computing: Informatics and Systems*, vol. 28, pp. 100447, Dec. 2020.

[29] Paula Sanchez-Checa, Javier Garcia-Blas, Jesus Carretero, and David E. Singh, "Combining Malleability and Distributed Control Mechanisms to Reduce I/O Contention," in *ISC High Performance 2025 International Workshops*, 2025.

[30] Paula Sanchez-Checa, Genaro Sanchez-Gallegos, Javier Garcia-Blas, Jesus Carretero, and David E. Singh, "Comparative Analysis of Algorithms for Malleability Decision-Making in Applications and File Systems," in *Euro-Par 2025: Parallel Processing Workshops*. 2025, Springer Nature Switzerland.

[31] Aymar Cublier Martínez, Jesús Carretero, and David E. Singh, "Detailed parallel social modeling for the analysis of COVID-19 spread," *The Journal of Supercomputing*, vol. 80, no. 9, pp. 12408–12429, 2024.

[32] Miguel Guzman Merino, Maria-Cristina Marinescu, Alberto Cascajo, Jesus Carretero, and David E. Singh, "Evaluating the spread of Omicron COVID-19 variant in Spain," *Future Generation Computer Systems*, vol. 149, pp. 547–561, Dec. 2023.

[33] Gonzalo Martín, Maria-Cristina Marinescu, David E. Singh, and Jesús Carretero, "FLEX-MPI: An MPI Extension for Supporting Dynamic Load Balancing on Heterogeneous Non-dedicated Systems," in *Euro-Par 2013 Parallel Processing*, Wolf, Felix and Mohr, Bernd and an Mey, Dieter, Ed., Berlin, Heidelberg, 2013, p. 138–149, Springer.

[34] "AMD EPYC™ 7713," Available at: https://www.amd.com/en/products/processors/server/epyc/7003-series/amd-epyc-7713.html, Accessed: 2025-11-15.

[35] "AMD RAPL PowerCap Patches Updated For Linux - Now Include Family 19h (Zen 3)," Available at: https://www.phoronix.com/news/AMD-RAPL-Linux-Now-19h, Accessed: 2025-11-08.

[36] Kajol Kulkarni et al., "Harvesting energy consumption on European HPC systems: Sharing Experience from the CEEC project," , no. arXiv:2511.03029, Nov. 2025, arXiv:2511.03029.

Parte 2.- X Jornadas de Computación Empotrada y Reconfigurable

Aplicaciones de los sistemas empotrados y reconfigurables

Comparativa entre transformada de Fourier de tiempo corto y sonogramas neuromórficos en audio para mantenimiento predictivo con redes de convolución

Juan M. Montes-Sánchez[1], Plácido Fernández Cuevas[1], Carlos Rangel Cascajosa[1], Juan P. Domínguez-Morales[1], Saturnino Vicente Díaz[1] y Ángel Jiménez Fernández[1]

Resumen— **Los algoritmos de inteligencia artificial (IA) son muy útiles en tareas de clasificación de señales. En algunas implementaciones de mantenimiento predictivo (PdM), las señales de audio pueden utilizarse como entrada para diagnosticar fallos en dispositivos industriales y anticipar problemas futuros. La Transformada de Fourier de Tiempo Corto (STFT) sigue siendo la técnica de extracción de características más utilizada para señales de audio. No obstante, en los últimos años se han desarrollado otros métodos alternativos basados en la biología. En este estudio, se presenta una validación de un conjunto de datos de audio de PdM procesado con dos métodos diferentes de extracción de características: imágenes de espectrogramas STFT e imágenes de sonogramas neuromórficos. Estas últimas se obtuvieron utilizando el sensor auditivo neuromórfico (NAS) y el software pyNAVIS en un trabajo anterior. Se han entrenado modelos de IA utilizando la misma arquitectura de red neuronal convolucional (CNN) para comparar estos dos métodos en este escenario PdM. Los resultados mostraron que, aunque el rendimiento fue ligeramente mejor utilizando el espectrograma STFT, el método alternativo usando el NAS utiliza potencialmente menos recursos computacionales, al tiempo que mantiene excelentes métricas de clasificación.**

Palabras clave— **Espectrograma, Mantenimiento Predictivo, Audio, Ingeniería Neuromórfica, CNN.**

I. Introducción

ACTUALMENTE vivimos en la era de la Inteligencia Artificial (IA). Aunque el campo de la IA no es nada nuevo [1], el florecimiento de nuevos métodos y aplicaciones basados en IA en la última década ha sido espectacular, y aún continúa. Nuevos conceptos como la IA en el borde (Edge AI) [2] o nuevas aplicaciones industriales basadas en IA para Mantenimiento Predictivo (PdM) [3] también están en auge, creando nuevas oportunidades para investigadores de todo el mundo, actuando a veces como una alternativa a otros modelos basados en IA de alto consumo energético.

En tareas de clasificación de señales también se utilizan muchos algoritmos de IA. Estos algoritmos a menudo requieren un cierto preprocesamiento de las señales de entrada para extraer sus principales características antes de poder clasificarlas correctamente. Es muy común encontrar trabajos académicos que utilizan la Transformada de Fourier (FT) en muchas de sus variantes para extraer dichas características de una señal [4]. Dado que la expresión matemática de la FT continua es difícil de calcular por su coste computacional, las computadoras utilizan otros métodos para obtener, en su lugar, la Transformada Discreta de Fourier (DFT), ya que trabajan en el dominio discreto. La Transformada Rápida de Fourier (FFT, por sus siglas en inglés) es una familia de algoritmos que constituye la forma más común de obtener la DFT de una señal, ya que requieren menos cálculos y más simples en comparación con un cálculo directo de la DFT. La técnica que aplica la FFT a segmentos secuenciales de una señal, obteniendo cómo varía la frecuencia de dicha señal a lo largo del tiempo, se denomina Transformada de Fourier de Tiempo Corto (STFT, por sus siglas en inglés). Este es el método de extracción de características más utilizado para audio, aunque existen muchos otros [5].

En la literatura podemos encontrar algunos ejemplos recientes del uso de estos métodos para extraer características en tareas de clasificación de sonidos que convierten el audio en imágenes de espectrogramas STFT para su posterior clasificación mediante redes neuronales convolucionales (CNN) [6].

Para crear estas imágenes de espectrograma, el número de operaciones necesarias depende de la frecuencia de muestreo de la señal y del número de puntos de la FFT. Por lo tanto, en algunas aplicaciones, utilizar este método de extracción de características podría requerir mucha más potencia de cálculo de la que la mayoría de los sistemas empotrados de bajo consumo disponen.

Esta es una de las razones detrás del desarrollo de métodos alternativos para la extracción de características que imitan la forma en que la biología procesa las señales de audio. Estos métodos se conocen generalmente como ingeniería neuromórfica. Nuestro cerebro procesa la información de una manera muy robusta y energéticamente eficiente, por lo que muchos investigadores han estudiado cómo creemos que se lleva a cabo este proceso y han desarrollado muchos sensores neuromórficos, como retinas para la visión [7], [8] y cócleas para el audio [9], [10]. En la última década, el procesamiento de audio neuromórfico ha demostrado superar a su contraparte tradicional en varios escenarios, ya no solo en consumo de energía y latencia, sino también en precisión

[1]Robotics and Technology of Computers Lab., ETSII-EPS, Universidad de Sevilla, Sevilla, Spain, e-mail: {juanmanuelmontes, pfernandez9, crangel, jpdominguez, satur, angel}@us.es.

de clasificación y rendimiento general [11], [12], [13], [14], [15].

Sin embargo, muchas aplicaciones específicas para estos métodos permanecen sin haber sido estudiadas, debido a que estos sensores no están tan ampliamente disponibles como otros métodos. En el campo del Mantenimiento Predictivo (PdM), donde el audio es una de las señales de entrada más usadas en aplicaciones de clasificación de fallos en motores [16], [17], [18], [19], [20], no hay autores que estén trabajando activamente con sensores neuromórficos, según nuestros datos.

Esta es la razón por la que se elaboró un conjunto de datos de audio neuromórfico para PdM, que está disponible públicamente [21], y que fue presentado durante el congreso ISCAS 2025 [22]. Ese trabajo fue una continuación de otro primer enfoque sobre PdM con sensores inerciales (sin utilizar audio ni sensores neuromórficos) [23]. El presente trabajo es una actualización sobre la validación del conjunto de datos neuromórfico ya presentada, esta vez en comparación con el método de extracción de imágenes de espectrograma STFT. Sabemos que este método basado en STFT ha sido utilizado para la clasificación de señales de audio en PdM por otros autores [24].

II. Materiales y métodos

El conjunto de datos de audio neuromórfico para PdM contiene muestras de audio mono para dos escenarios de fallo diferentes en bombas peristálticas. Cada muestra está etiquetada y diferenciada de las demás. El primer escenario (*ESCENARIO DE ENVEJECIMIENTO* o *AGING*) contiene datos recogidos a partir de bombas nuevas y de bombas *viejas* (esto es, con su vida útil recomendada ya expirada), mientras que el segundo escenario (*ESCENARIO DE BURBUJAS* o *BUBBLE*) contiene grabaciones de una bomba nueva en funcionamiento normal junto con esa misma bomba en una situación anómala en la que entran burbujas de aire en el circuito hidráulico, mezclándose con el líquido.

Estas grabaciones se han realizado utilizando piezas de un equipo de análisis biomédico comercial, concretamente un bloque hidráulico de bombas peristálticas y electroválvulas. El funcionamiento del dispositivo, así como el proceso de grabación, se encuentran publicados en trabajos anteriores [23], [25].

Las grabaciones de audio han sido posteriormente procesadas con el sensor auditivo neuromórfico (NAS) [10]. Este sensor se puede implementar en varios modelos de chips FPGA de lógica programable utilizando la herramienta OpenNAS [26]. Para elaborar el conjunto de datos que se ha usado en este trabajo, se eligió una placa de evaluación *ZTEX USB-FPGA Module 2.13* con electrónica adicional para capturar audio (placa de micrófonos). Se eligió una configuración mono y 64 canales en la distribución por defecto de frecuencias. La caracterización resultante de la configuración del sensor NAS utilizado para esta extracción de características neuromórfica puede verse en la Figura 1.

Cuando recibe una entrada de señal de audio, el sensor NAS genera a su salida pulsos asociados a cada canal codificados en formato *Address Event Representation* (AER). Estos datos se pueden capturar desde un PC utilizando la placa *USB AER mini* en conjunto con las librerías *jAER* [27].

El conjunto hardware formado por la placa FPGA con NAS, la interfaz de conexión USB y algunos elementos adicionales se puede ver en la Figura 2.

Las grabaciones para ambos escenarios se presentan en tres formatos diferentes: archivos de audio *WAV*, archivos neuromórficos *AEDAT* generados con el NAS, e imágenes de sonogramas neuromórficos de 0,5 s cada una obtenidas a partir de los archivos AEDAT mediante el software pyNAVIS [28].

En el trabajo previo [22], las imágenes de sonograma se validaron realizando una clasificación usando una CNN entrenada con este mismo conjunto de datos. La arquitectura seleccionada de esa CNN fue la popular *DenseNet-121* [29], que ofrece un rendimiento muy alto en tareas de clasificación de imágenes.

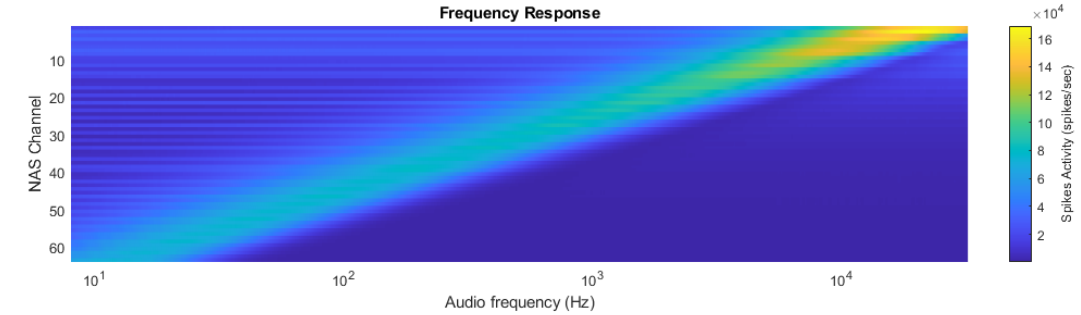
Sin embargo, solo con esa prueba de validación, no teníamos ninguna otra referencia para comparar la calidad de este método de extracción con otras alternativas. El método más similar sin utilizar ingeniería neuromórfica sería la creación de un conjunto de datos de imágenes equivalente usando la STFT para obtener imágenes de espectrograma.

Para llevar a cabo la comparación, utilizamos las librerías FFT integradas en Matlab para crear imágenes de espectrograma de 0,5 s cada una, con un solapamiento del 50 % (misma duración y solapamiento que las imágenes de sonograma del NAS con pyNAVIS). Se crearon dos conjuntos de datos de espectrograma diferentes para cada escenario: uno con un número bajo de puntos FFT (128 puntos FFT) y un segundo con un número mayor de puntos FFT (1024 puntos FFT). En la Figura 3 se muestra un ejemplo de los resultados de los tres métodos.

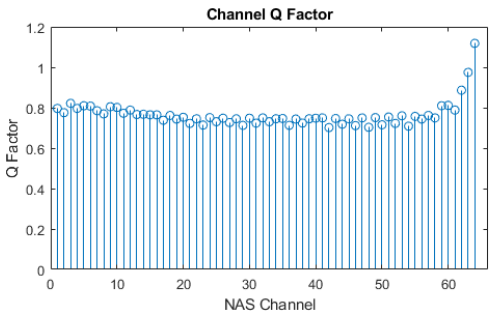
Se seleccionó la arquitectura *DenseNet121* y se entrenó con cada conjunto de datos (sonograma, FFT 128 y FFT 1024) y para cada escenario, siempre utilizando los mismos hiperparámetros para cada sesión de entrenamiento: una tasa de aprendizaje de 0,002 y un tamaño de lote de 32. Cada conjunto de datos se dividió aleatoriamente en 80 % para entrenamiento y 20 % para validación. El equilibrio entre clases se mantuvo durante esta división.

III. Resultados experimentales

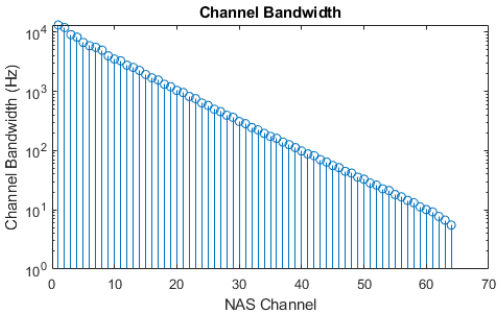
El conjunto de validación se utilizó para evaluar el rendimiento de los modelos entrenados. Se obtuvieron las siguientes métricas, que son ampliamente utilizadas en la literatura para la comparación de redes: precisión, exhaustividad (*recall* en inglés) y puntuación F1 (*F1 score*) para cada clase [30]. También determinamos la media ponderada de F1: un promedio de los tres valores de puntuación F1 de nuestro modelo (correspondientes a cada clase) que también tiene en cuenta el número de ocurrencias de la clase. Los resultados se enumeran en la Tabla I. La colum-



(a) Respuesta en frecuencia



(b) Factor de calidad Q



(c) Ancho de banda

Fig. 1: Respuesta en frecuencia (a), factor de calidad Q (b) y ancho de banda (c) del sensor NAS para cada uno de sus 64 canales.

na de soporte de la tabla muestra cuántas imágenes estaban presentes para cada clase en el conjunto de validación.

IV. Conclusiones

Los resultados de clasificación muestran que el método del espectrograma es solo ligeramente superior al método del sonograma NAS en términos de puntuación F1. De hecho, el rendimiento en el caso del escenario de envejecimiento (*AGING*) es prácticamente idéntico en las tres opciones comparadas. Para el escenario de burbujas (*BUBBLE*), el espectrograma de 1024 puntos FFT supera ligeramente a las otras dos opciones, que tuvieron algunos problemas para diferenciar la clase *NORMAL* de la clase *BUBBLE* (con burbujas de aire). Este suceso es promotedor, ya que demuestra que podría llegarse a una clasificación adecuada de este conjunto de datos si se mejora la resolución en el proceso de extracción de características. Esto no estaba claro en el estudio previo, ya que solo contemplaba los resultados de clasificación para el método basado en sonogramas.

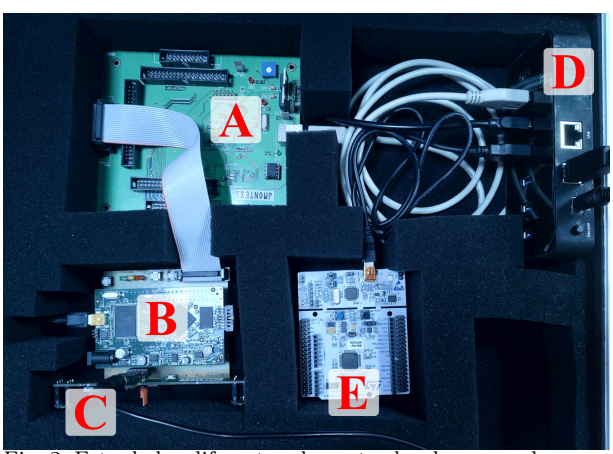


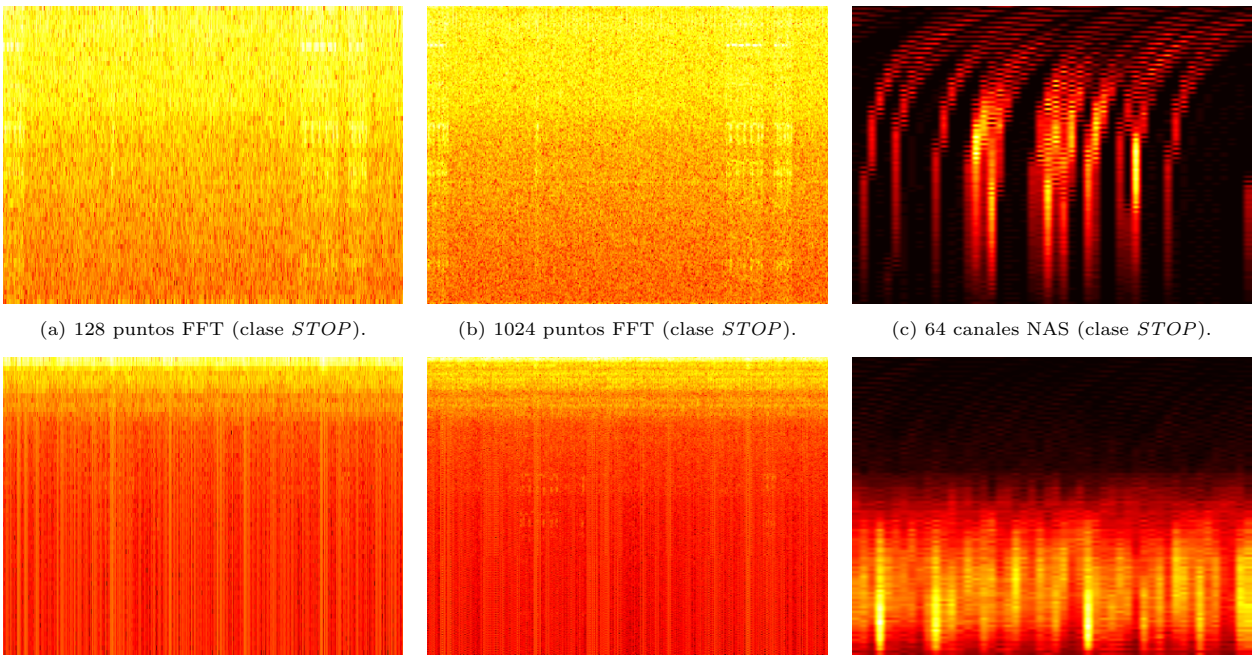
Fig. 2: Foto de los diferentes elementos hardware usados para la conversión de audio a archivos AEDAT, conectados entre sí. **A** es la placa *USB AER mini*, **B** es el conjunto FPGA con entrada de audio *ZTEX*, **C** es la placa de micrófonos con la entrada de audio jack, **D** es un HUB USB comercial, y **E** es una placa *STM32 NUCLEO* auxiliar que no se ha utilizado en el presente trabajo.

Incluso si estos resultados sugieren que el método del sonograma NAS podría ser inferior en términos de valor F1, es importante señalar que la extracción de características es casi inmediata cuando se realiza con el NAS, utilizando además muy pocos recursos computacionales y energéticos, mientras que el método STFT generalmente necesita más cantidad de estos recursos. Por lo tanto, y sobre todo si nos fijamos en el escenario de envejecimiento, usar el NAS de 64 canales llegaría a darnos los mismos resultados pero con una mayor eficiencia. La mejora exacta en el rendimiento también se debería calcular en trabajos futuros, obteniendo una medición precisa tanto del coste energético como el computacional para cada enfoque.

El NAS utilizado en este estudio se ha configurado con un número de canales pequeño (64), pero es posible aumentar este número sin que el impacto en recursos sea apreciable. Como se ha observado una mejora en la clasificación en el escenario de burbujas entre el método STFT de 128 puntos y el de 1024 puntos, se debería estudiar también si esta mejora se observa también al aumentar el número de canales del NAS.

Mejorar el conjunto de datos de burbujas añadiendo más muestras es otra opción que valdría la pena probar, ya que podría ayudar a obtener mejores resultados de clasificación.

Sobre la arquitectura de red utilizada, la selección de DenseNet-121 sobre variantes más profundas de



(a) 128 puntos FFT (clase *STOP*). (b) 1024 puntos FFT (clase *STOP*). (c) 64 canales NAS (clase *STOP*).

(d) 128 puntos FFT (clase *NEW*). (e) 1024 puntos FFT (clase *NEW*). (f) 64 canales NAS (clase *NEW*)

Fig. 3: Ejemplos de sonogramas NAS y espectrogramas FFT para ventanas de audio extraídas del conjunto de datos de envejecimiento y convertidas en imágenes. Las subfiguras (a), (b) y (c) se obtuvieron de la misma ventana de audio de 0,5s con la clase correspondiente *STOP*. Las subfiguras (d), (e) y (f) se obtuvieron de la misma ventana de audio de 0,5s con la clase correspondiente *NEW*.

Tabla I: Resultados de clasificación de sonogramas y espectrogramas.

Método	Escenario	Clase	Precisión	Exhaustividad	F1 score	Soporte	Media ponderada de F1
Sonograma NAS (64 canales)	AGING	STOP	1,00	1,00	1,00	106	0,99
		NEW	1,00	0,95	0,98	83	
		OLD	0,97	1,00	0,99	139	
	BUBBLE	STOP	1,00	1,00	1,00	78	0,71
		NORMAL	0,60	0,44	0,51	82	
		BUBBLE	0,59	0,73	0,65	90	
Espectrograma (128 puntos FFT)	AGING	STOP	1,00	1,00	1,00	106	0,99
		NEW	0,98	0,99	0,98	83	
		OLD	0,99	0,99	0,99	141	
	BUBBLE	STOP	1,00	1,00	1,00	76	0,78
		NORMAL	0,64	0,69	0,66	80	
		BUBBLE	0,72	0,67	0,70	95	
Espectrograma (1024 puntos FFT)	AGING	STOP	1,00	1,00	1,00	106	0,99
		NEW	0,99	0,99	0,99	83	
		OLD	0,99	0,99	0,99	141	
	BUBBLE	STOP	1,00	1,00	1,00	76	0,83
		NORMAL	0,74	0,74	0,74	80	
		BUBBLE	0,78	0,78	0,78	95	

DenseNet y otras arquitecturas (VGG [31], ResNet [32], EfficientNet [33] o MobileNet [34]) se debió a que esta primera ofrece una alta precisión con un costo computacional relativamente bajo, lo que permite un entrenamiento rápido en hardware de sobremesa estándar. Los otros modelos alternativos citados, u otros similares, se podrían explorar también en trabajos futuros para ver si alguno mejora el rendimiento de la clasificación en el escenario de burbujas, que parece ser el que mayor reto presenta.

Para finalizar, ahora que el conjunto de datos se puede considerar validado, creemos importante cambiar el proceso de clasificación en el caso de los datos neuromórficos. Resulta obvio que, para beneficiarse realmente de la naturaleza neuromórfica del NAS, la clasificación debería realizarse en este mismo dominio pulsante utilizando una Red Neuronal Pulsante (SNN, por sus siglas en inglés). De usar este enfoque, la entrada de audio podría enviarse directamente procesada desde el NAS a la red sin el paso de conversión a imágenes de sonograma.

Agradecimientos

Este trabajo ha recibido financiación de los proyectos PREDICAR (ref. PID2023-149777OB-I00), NASAI (PDC2023-145841-C33) y NEKOR (PID2023-149071NB-C54) del Ministerio de Ciencia e Innovación (MCIN), Agencia Estatal de Investigación (AEI), Gobierno de España.

Las tareas de investigación del autor P. Fernández Cuevas están financiadas por un contrato de formación del plan "VII PPIT-US"de la Universidad de Sevilla.

El conjunto de datos utilizado en el estudio [23] fue grabado con apoyo de la Universidad de Tokushima bajo la dirección de los profesores Yoko Uwate y Yoshifumi Nishio en una estancia de investigación del autor Juan M. Montes Sánchez en junio de 2023.

Referencias

[1] Attila Benko and Cecília Sik Lányi, "History of artificial intelligence," in *Encyclopedia of Information Science and Technology, Second Edition*, pp. 1759–1762. IGI global, 2009.

[2] Mohammad Abidur Rahman, Farhan Shahrior, Kamran Iqbal, and Ali A Abushaiba, "Enabling Intelligent Industrial Automation: A Review of Machine Learning Applications with Digital Twin and Edge AI Integration," *Automation 2025, Vol. 6, Page 37*, vol. 6, no. 3, pp. 37, 8 2025.

[3] Anthony D. Scaife, "Improve predictive maintenance through the application of artificial intelligence: A systematic review," *Results in Engineering*, vol. 21, pp. 101645, 3 2024.

[4] Khalid Zaman, Melike Sah, Cem Direkoglu, and Masashi Unoki, "A Survey of Audio Classification Using Deep Learning," *IEEE Access*, vol. 11, pp. 106620–106649, 2023.

[5] Francesc Alías, Joan Claudi Socoró, and Xavier Sevillano, "A review of physical and perceptual feature extraction techniques for speech, music and environmental sounds," *Applied Sciences*, vol. 6, no. 5, 2016.

[6] Nur Fatinah Md Hafiz, Syamsiah Mashohor, Mohammad Habib Shah Ershad Mohd Azrul Shazril, Azizi Mohd Ali, and Mohd Fadlee A. Rasid, "Machine Learning Framework for Industrial Machine Sound Classification in Predictive Maintenance," *IEEE Access*, 2025.

[7] Teresa Serrano-Gotarredona and Bernabé Linares-Barranco, "A 128x128 1.5 % contrast sensitivity 0.9 % fpn 3 us latency 4 mw asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 3, pp. 827–838, 2013.

[8] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck, "A 128x128 120 db 15 us latency asynchronous temporal contrast vision sensor," *IEEE journal of solid-state circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[9] Vincent Chan, Shih-Chii Liu, and Andr van Schaik, "Aer ear: A matched silicon cochlea pair with address event representation interface," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 1, pp. 48–59, 2007.

[10] Angel Jiménez-Fernández, Elena Cerezuela-Escudero, Lourdes Miró-Amarante, Manuel Jesus Domínguez-Morales, Francisco de Asís Gómez-Rodríguez, Alejandro Linares-Barranco, and Gabriel Jiménez-Moreno, "A binaural neuromorphic auditory sensor for fpga: a spike signal processing approach," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 4, pp. 804–818, 2016.

[11] Juan P Dominguez-Morales, Angel F Jimenez-Fernandez, Manuel J Dominguez-Morales, and Gabriel Jimenez-Moreno, "Deep neural networks for the recognition and classification of heart murmurs using neuromorphic auditory sensors," *IEEE transactions on biomedical circuits and systems*, vol. 12, no. 1, pp. 24–34, 2017.

[12] Jithendar Anumula, Daniel Neil, Tobi Delbruck, and Shih-Chii Liu, "Feature representations for neuromorphic audio spike streams," *Frontiers in neuroscience*, vol. 12, pp. 23, 2018.

[13] Juan P Dominguez-Morales, Stefano Buccelli, Daniel Gutierrez-Galan, Ilaria Colombi, Angel Jimenez-Fernandez, and Michela Chiappalone, "Real-time detection of bursts in neuronal cultures using a neuromorphic auditory sensor and spiking neural networks," *Neurocomputing*, vol. 449, pp. 422–434, 2021.

[14] Marco Rasetto, Juan P Dominguez-Morales, Angel Jimenez-Fernandez, and Ryad Benosman, "Event based time-vectors for auditory features extraction: a neuromorphic approach for low power audio recognition," *arXiv preprint arXiv:2112.07011*, 2021.

[15] Thorben Schoepe, Daniel Gutierrez-Galan, Juan P Dominguez-Morales, Hugh Greatorex, Angel Jimenez-Fernandez, Alejandro Linares-Barranco, and Elisabetta Chicca, "Closed-loop sound source localization in neuromorphic systems," *Neuromorphic Computing and Engineering*, vol. 3, no. 2, pp. 024009, 2023.

[16] Rommel Stiward Prieto Estacio, Diego Alberto Bravo Montenegro, and Carlos Felipe Rengifo Rodas, "Dataset of audio signals from brushless dc motors for predictive maintenance," *Data in Brief*, vol. 50, pp. 109569, 2023.

[17] Rommel Prieto and Diego Bravo, "Machine learning and audio signal processing for predictive maintenance: A review," in *2023 IEEE 6th Colombian Conference on Automatic Control (CCAC)*. IEEE, 2023, pp. 1–6.

[18] Seulki Han, Nasir Mannan, Daryl C Stein, Krishna R Pattipati, and George M Bollas, "Classification and regression models of audio and vibration signals for machine state monitoring in precision machining systems," *Journal of Manufacturing Systems*, vol. 61, pp. 45–53, 2021.

[19] Xiaofei Zhang, Weizhi Liang, Yinpeng Qu, Xiwei Wang, and Sheng Huang, "Motor fault diagnosis based on time–frequency swinging door algorithm and convolutional kernel receptive field matching framework," *IEEE Transactions on Reliability*, 2023.

[20] Yousef Alkhanafseh, Tahir Cetin Akinci, Emine Ayaz, and Alfredo A Martinez-Morales, "Advanced dual rnn architecture for electrical motor fault classification," *IEEE Access*, 2023.

[21] J.M. Montes-Sánchez, J.P. Domínguez-Morales, S. Vicente-Díaz, and A. Jiménez-Fernández, "Neuromorphic audio for predictive maintenance in peristaltic pumps [dataset]," *idUS (Depósito de Investigación de la Universidad de Sevilla)*, 2025.

[22] Juan M. Montes-Sánchez, Juan P. Domínguez-Morales, Saturnino Vicente-Díaz, and Ángel Jiménez-Fernández, "Neuromorphic Audio Dataset for Predictive Maintenance in Peristaltic Pumps," *Proceedings - IEEE International Symposium on Circuits and Systems*, 2025.

[23] Juan M. Montes-Sanchez, Yoko Uwate, Yoshifumi Nishio, Saturnino Vicente-Diaz, and Angel Jimenez-Fernandez,

"Predictive Maintenance Edge Artificial Intelligence Application Study Using Recurrent Neural Networks for Early Aging Detection in Peristaltic Pumps," *IEEE Transactions on Reliability*, vol. 74, no. 3, pp. 3730–3744, 2025.

[24] Emanuele Di Fiore, Antonino Ferraro, Antonio Galli, Vincenzo Moscato, and Giancarlo Sperlì, "An anomalous sound detection methodology for predictive maintenance," *Expert Systems with Applications*, vol. 209, pp. 118324, 2022.

[25] Juan M Montes-Sánchez, Yoko Uwate, Yoshifumi Nishio, Saturnino Vicente-Díaz, and Ángel Jiménez-Fernández, "Peristaltic pump dataset recording method," in *Workshop in R&D+ i & International Workshop on STEM of EPS*. Springer, 2024, pp. 259–264.

[26] Daniel Gutierrez-Galan, Juan Pedro Dominguez-Morales, Angel Jimenez-Fernandez, Alejandro Linares-Barranco, and Gabriel Jimenez-Moreno, "Opennas: open source neuromorphic auditory sensor hdl code generator for fpga implementations," *Neurocomputing*, vol. 436, pp. 35–38, 2021.

[27] Tobi Delbruck et al., "Frame-free dynamic digital vision," in *Proceedings of Intl. Symp. on Secure-Life Electronics, Advanced Electronics for Quality Life and Society*. Citeseer, 2008, vol. 1, pp. 21–26.

[28] Juan P Dominguez-Morales, Daniel Gutierrez-Galan, Antonio Rios-Navarro, Lourdes Duran-Lopez, Manuel Dominguez-Morales, and Angel Jimenez-Fernandez, "Pynavis: An open-source cross-platform software for spike-based neuromorphic audio information processing," *Neurocomputing*, vol. 449, pp. 172–175, 2021.

[29] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[30] Marina Sokolova and Guy Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing and Management*, vol. 45, pp. 427–437, 2009.

[31] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Identity mappings in deep residual networks," 2016.

[33] Mingxing Tan and Quoc V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[34] Danfeng Qin, Chas Leichner, Manolis Delakis, Marco Fornoni, Shixin Luo, Fan Yang, Weijun Wang, Colby Banbury, Chengxi Ye, Berkin Akin, et al., "Mobilenetv4: Universal models for the mobile ecosystem," in *European Conference on Computer Vision*. Springer, 2024, pp. 78–96.

Predicción solar de bajo coste computacional para plataformas IoT empotradas

Melisa Kuzman[1], Soledad Escolar[1], Paula Cervellini[2], Carlos Hernández[1], José Luis Mira[1] y Javier Romero[1]

Resumen— **Muchos dispositivos electrónicos desplegados en exteriores dependen de la energía solar como fuente complementaria para sustentar tareas de detección, procesamiento y comunicación. En el contexto del Internet de las Cosas (IoT), estos dispositivos funcionan también como sistemas de recolección de energía solar (EHS), donde contar con predicciones confiables de producción es crucial para optimizar la planificación de tareas. Sin embargo, el estado del arte carece de modelos de predicción solar de bajo coste computacional que sean fiables sin comprometer la capacidad de cómputo o memoria de los nodos. Este trabajo propone tres modelos de predicción de bajo coste de procesamiento: L-T, L-Mix y L-AR, que integran mecanismos de adaptación local basados en observaciones recientes. Los tres han sido implementados y evaluados en una Raspberry Pi Pico 2, analizando el error cuadrático medio normalizado (NRMSE), el tiempo de ejecución y el uso de memoria. Los resultados muestran que los modelos propuestos superan a los métodos base con una sobrecarga mínima, confirmando su viabilidad para dispositivos IoT con recursos limitados.**

Palabras clave— **Predicción solar, IoT, Modelos de baja complejidad, Sistemas empotrados.**

I. Introducción

La evolución del IoT ha integrado adquisición de datos en tiempo real, procesamiento local y soporte a la decisión predictivo en distintos intervalos de tiempo. Estas capacidades son clave en aplicaciones de eficiencia energética y sostenibilidad como la agricultura inteligente [1], los hogares inteligentes [2] y las redes eléctricas [3], donde las estimaciones precisas de generación energética resultan fundamentales para la planificación y el control.

En el contexto de sistemas IoT basados en recolección de energía, y en particular aquellos que dependen de la energía solar, estas estimaciones resultan especialmente relevantes, ya que permiten definir políticas de planificación de tareas que equilibran el consumo energético de las aplicaciones con la energía recolectada del entorno [4].

Sin embargo, la predicción de la generación solar continúa siendo un problema desafiante. Por un lado, la energía solar presenta una alta variabilidad y depende fuertemente de condiciones meteorológicas cambiantes [5], lo que introduce incertidumbre y limita la precisión de los modelos. Por otro lado, muchos dispositivos IoT operan bajo restricciones de memoria, capacidad de cómputo y disponibilidad energética, lo que dificulta el uso de técnicas de predicción intensivas en datos basadas en grandes volúmenes de información histórica y en procesamiento estadístico complejo. En consecuencia, existe una clara necesidad de modelos de predicción solar que puedan ejecutarse en dispositivos con recursos limitados, manteniendo una precisión adecuada sin requerir el almacenamiento ni el procesamiento de grandes conjuntos de datos.

En este artículo se diseñan e implementan distintos modelos de baja complejidad para la predicción de generación solar, demostrando que mejoran la precisión de los enfoques representativos del estado del arte. El artículo se organiza de la siguiente manera. La sección II revisa los trabajos relacionados. El modelo del sistema se presenta en la sección III. Los modelos de predicción propuestos se describen en la sección IV. El entorno experimental y el análisis los resultados en términos de precisión, tiempo de ejecución y uso de memoria se presentan en la sección V. Finalmente, la sección VI presenta las conclusiones y las líneas de trabajo futuro.

II. Trabajos relacionados

En esta sección, los enfoques existentes para la predicción de energía solar se organizan en dos grupos con el fin de facilitar su análisis. El primer grupo incluye modelos base y de baja complejidad, comúnmente utilizados en sistemas IoT con recolección de energía debido a su simplicidad y bajos requerimientos computacionales. El segundo grupo comprende técnicas de predicción más avanzadas, generalmente basadas en métodos estadísticos o de aprendizaje automático, que buscan mejorar la precisión a costa de mayores demandas computacionales y de memoria.

A. Modelos base

Los modelos base permiten establecer referencias de desempeño y evaluar las mejoras introducidas por estrategias más adaptativas. Se han seleccionado métodos de baja complejidad computacional y mínimos requerimientos de memoria, adecuados para plataformas empotradas con recursos limitados. Los tres primeros, *Previous Half Hour* (PHH), *Previous Day* (PD) y *Average of the Above* (AA), se basan en estrategias de persistencia y promediado simple. El cuarto, el *Exponentially Weighted Moving Average* (EWMA), pondera valores históricos mediante un factor de suavizado.

Estos métodos son ampliamente utilizados en la predicción de energía solar debido a su simplicidad,

[1]Escuela Superior de Informática, Universidad de Castilla-La Mancha, e-mail: melisagisele.kuzman@uclm.es

[2]Instituto de Investigaciones Científicas y Tecnológicas, Universidad Nacional de Mar del Plata

robustez y bajo coste computacional. Los modelos base considerados en este trabajo, junto con sus correspondientes ecuaciones de predicción, se resumen en la Tabla I.

Tabla I: Modelos base y ecuaciones de predicción

Modelo	Ecuación de predicción
PHH	$\hat{p}_d(i) = p_d(i-1)$
PD	$\hat{p}_d(i) = p_{(d-1)}(i)$
AA	$\hat{p}_d(i) = \omega \cdot p_d(i-1) + (1-\omega) \cdot p_{(d-1)}(i)$
EWMA	$\hat{p}_d(i) = \alpha \cdot \hat{p}_{(d-1)}(i) + (1-\alpha) \cdot p_{(d-1)}(i)$

PHH: Se basa en la persistencia a corto plazo, asumiendo que la energía solar generada en el siguiente intervalo de tiempo es igual al valor observado en el intervalo inmediatamente anterior. Este enfoque aprovecha el hecho de que las condiciones de irradiancia suelen mantenerse estables en intervalos cortos. Los predictores basados en persistencia se utilizan ampliamente como referencia en estudios de predicción solar debido a su simplicidad y robustez [6]. Este modelo ofrece una precisión razonable bajo condiciones atmosféricas de variación lenta y actúa como un límite inferior en evaluaciones comparativas.

PD: Aprovecha la periodicidad diaria de la generación solar, asumiendo que el perfil energético del día siguiente será similar al del día anterior. En consecuencia, a cada intervalo del día a predecir se le asigna el valor medido en el mismo intervalo del día previo. Estrategias de persistencia día-a-día similares se utilizan comúnmente como referencia en modelos de predicción solar [7]. Si bien este enfoque captura la estructura diaria típica de la generación solar, su desempeño se degrada ante cambios rápidos en las condiciones meteorológicas, ya que no incorpora información reciente intra-diaria.

AA: Combina la adaptabilidad a corto plazo de PHH con la persistencia diaria de PD. La predicción se calcula como una combinación lineal ponderada entre la medición más reciente y el valor observado en el mismo intervalo del día anterior, controlada por un parámetro de ponderación $\omega \in [0, 1]$. Estrategias de combinación lineal similares son ampliamente utilizadas en esquemas simples de predicción híbrida o en ensambles [8]. Al integrar información de corto plazo y diaria, este modelo proporciona predicciones más suaves que PHH y más adaptativas que PD, por lo que se emplea habitualmente como referencia intermedia en estudios comparativos.

Modelo basado en EWMA: Este predictor, propuesto por Kansal en [5], es un modelo de predicción a corto plazo de baja complejidad diseñado para sistemas IoT con recolección de energía que operan bajo recursos computacionales limitados. Aprovecha el patrón diario regular de la generación solar, actualizando la predicción de cada intervalo a partir de observaciones históricas del mismo intervalo en días anteriores. El parámetro de suavizado $\alpha \in [0, 1]$ controla el compromiso entre estabilidad y capacidad de adaptación: valores altos generan predicciones más suaves dominadas por información pasada, mientras que valores bajos incrementan la respuesta frente a cambios recientes. Este modelo se ha consolidado como una referencia común en sistemas IoT con recolección de energía. Los modelos introducidos en la Sección IV extienden esta formulación mediante mecanismos adicionales de adaptación de baja complejidad.

B. Modelos avanzados

Trabajos recientes han abordado la predicción de energía solar en el contexto de plataformas IoT mediante la combinación de adquisición de datos en tiempo real con procesamiento en la nube y técnicas de aprendizaje automático. Por ejemplo, [9] propone una plataforma IoT que integra datos de producción fotovoltaica e información meteorológica con modelos de aprendizaje en ensamble para realizar predicción a corto plazo y detección automática de anomalías. De forma similar, otros estudios exploran arquitecturas IoT completas para monitoreo y predicción de energía solar, basadas en algoritmos supervisados entrenados con grandes volúmenes de datos históricos y múltiples variables exógenas [10, 11]. Estos enfoques suelen alcanzar alta precisión y permiten funcionalidades avanzadas como paneles de visualización, notificaciones y detección de fallos.

Estos modelos suelen implementarse en arquitecturas IoT donde la etapa de predicción se ejecuta en servidores externos o en la nube, asumiendo conectividad continua, entrenamiento fuera de línea y recursos computacionales superiores a los de los dispositivos con recolección de energía. Este trabajo, en contraste, se centra en algoritmos que operan localmente sobre plataformas empotradas con recursos limitados, sin depender de procesamiento remoto ni de *pipelines* complejos de aprendizaje. El foco se sitúa en modelos de baja complejidad adecuados para dispositivos IoT con recolección de energía, como los evaluados en [12], logrando al mismo tiempo mejoras significativas de precisión respecto a los enfoques base.

III. Modelo del sistema

En esta sección se describe el modelo del sistema y la notación adoptada a lo largo del trabajo para la predicción de energía solar en dispositivos IoT con recolección de energía.

Se consideran dispositivos IoT con EHS solar que operan bajo restricciones estrictas de memoria, capacidad de cómputo y disponibilidad energética. Estos dispositivos ejecutan estrategias de planificación de tareas [13] con el objetivo de alcanzar neutralidad energética [5]. Dichas estrategias se apoyan en pronósticos de generación solar que, combinados con modelos de consumo energético de las aplicaciones, permiten anticipar planificaciones óptimas en un horizonte temporal diario, garantizando la neutrali-

dad energética y optimizando métricas adicionales de desempeño, como la maximización de la utilidad de la aplicación [14, 15] o la minimización de las variaciones de calidad entre tareas consecutivas [16]. En consecuencia, las mejoras en la precisión de la predicción impactan directamente en la calidad y estabilidad de las decisiones de planificación.

A partir de este contexto, se define la notación utilizada en el resto del trabajo. Se considera un horizonte diario, discretizando cada día d en $k = 48$ intervalos de tiempo indexados por i. La potencia solar medida (o magnitud eléctrica equivalente) en el intervalo i del día d se denota como $p_d(i)$, mientras que $\hat{p}_d(i)$ representa el valor predicho correspondiente. En general, un algoritmo de predicción de energía solar puede expresarse como:

$$\hat{p}_d(i) = f_m(\cdot) \pm e \quad \forall i \in [1, k] \tag{1}$$

donde $f_m(\cdot)$ denota un algoritmo de predicción, $(\cdot)$ representa un conjunto de variables de entrada derivadas de mediciones históricas y/o predicciones previamente calculadas asociadas al día actual o a días anteriores, y e representa el error de predicción. La predicción de referencia basada en EWMA se denota como $\hat{p}_d^{(0)}(i)$ y se utiliza como línea base para los modelos mejorados introducidos en la siguiente sección.

IV. Modelos de predicción propuestos

En esta sección se presentan tres modelos de predicción de energía solar de baja complejidad desarrollados en este trabajo: i) L–T (corrección basada en temperatura), ii) L–Mix (híbrido), y iii) L–AR (autorregresivo). Los modelos propuestos se inspiran en el enfoque de predicción basado en EWMA revisado en la Sección II, y están diseñados para mejorar la precisión bajo condiciones atmosféricas variables, manteniendo la baja complejidad computacional requerida por dispositivos IoT con recolección de energía. Cada modelo incorpora información temporal adicional para aumentar la capacidad de adaptación sin introducir un coste significativo en procesamiento o memoria.

A. Modelo L–T

El primer modelo propuesto, denominado L–T, extiende un predictor base basado en EWMA mediante la incorporación de una corrección en función de la temperatura. Esta extensión se motiva en el impacto negativo que las altas temperaturas ejercen sobre la eficiencia eléctrica de los paneles fotovoltaicos, reduciendo su producción respecto a las condiciones nominales. El modelo ajusta la predicción base de acuerdo con la tendencia térmica esperada entre días consecutivos. La predicción resultante se define como:

$$\hat{p}_d(i) = \hat{p}_d^{(0)}(i) \cdot f_T(d) \tag{2}$$

donde $f_T(d)$ es un factor de corrección definido como:

$$f_T(d) = 1 - \lambda \cdot \Delta T(d).$$

$\Delta T(d)$ representa la variación de la temperatura máxima diaria prevista entre dos días consecutivos, es decir, $\Delta T(d) = T_{\text{máx}}(d) - T_{\text{máx}}(d-1)$, y λ es un coeficiente de escala que controla la sensibilidad de la corrección.

Esta formulación permite capturar efectos térmicos entre días utilizando información de temperatura de baja granularidad, manteniendo la naturaleza de baja complejidad del modelo. La Figura 1 ilustra el mecanismo de corrección.

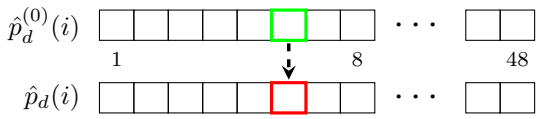


Fig. 1: Esquema de predicción del modelo L–T

B. Modelo L–Mix

El modelo L–Mix extiende el predictor base EWMA combinando la predicción día a día con información de persistencia a corto plazo. Esta estrategia híbrida busca mejorar la respuesta ante variaciones rápidas de la irradiancia incorporando la medición más reciente disponible durante el día actual. La predicción en el intervalo de tiempo i del día d se calcula como:

$$\hat{p}_d(i) = \gamma \cdot p_d(i-1) + (1-\gamma) \cdot \hat{p}_d^{(0)}(i) \tag{3}$$

donde $p_d(i-1)$ denota la potencia solar medida en el intervalo anterior y $\gamma \in [0, 1]$ controla el compromiso entre adaptación a corto plazo y suavizado basado en información histórica. Este diseño mejora la capacidad de respuesta a corto plazo sin comprometer la simplicidad estructural del modelo. La Figura 2 ilustra el mecanismo de predicción del modelo L–Mix.

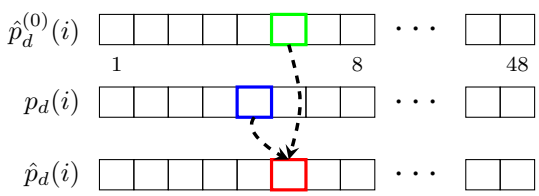


Fig. 2: Esquema de predicción del modelo L–Mix

C. Modelo L–AR

El modelo L–AR introduce una corrección autorregresiva basada en el error de predicción observado en el intervalo de tiempo anterior. A diferencia de enfoques tradicionales basados en promedios históricos, este modelo compensa explícitamente desajustes recientes entre los valores predichos y medidos de generación solar. La predicción se calcula como:

$$\hat{p}_d(i) = \hat{p}_d^{(0)}(i) + \beta \cdot e_d(i-1) \tag{4}$$

donde $e_d(i-1)$ representa el error de predicción en el intervalo anterior (esto es, $p_d(i-1) - \hat{p}_d(i-1)$) y β es un parámetro de ganancia que controla la influencia de la realimentación del error. Al incorporar este

término, el modelo ajusta dinámicamente la predicción base, permitiendo una adaptación más rápida frente a cambios abruptos en la irradiancia. La Figura 3 ilustra este esquema de predicción.

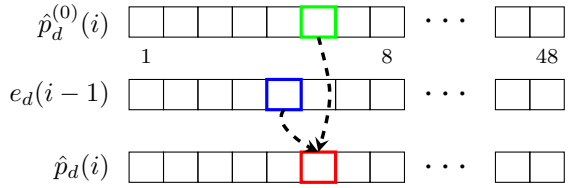


Fig. 3: Esquema de predicción del modelo L–AR

V. Configuración experimental y resultados

En esta sección se presenta la metodología experimental y los resultados obtenidos en la evaluación de los modelos de predicción propuestos. El objetivo es analizar su precisión utilizando mediciones solares reales, así como evaluar su viabilidad computacional en plataformas empotradas con recursos limitados.

A. Metodología

El proceso de evaluación se compone de cuatro etapas: i) preprocesamiento del conjunto de datos; ii) definición de métricas de evaluación; iii) ajuste de parámetros de los modelos; y iv) evaluación del rendimiento. Las siguientes subsecciones describen cada etapa en detalle.

A.1 Preprocesamiento del conjunto de datos

El estudio experimental se basa en un conjunto de datos público introducido en [12], obtenido a partir de un banco de pruebas IoT con recolección de energía. El conjunto de datos incluye mediciones reales tales como la corriente de cortocircuito fotovoltaica (utilizada como indicador de la disponibilidad instantánea de energía solar), consumo energético, corriente y tensión de la batería, y variables ambientales (temperatura, humedad y velocidad del viento).

Los datos abarcan 64 días consecutivos bajo diversas condiciones atmosféricas, con un intervalo de muestreo de un minuto, lo que da lugar a más de 93.000 registros, disponibles en https://github.com/UCLM-ARCO/energy-harvesting-dataset. Para fines de predicción, las mediciones se agrupan en intervalos fijos de 30 minutos, obteniéndose $k = 48$ intervalos por día, en coherencia con la discretización temporal definida en la Sección III.

El modelo L-T incorpora, adicionalmente, predicciones de temperatura máxima diaria proporcionadas por la Agencia Estatal de Meteorología [17]. Los pronósticos de temperatura máxima se pueden obtener mediante acceso a la API del sitio.

A.2 Métricas de evaluación

La precisión de la predicción se evalúa comparando los valores predichos con las mediciones correspondientes mediante métricas de error estándar comúnmente utilizadas en la predicción de energía solar. En particular, se emplea el Error Cuadrático Medio Normalizado (NRMSE) como indicador principal, ya que proporciona una medida independiente de la escala que permite la comparación directa entre distintos días y condiciones de operación. De forma complementaria, también se reportan el Error Absoluto Medio (MAE) y el Error Cuadrático Medio (RMSE) en el análisis global de rendimiento. Además de la precisión, el desempeño de los algoritmos propuestos se evalúa en términos de tiempo de ejecución y uso de memoria en una RP Pico 2.

A.3 Ajuste de parámetros

Los tres modelos propuestos se ajustan mediante un procedimiento de optimización común que minimiza el NRMSE global sobre el conjunto completo de datos. Esta estrategia garantiza que cada modelo opere bajo una configuración representativa derivada del mismo criterio objetivo. Los modelos sin parámetros libres se evalúan directamente utilizando los datos medidos.

El Algoritmo 1 resume el procedimiento de ajuste de parámetros como una búsqueda exhaustiva sobre un conjunto finito de configuraciones candidatas. En este contexto, $\Theta[1..Z]$ denota el conjunto de posibles configuraciones evaluadas para cada modelo. La estructura bidimensional $p_d(i)$ se reorganiza en un único vector p_c de longitud $N \times k$, con $N = 64$ y $k = 48$, que contiene todas las mediciones a lo largo de los días e intervalos de tiempo, con el fin de calcular el NRMSE global sobre el conjunto completo. El factor de normalización $p_{\text{máx}}$ corresponde al valor máximo observado en p_c y se utiliza para garantizar la comparación independiente de la escala entre modelos. La Tabla II resume los valores óptimos de los parámetros de suavizado para cada algoritmo evaluado, correspondientes a aquellos que minimizan el NRMSE.

Algoritmo 1 Selección de parámetros

Entrada: $p_d[1..k] \,\forall\, d \in [1, N]\,, f_m(\cdot, \theta),\ \Theta[1..Z]$
Salida: $\theta^\star$
Construir p_c tal que $p_c[(d-1)\cdot k + i] \leftarrow p_d(i),\ \forall d \in [1, N],\ \forall i \in [1, k]$
$p_{\text{máx}} \leftarrow \text{máx}(p_c)$
$\text{NRMSE}_{\text{mín}} \leftarrow +\infty$
$\theta^\star \leftarrow \Theta[1]$
for $z = 1$ to Z **do**
  $\theta \leftarrow \Theta[z]$, $S \leftarrow 0$
  for $j = 1$ to $N \times k$ **do**
    $\hat{p}_c[j] \leftarrow f_m(j, p_c[\cdot], \theta)$
    $S \leftarrow S + (p_c[j] - \hat{p}_c[j])^2$
  end for
  $\text{NRMSE} \leftarrow \sqrt{\text{S}/(\text{N} \times \text{k})}/\text{p}_{\text{máx}} \times 100$
  if $\text{NRMSE} < \text{NRMSE}_{\text{mín}}$ **then**
    $\text{NRMSE}_{\text{mín}} \leftarrow \text{NRMSE}$
    $\theta^\star \leftarrow \theta$
  end if
end for
return $\theta^\star$

A.4 Evaluación del rendimiento

La evaluación se realiza aplicando cada modelo de predicción sobre el conjunto completo de datos utili-

Tabla II: Valores de los parámetros de los modelos de predicción evaluados

Modelo	Parámetros	Valores
AA	w	0.448
EWMA	α	0.863
L–T	α, λ	0.863, 0.013
L–Mix	α, γ	0.943, 0.351
L–AR	α, β	0.828, 0.673

zando las mismas entradas preprocesadas. Para cada modelo, se generan predicciones para todos los intervalos de tiempo y se comparan con las mediciones correspondientes. Las métricas de error resultantes se calculan sobre el conjunto completo de muestras y se utilizan para caracterizar tanto el comportamiento en el peor caso como el rendimiento promedio. Este procedimiento garantiza la consistencia entre modelos y refleja el desempeño esperable en un escenario real de implementación sobre dispositivos IoT con recolección de energía.

B. Resultados

Esta subsección presenta los resultados obtenidos al aplicar los modelos de predicción descritos en la Sección IV sobre el conjunto de datos experimental introducido previamente. El análisis comienza reportando la precisión global de los modelos sobre el conjunto completo, así como su comportamiento bajo condiciones atmosféricas exigentes. Posteriormente, los modelos se comparan en términos de tiempo de ejecución, uso de memoria y coste computacional.

B.1 Análisis sobre todo el conjunto de datos

Con los parámetros obtenidos previamente, la Tabla III muestra que L-AR alcanza el menor error global en todas las métricas, con un NRMSE de 4.65 %. Esto representa una reducción aproximada del 27 % respecto al modelo EWMA original (6.34 %) y cercana al 50 % en comparación con el predictor de persistencia más simple PHH (9.23 %). Los modelos L-Mix y L-T también mejoran la formulación base EWMA, confirmando que mecanismos adaptativos de baja complejidad pueden aportar mejoras consistentes en precisión sin incrementar significativamente la complejidad algorítmica.

Tabla III: Error de predicción sobre el conjunto completo

Modelo	MAE [mA]	RMSE [mA]	NRMSE [%]
PHH	16.17	33.23	9.23
PD	11.15	30.30	8.42
AA	10.81	23.86	6.63
EWMA	9.39	22.82	6.34
L–T	9.31	22.56	6.27
L–Mix	8.81	19.27	5.35
L–AR	**6.67**	**16.76**	**4.65**

Para facilitar una interpretación cualitativa de estos resultados globales, la Figura 4 muestra la corriente de cortocircuito fotovoltaica medida junto con las predicciones correspondientes durante cinco días consecutivos. Esta comparación permite analizar la capacidad de cada modelo para seguir la evolución temporal de la señal bajo condiciones reales de operación. Como se observa, los modelos basados en persistencia (PHH, PD y AA) generan transiciones temporales suaves, pero presentan un retardo significativo frente a variaciones rápidas tanto entre días como dentro de un mismo día. Este retardo se traduce en una mayor acumulación de error en periodos prolongados. Por su parte, el modelo EWMA mejora la representación del perfil solar diario al explotar información histórica a nivel de intervalo, aunque su fuerte efecto de suavizado limita su capacidad de reacción ante cambios abruptos de irradiancia.

Los modelos adaptativos propuestos mitigan esta limitación. En particular, L–AR muestra la mayor concordancia con la señal medida a lo largo de días consecutivos, especialmente durante el incremento matutino, el pico de irradiancia y la caída vespertina. La incorporación de un término de realimentación del error de baja complejidad permite corregir desviaciones recientes de forma más rápida, manteniendo la estabilidad de la estructura EWMA subyacente, lo que se traduce directamente en la reducción observada del NRMSE global.

B.2 Análisis de escenario de peor caso

Como complemento a la evaluación global, se analiza un escenario de peor caso seleccionando el día en el que la mayoría de los modelos alcanza su mayor error de predicción. Esta situación ocurre el 8 de septiembre y se caracteriza por una alta variabilidad intra-diaria, incluyendo fluctuaciones abruptas de irradiancia que incrementan significativamente la dificultad de la predicción a corto plazo.

Los valores de NRMSE correspondientes se presentan en la Tabla IV. En este escenario altamente dinámico, el modelo PHH basado en persistencia obtiene el menor error (12.78 %), beneficiándose de su capacidad de reacción inmediata ante variaciones rápidas de la señal. Sin embargo, este comportamiento contrasta con su bajo rendimiento global sobre el conjunto completo (Tabla III), evidenciando el compromiso conocido entre capacidad de respuesta a corto plazo y estabilidad a largo plazo.

Entre los restantes modelos, L–AR presenta el mejor desempeño (14.74 %), superando ampliamente al modelo EWMA original (27.58 %) y al resto de variantes. Este resultado confirma que la incorporación de un mecanismo ligero de realimentación del error mejora la robustez frente a condiciones de irradiancia altamente variables, evitando el retardo excesivo introducido por estrategias de suavizado más agresivas.

La Figura 5 muestra la comparación entre la corriente de cortocircuito fotovoltaica medida y las predicciones generadas por todos los modelos en este día

Tabla IV: NRMSE obtenido en el día de peor caso

Modelo	PHH	PD	AA	EWMA	L–T	L–Mix	L–AR
NRMSE (%)	12.78	27.24	17.57	27.58	26.63	20.04	**14.74**

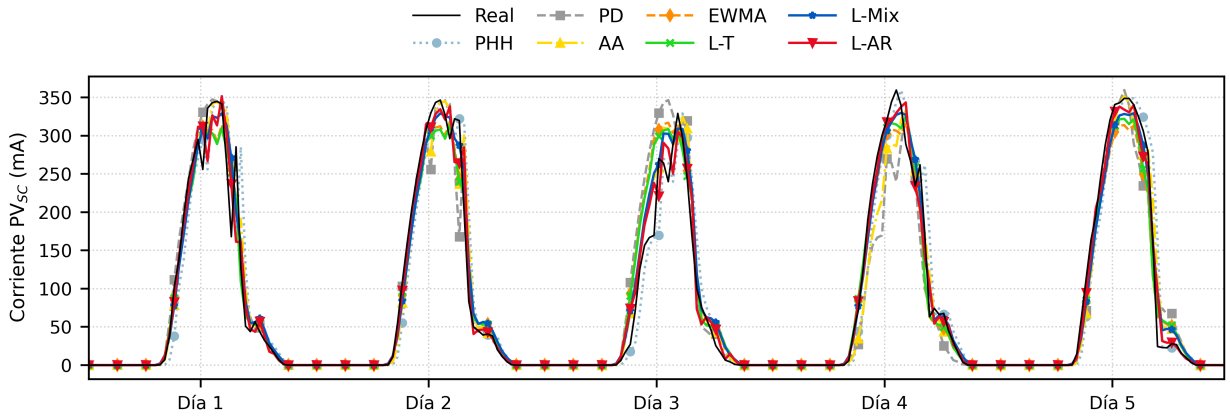


Fig. 4: Desempeño de predicción durante cinco días consecutivos

de peor caso. La figura evidencia cómo los modelos basados en persistencia siguen de cerca las transiciones abruptas, mientras que los modelos puramente basados en EWMA presentan respuestas retardadas. El modelo L–AR compensa parcialmente este efecto ajustando la predicción en función del error más reciente, logrando un mejor equilibrio entre capacidad de respuesta y estabilidad.

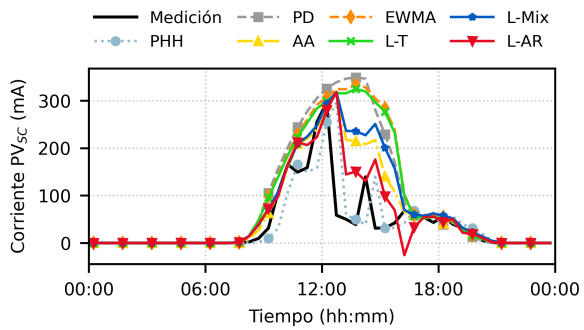


Fig. 5: Predicciones en el día con mayor variabilidad

B.3 Análisis del rendimiento computacional

Además de la precisión, la aplicabilidad práctica de los modelos propuestos depende de su viabilidad en plataformas empotradas con recursos limitados. Por este motivo, todos los modelos se implementan y ejecutan en un microcontrolador Raspberry Pi Pico 2 [18], basado en un núcleo ARM Cortex-M33 y con 520,KB de memoria SRAM, utilizando MicroPython. El correcto funcionamiento se verifica comparando los valores de NRMSE obtenidos en el microcontrolador con los calculados en una implementación en PC, cuya estrecha concordancia confirma la validez de las implementaciones empotradas. Para garantizar una comparación justa, se desarrollan scripts equivalentes para todos los modelos, aislando la etapa de predicción y excluyendo la adquisición y el preprocesamiento de datos. El tiempo de ejecución se mide utilizando datos reales y repitiendo las mediciones para reducir el *jitter* temporal.

La Figura 6 muestra el tiempo de ejecución por predicción en función de la frecuencia del microcontrolador. Todos los modelos presentan una relación aproximadamente inversa entre latencia y frecuencia de reloj, consistente con una carga dominada por operaciones aritméticas y accesos a memoria local.

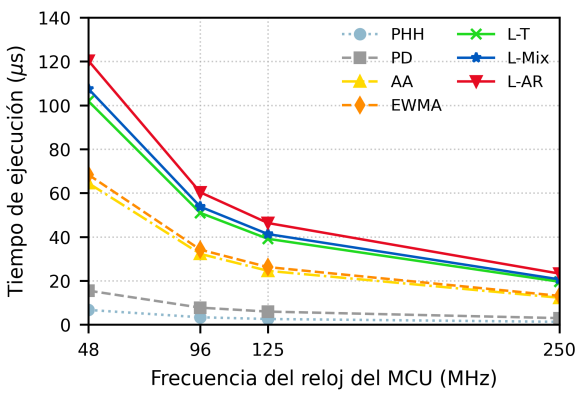


Fig. 6: Tiempo de ejecución por predicción en función de la frecuencia del microcontrolador para los modelos evaluados en Raspberry Pi Pico 2

A frecuencias bajas, típicamente empleadas en sistemas IoT con recolección de energía para reducir el consumo, las diferencias relativas entre modelos se acentúan. A 48 MHz, PHH requiere aproximadamente 6.7 μs por predicción, el modelo basado en EWMA 68.4 μs y L–AR 120.3 μs. Aunque L–AR introduce un mayor coste computacional debido a la realimentación del error, su tiempo de ejecución se mantiene en el orden de cientos de microsegundos incluso en este régimen conservador.

A la frecuencia nominal de 250 MHz, los tiempos de ejecución se reducen a 1.29 μs para PHH, 13.13 μs para EWMA y 23.28 μs para L–AR. Considerando que las predicciones se generan cada 30 minutos, esta latencia resulta despreciable para la aplicación objetivo y no introduce retardos perceptibles a nivel de sistema.

Los requerimientos de memoria persistente, resumidos en la Tabla V, muestran una tendencia similar. PHH requiere solo 48 B, mientras que los modelos EWMA y L–AR requieren 448 B y 464 B, respectivamente. El incremento introducido por las variantes adaptativas es, por tanto, mínimo y se mantiene muy por debajo de 1 KB en todos los casos.

Estos resultados demuestran que las mejoras en

Tabla V: Requerimientos de Memoria persistente en Raspberry Pi Pico 2

Modelo	PHH	PD	AA	EWMA	L–T	L–Mix	L–AR
Memoria (B)	48	208	256	448	448	464	464

precisión logradas por L–AR se obtienen sin un incremento significativo en el coste computacional ni en el uso de memoria, manteniendo plena compatibilidad con microcontroladores de bajo consumo.

VI. Conclusiones

En este trabajo se proponen tres modelos de predicción de energía solar de baja complejidad, L-T, L-Mix y L-AR, diseñados para dispositivos IoT con recolección de energía y recursos limitados, evaluados frente a enfoques representativos del estado del arte sobre un conjunto de datos real de 64 días. Los resultados muestran que la incorporación de mecanismos de adaptación simples sobre un predictor base EWMA reduce de forma significativa el error global sin comprometer la simplicidad estructural. L-AR, mediante una corrección basada en la realimentación del error, alcanza la mejor precisión global. L-Mix y L-T también mejoran el modelo base incorporando, respectivamente, información de persistencia a corto plazo y predicciones de temperatura, ofreciendo compromisos intermedios entre estabilidad y adaptación. Los experimentos sobre una Raspberry Pi Pico 2 con MicroPython confirman la viabilidad de todos los modelos en microcontroladores de bajo consumo, con un sobrecoste moderado en tiempo de ejecución y requerimientos de memoria reducidos, lo que demuestra que las mejoras en precisión se logran sin exigir recursos significativos.

Como trabajo futuro, los modelos se integrarán en esquemas de planificación energéticamente neutral y se evaluarán en experimentos de larga duración sobre plataformas IoT empotradas, analizando el impacto de la precisión de la predicción en la gestión de tareas y la estabilidad del almacenamiento energético bajo condiciones solares variables.

Agradecimientos

Esta investigación cuenta con el apoyo parcial del Gobierno Regional de Castilla-La Mancha (INNOCAM) y FEDER a través de los proyectos RURALCARE (SBPLY/24/180225/000158) y REACH (SBPLY/24/180225/000105), del Ministerio de Economía, Comercio y Empresa de España mediante el proyecto OASIS (PID2023-148285OB-C41), del programa PERTE-Chip (TSI-069100-2023-0014), y del programa Horizon Europe a través del proyecto MAYA (GA Nº 101213323).

Referencias

[1] Hany Abdelsalam, Alireza Souri, and Nihat İnanç, "A time series forecasting approach based on gradient boosting method for iot-based solar energy production systems," *Energy Storage and Saving*, vol. 4, no. 4, pp. 413–420, 2025.

[2] Giuseppe Del Fiore, Teodoro Montanaro, Ilaria Sergi, and Luigi Patrono, "Adaptive iot architecture with incremental learning for on-line solar production forecasting," *Internet of Things*, vol. 33, pp. 101724, 2025.

[3] Pierre Bouquet, Ilya Jackson, Mostafa Nick, and Amin Kaboli, "Ai-based forecasting for optimised solar energy management and smart grid efficiency," *International Journal of Production Research*, vol. 62, no. 13, pp. 4623–4644, 2024.

[4] Antonio Caruso, Stefano Chessa, Soledad Escolar, Jesús Barba, and Juan Carlos López, "Collection of data with drones in precision agriculture: Analytical model and lora case study," *IEEE Internet of Things Journal*, vol. 8, no. 22, pp. 16603–16615, 2021.

[5] Aman Kansal, Jason Hsu, Sadaf Zahedi, and Mani Srivastava, "Srivastava mb power management in energy harvesting sensor networks," *ACM Trans. Embedded Comput. Syst.*, vol. 6, 09 2007.

[6] Rich H. Inman, Hugo T.C. Pedro, and Carlos F.M. Coimbra, "Solar forecasting methods for renewable energy integration," *Progress in Energy and Combustion Science*, vol. 39, no. 6, pp. 535–576, 2013.

[7] Peder Bacher, Henrik Madsen, and Henrik Aalborg Nielsen, "Online short-term solar power forecasting," *Solar Energy*, vol. 83, no. 10, pp. 1772–1783, 2009.

[8] Dazhi Yang, Wenting Wang, Christian A. Gueymard, Tao Hong, Jan Kleissl, Jing Huang, Marc J. Perez, Richard Perez, Jamie M. Bright, Xiang'ao Xia, Dennis van der Meer, and Ian Marius Peters, "A review of solar forecasting, its dependence on atmospheric sciences and implications for grid integration: Towards carbon neutrality," *Renewable and Sustainable Energy Reviews*, vol. 161, pp. 112348, 2022.

[9] Giuseppe Del Fiore, Maria Luisa Belcuore, Giacomo Erroi, Ilaria Sergi, Teodoro Montanaro, and Luigi Patrono, "A real-time iot-based system for solar energy forecasting and automated anomaly notifications," in *2025 10th International Conference on Smart and Sustainable Technologies (SpliTech)*, 2025, pp. 1–6.

[10] Preeti Mittal and Vivek Kumar Singh, "Smart solar energy forecasting and monitoring using iot: Current issues and recent trends," in *2025 Seventeenth International Conference on Contemporary Computing (IC3)*, 2025, pp. 1–5.

[11] Harish Reddy Gantla, Mayank Namdev, Udit Mamodiya, Indra Kishor, Gireesh Kumar, and Parul Goyal, "A smart monitoring framework for sustainable solar energy harvesting using iot and machine learning techniques," in *2025 International Conference on Sustainability, Innovation & Technology (ICSIT)*, 2025, pp. 1–6.

[12] Melisa Kuzman, Xavier del Toro García, Soledad Escolar, Antonio Caruso, Stefano Chessa, and Juan Carlos López, "A testbed and an experimental public dataset for energy-harvested iot solutions," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, 2019, vol. 1, pp. 869–876.

[13] Soledad Escolar, Antonio Caruso, Stefano Chessa, Xavier del Toro, Félix J. Villanueva, and Juan C. López, "Statistical energy neutrality in iot hybrid energy-harvesting networks," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, 2018, pp. 00444–00449.

[14] Antonio Caruso, Stefano Chessa, Soledad Escolar, Xavier del Toro, and Juan Carlos López, "A dynamic programming algorithm for high-level task scheduling in energy harvesting iot," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2234–2248, 2018.

[15] Junlong Zhou, Jin Sun, Peijin Cong, Zhe Liu, Xiumin Zhou, Tongquan Wei, and Shiyan Hu, "Security-critical energy-aware task scheduling for heterogeneous real-time mpsocs in iot," *IEEE Transactions on Services Computing*, vol. 13, no. 4, pp. 745–758, 2020.

[16] Antonio Caruso, Stefano Chessa, Soledad Escolar, Fernando Rincón, and Juan C. López, "Task scheduling stabilization for solar energy harvesting internet of things devices," in *2022 IEEE Symposium on Computers and Communications (ISCC)*, 2022, pp. 1–6.

[17] Agencia Estatal de Meteorología, "AEMET - página principal," Accessed: February 10, 2026.

[18] Raspberry Pi Foundation, "Raspberry pi pico 2 datasheet: An rp2350-based microcontroller board," Accessed: February 10, 2026.

Controlador neuromórfico de robot articulado en FPGA con interfaz para aceleradores de redes neuronales pulsantes

Alejandro Linares-Barranco, Daniel Casanueva-Morato, Diógenes Trejos-Hernández, Daniel Cascado-Caballero, Juan P. Domínguez-Morales, Ángel F. Jiménez-Fernández[1],

Resumen— **Este artículo presenta el desarrollo y la implementación de un controlador robótico neuromórfico en una FPGA, diseñado para admitir trayectorias fluidas basadas en redes neuronales de impulsos (SNN). Utilizando el marco ED-Scorbot, el sistema controla un brazo robótico Scorbot ER-VII con seis grados de libertad a través de una arquitectura proporcional-integral-derivativa (SPID) basada en eventos. A diferencia del control robótico tradicional, este enfoque emplea la modulación por frecuencia de pulso (PFM) para accionar motores de corriente continua, imitando la activación de las neuronas motoras biológicas. El controlador se sintetiza dentro de la lógica programable (PL) de un Xilinx Zynq 7100, mientras que el sistema de procesamiento (PS) gestiona la configuración a través de protocolos MQTT y conectividad remota. Nuestros resultados demuestran que la arquitectura de procesamiento basada en impulsos es altamente eficiente, ya que consume solo el 4,8 % de las LUT y el 2,0 % de los registros. Con un consumo energético total estimado del chip de 2,04 W, el sistema demuestra ser una solución escalable y de bajo consumo para el control motor neuroinspirado en tiempo real en robots con múltiples grados de libertad.**

Palabras clave— **FPGA, MPSoC, control PID pulsante, motores DC, Ingeniería Neuromórfica**

I. Introducción

La investigación en robótica neuromórfica se lleva a cabo principalmente en la intersección entre la informática, la ingeniería y la neurociencia. A esta área se la conoce también con varios términos relacionados, entre los que se incluyen: robots bioinspirados, robots basados en eventos, dispositivos inspirados en el cerebro, robots cognitivos, ingeniería neuromórfica y neuro-robótica, entre otros. Un origen comúnmente citado de este campo son los robots tortuga de William Grey Walter de la década de 1950 [1][2], cuyas acciones sorprendentemente complejas surgían de un sencillo "sistema nervioso" electrónico analógico. Otro hito clave fue el de los Vehículos de Braitenberg[3], que motivaron a muchos investigadores a adoptar lo que él denominó una metodología sintética para investigar el sistema integrado de cerebro, cuerpo y comportamiento. La metodología sintética puede interpretarse como "comprender mediante la construcción", un principio rector especialmente adecuado para la robótica neuromórfica y basada en el cerebro.

El término robot se introdujo por primera vez en 1921 en la obra de Karel Čapek R.U.R. (Rossum's Universal Robots)[4], durante la Revolución Industrial, para describir máquinas destinadas a asumir tareas humanas. El campo de la robótica avanzó sustancialmente a lo largo del siglo XX, especialmente en la industria, aunque la imitación detallada de las capacidades humanas recibió comparativamente poca atención. En el presente siglo, las expectativas respecto al progreso tecnológico son aún mayores. Los robots se utilizan actualmente en ámbitos comerciales, industriales, militares, médicos y de entretenimiento (por ejemplo, aspiradoras robóticas, futbolistas robóticos, vehículos autónomos terrestres y aéreos, y sistemas quirúrgicos robóticos), lo que mejora significativamente la productividad.

Sin embargo, a pesar de las mejoras en los potentes procesadores integrados, los dispositivos de IA periférica y las técnicas de bajo consumo, el comportamiento, la inteligencia y la eficiencia energética de los robots siguen muy por debajo de los de los seres humanos, los animales o incluso los insectos con sistemas nerviosos. Los organismos biológicos prosperan en entornos dinámicos, mostrando una flexibilidad, una adaptabilidad y unas estrategias de supervivencia robustas que los sistemas artificiales no pueden igualar. La robótica neuromórfica tiene como objetivo investigar los fundamentos de la inteligencia biológica mediante la integración de modelos de sistemas neuronales en plataformas robóticas. Los robots neuromórficos son dispositivos físicos cuyo sistema de control se ha modelado imitando algún aspecto clave del sistema neuronal biológico, como su arquitectura antropomórfica, su control motor inspirado en el sistema nervioso, su conjunto de sensores neuromórficos, etc.

La robótica neuromórfica se basa en la cognición incorporada, postulando que el comportamiento inteligente surge de la interacción dinámica entre el sistema nervioso central, el cuerpo físico y el entorno [5]. Mientras que la neurociencia tradicional emplea técnicas analíticas para estudiar la función cerebral, la ingeniería neuromórfica utiliza una metodología sintética. Mediante la construcción de robots inspirados en la neurobiología, los investigadores establecen heurísticas empíricas para comprender cómo las redes neuronales procesan las sensaciones para controlar el movimiento y generar un comportamiento complejo y adaptativo.

Los movimientos de alcance constituyen un foco principal dentro de este ámbito [6]. En los sistemas biológicos, el sistema nervioso central (SNC) coordi-

[1]Dpto. de Arquitectura y Tecnología de Computadores, EPS-ETSII, Universidad de Sevilla, e-mail: alinares@atc.us.es

na estas acciones a través de la comunicación neuronal basada en pulsos. La corteza cerebral procesa la información sensoriomotora a través de una arquitectura organizada jerárquicamente (que comprende áreas primarias, secundarias y terciarias) que traduce los estímulos externos en una ejecución motora precisa. Este bucle sensoriomotor continuo opera a través de la médula espinal, el tronco encefálico y la corteza motora mediante una red masivamente interconectada e impulsada por retroalimentación [7]. La bibliografía actual demuestra la eficacia de los controladores motores inspirados en el sistema nervioso para conciliar la adaptabilidad biológica y la eficiencia computacional [8][9]. Las implementaciones van desde emulaciones VLSI de circuitos espinales de bucle cerrado para el control de músculos antagonistas [10], hasta robots móviles que aprovechan modelos de un cuarto de millón de neuronas para el aprendizaje sensoriomotor de bajo consumo [11]. Otros modelos exitosos incluyen la locomoción inspirada en los insectos, impulsada por generadores de patrones centrales basados en redes neuronales de impulsos (SNN) en el hardware SpiNNaker [12], y la navegación por laberintos ejecutada mediante circuitos neuromórficos orgánicos [13].

Como tercera generación de inteligencia artificial, las arquitecturas neuromórficas proporcionan una capa cognitiva fundamental para la robótica moderna [14]. No obstante, la optimización de los controladores de motores de impulsos para reducir aún más la latencia y el consumo de energía sigue siendo un reto de investigación pendiente. Durante la última década, se han implementado con éxito varios controladores inspirados en la biología, como SECLOC [15], SPID [16], VITE-FLETE [17] y SVITE [18].

Este artículo se centra en la implementación en FPGA del SPID para el control de un brazo robótico basado en eventos. Inicialmente, la infraestructura ED-Biorob [19] demostró la aplicación del SPID para controlar múltiples articulaciones robóticas mediante un enfoque neuromórfico, utilizando dos FPGA Spartan junto con un ordenador host para la configuración del controlador y la generación de comandos. Posteriormente, la arquitectura SPID se adaptó al marco ED-Scorbot [20] utilizando un MPSoC Zynq [21]. En este sistema integrado, los controladores de motor necesarios se sintetizaron dentro de la lógica programable (PL), mientras que los protocolos de configuración y conectividad remota se ejecutaron en el sistema de procesamiento (PS). Desde entonces, la infraestructura ED-Scorbot se ha ampliado para admitir la interfaz con aceleradores de hardware de SNNs, incluidos SpiNNaker [22] y Dynap-SE [23]. Las secciones siguientes detallan la arquitectura lógica de la FPGA y presentan un análisis exhaustivo de la latencia del sistema y el consumo de energía.

II. Arquitectura FPGA

El ED-Scorbot es un brazo robótico neuromórfico compuesto por una placa basada en FPGA para el control de motores por pulsos, una etapa de potencia eléctrica compatible con la modulación de frecuencia de pulso (PFM) para accionar motores de corriente continua, un Scorbot ER-VII (Fig. 1) con seis grados de libertad (DOF), seis motores de 12 V CC que, mediante transmisiones, producen movimiento en los ejes de sus cinco articulaciones de tipo rotatorio y abren y cierran una pinza (sexta articulación) [24]. Cada motor es accionado por un controlador PID (SPID) basado en eventos [16], que utiliza PFM para suministrar potencia a los motores. A diferencia de la modulación por ancho de pulso (PWM), que se utiliza tradicionalmente en robótica, la PFM reproduce el funcionamiento de los sistemas biológicos, en los que los músculos se activan mediante ráfagas de impulsos generadas por las neuronas motoras [21].

La implementación anterior se llevó a cabo en el ED-BioRob [19], donde los SPID se replicaron y distribuyeron entre dos placas Spartan para controlar los brazos robóticos. La Spartan-6 1500 de la placa AER-Node [25] tenía capacidad suficiente para la mayor parte de la lógica, y una segunda placa, basada en una Spartan-3 400, y denominada AER-Scorbot [20] se conectó al AER-Node a través de dos interfaces de bus AER para expandir los pulsos de salida de la parte integradora y derivativa del controlador, y para convertir en pulsos la salida de los codificadores ópticos. Por lo tanto, el AER-Scorbot se conectó directamente a los motores de corriente continua.

En el caso del ED-Scorbot, esa lógica se integró y mejoró en el PL de un módulo Xilinx Mini-Module Plus 7Z100 (Zynq) [26]. Dado que este robot no incluía sensores propioceptivos, la única forma de conocer la posición real de las articulaciones es a través de los encoders ópticos. Por lo tanto, para leer las posiciones de las articulaciones, el ED-Scorbot utiliza un registro de 16 bits para cada articulación. Estos registros se implementan en el PL del Zynq, y sus valores se incrementan o decrementan (funcionando así como contadores) en función de la señal procedente del encoder óptico incremental de cada articulación. Esta información se transfiere al sistema de procesamiento (PS) del Zynq a través del bus AXI (Advanced eXtensible Interface). Los contadores proporcionan un método de retroalimentación, a través del protocolo MQTT (Message Queuing Telemetry Transport) que se ejecuta en el PS. A continuación, los valores de los contadores se publican en un topic MQTT. Se utiliza un segundo topic MQTT para recibir las posiciones objetivo de las articulaciones. Para más detalles, consulte el sitio web de Py-EDScorbotTool [1].

A. Controladores de motores

Mediante la modulación PFM, el SPID procesa la información sensorial y acciona los actuadores íntegramente en el dominio pulsante. Mientras que una versión anterior se utilizó para controlar la velocidad de robots móviles con ruedas [28], esta versión se ha adaptado para controlar con precisión las po-

[1] `https://py-edscorbottool.readthedocs.io/en/latest/index.html` (Consultado el 30 de abril de 2026)

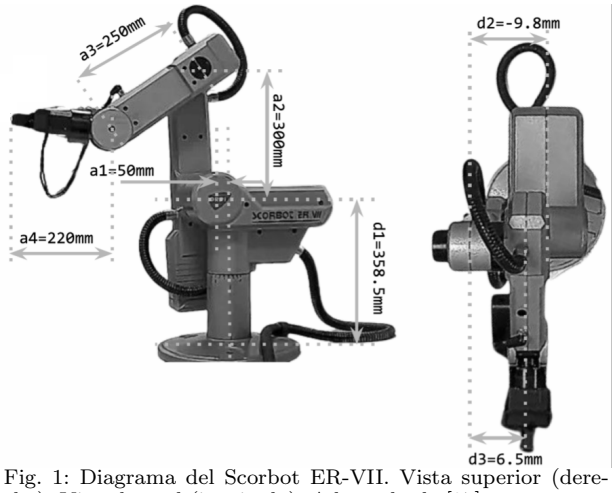


Fig. 1: Diagrama del Scorbot ER-VII. Vista superior (derecha). Vista lateral (izquierda). Adaptado de [27].

siciones de las articulaciones de un brazo robótico articulado [19]. El sistema reproduce la formulación clásica del PID en el dominio S a través de módulos de hardware especializados, conocidos como bloques de procesamiento de señales pulsantes (SSP) [29]. El módulo fundamental es el Spike-Generator, que convierte los valores de control digitales en flujos de pulsos PFM. Diseñado para una implementación eficiente en FPGA, utiliza un contador y un comparador internos controlados por reloj para activar pulsos cada vez que el valor binario en orden inverso del contador cae por debajo de la entrada. Este enfoque algorítmico garantiza computacionalmente una distribución temporal de los pulsos espaciada regularmente y casi uniforme, con un coste de recursos muy bajo.

El término integrador (K_i) del PID es gestionado por el modelo de neurona motora Integrate-And-Generate (I&G). Cuenta con un contador integrador que se incrementa o decrementa en función de la polaridad de los pulsos entrantes, actuando como un análogo digital del potencial de membrana de una neurona biológica. A continuación, un generador conectado emite pulsos proporcionales a este potencial acumulado, produciendo las tasas de disparo rápido típicas de las neuronas motoras biológicas. La constante de integración (K_i) viene definida matemáticamente por el tamaño del contador, la frecuencia de reloj y un divisor de frecuencia. Para calcular las diferencias entre las secuencias de pulsos, el sistema utiliza un bloque Hold-And-Fire (HF). Este módulo espera a que se produzca un pulso y lo retiene temporalmente hasta que llega una segunda entrada. Si los dos pulsos comparten la misma polaridad, se propaga uno; si tienen polaridades opuestas, se anulan entre sí, lo que da como resultado una frecuencia de salida que es la diferencia de las entradas. Al establecer un bucle cerrado entre un bloque H&F y un bloque I&G, el sistema forma el módulo Spike-Derivative. Esta configuración genera una frecuencia que representa la derivada de la frecuencia de entrada, cumpliendo así el requisito K_d de la ecuación PID.

Por último, el Spike-Expander funciona como el componente proporcional (K_p) y sirve como interfaz física final con el hardware. Dado que los motores DC estándar poseen una inercia inherente y actúan como filtros de paso bajo, los pulsos de un solo ciclo de reloj utilizados para el procesamiento interno de la FPGA serían filtrados sin generar ningún movimiento físico. El Spike-Expander alarga la duración de estos pulsos estrechos, lo que les permite ejercer una fuerza mecánica real. La ganancia K_p está directamente vinculada a la anchura del pulso recién ampliado, al periodo de reloj del sistema y a la fuente de alimentación de 12 V del motor. En conjunto, estos bloques SSP interconectados forman el controlador de velocidad SPID completo. Los motores de corriente continua deben llevar incorporado un codificador óptico, cuya señal de salida se convierte en pulsos por cada variación de la señal de salida, que se utilizan para calcular el error mediante un bloque HF. Para adaptar un SPID a su uso como controlador de posición de un brazo robótico articulado, se añade un segundo bloque IG al bucle con el fin de convertir la señal pulsante obtenida de los codificadores ópticos de velocidad articular a posición articular. Al utilizar la sincronización y la frecuencia de pulsos para accionar las articulaciones del brazo, se imita fielmente la forma en que los sistemas nerviosos biológicos accionan los músculos, traduciendo eficazmente los pulsos sensoriomotores en coordenadas cartesianas 3D para el efector final del robot.

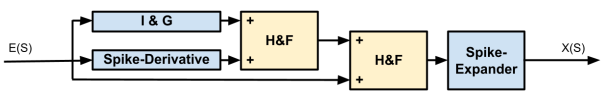


Fig. 2: Controlador PID de impulsos.

La figura 2 muestra el diagrama de bloques del controlador PID pulsante con los bloques descritos. Como se puede observar, la principal diferencia con respecto al controlador PID clásico es que la parte proporcional (Spike-Expander) no se procesa en paralelo a las partes integradora y derivativa, sino al final del proceso.

B. Controlador Zynq 7100 ED-Scorbot

La figura 3 representa la arquitectura. El PS ejecuta Petalinux 2016.4, que admite USB, UART y Ethernet. El sistema de archivos se almacena en la tarjeta SD externa. A través de SSH, un usuario puede acceder al sistema e implementar sus propios scripts para interactuar con el PL a través del bus AXI. A continuación se describe la funcionalidad de cada bloque:

B.1 AXI para la configuración

Los distintos parámetros para configurar la funcionalidad del sistema se asignan en la memoria del PS. Se configura una transferencia AXI para una única dirección de memoria, `0x4000000`, para una palabra de datos de 32 bits. A partir de este valor, se toman los 24 bits menos significativos y se envían a los dos bloques internos principales como una dirección de 8 bits y un bus de datos de 16 bits, junto con una

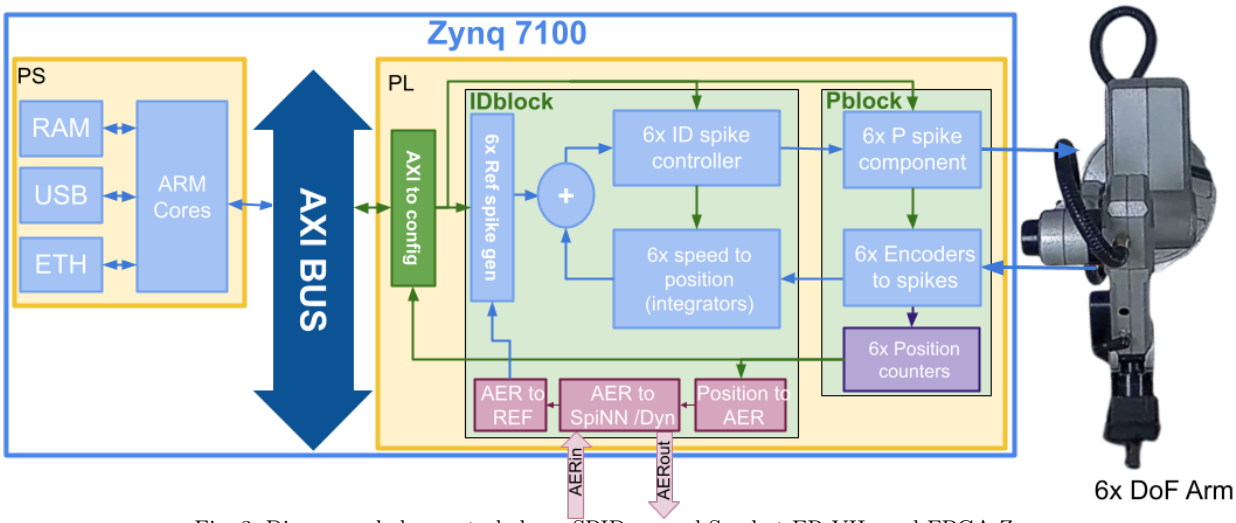


Fig. 3: Diagrama de los controladores SPID para el Scorbot ER-VII en el FPGA Zynq.

señal de strobe, para escribir en cualquiera de los 256 registros de configuración posibles repartidos por el sistema. Este bus de configuración interno (strobe, dirección y datos) se representa en la figura con un bus verde que llega a todos los bloques internos.

B.2 Bloque integrador-derivador.

Este bloque (IDblock en la figura) no solo implementa la parte integradora y derivadora de los controladores, sino que también integra otras funcionalidades importantes: (1) el bloque H&F del bucle cerrado, (2) el I&G para convertir la señal pulsante de velocidad en una señal pulsante de posición para cerrar el bucle, (3) la generación/adaptación de la señal de referencia pulsante de entrada para cada SPID articular, y (4) la interfaz para los chips SpiNNaker y/o Dynap-SE.

El bloque H&F se representa en la figura con el conector "+", mientras que los bloques I&G para la conversión de velocidad a posición se representan con un rectángulo conectado al conector "+". El bloque "Ref spike gen" es un generador de pulsos que obtiene una señal pulsante a partir de un número digital. Este número puede proceder del bus AXI o del chip neuromórfico externo a través del bloque "AER to SpiNN/Dyn". En cuanto a la funcionalidad (4), su implementación depende del chip neuromórfico externo. El código VHDL de este bloque debe traducir el formato de eventos de dicho chip externo a un formato AER plano con el protocolo de handshake clásico, basado en las señales REQ y ACK. Este protocolo AER también se utiliza internamente para conectar el "IDblock" al "Pblock". Dependiendo del significado de los eventos procedentes del chip neuromórfico, el bloque "AER to REF" podría adaptarse para traducir dichos eventos en la referencia adecuada para el "Ref spike gen". Del mismo modo, el bloque "Position to AER" traduce los contadores de posición actuales de las articulaciones en eventos que se enviarán al chip neuromórfico.

Por ejemplo, para el control de una trayectoria suave implementada en SpiNNaker con varias SNN funcionando en ese chip [30], es necesario proporcionar el estado actual de las articulaciones como una señal de disparo cada vez que la posición de cada articulación supera un paso de ángulo de grado configurado. Cada espacio de trabajo operativo de las articulaciones se divide en 16 secciones; por lo tanto, se envía un nuevo evento a SpiNNaker cuando la articulación pasa de una sección a la siguiente. Del mismo modo, cuando la SNN que se ejecuta en SpiNNaker decide que una articulación concreta debe desplazarse de su posición actual a la siguiente, se recibe un evento de SpiNNaker que se convierte en la referencia pulsante correspondiente para mover esa articulación al siguiente ángulo objetivo.

Cada bloque "ID spike controller" representa la parte integradora y derivativa del PID. La parte proporcional ("P spike component") se delega al "Pblock", donde la comunicación se basa en buses AER.

B.3 Bloque proporcional.

Este bloque ("Pblock" en la figura) prepara la señal pulsante de salida integradora-derivativa AER, que se enviará a los motores DC, utilizando el circuito "Spike-Expander". Por cada evento de entrada, se envía un pulso con una anchura configurable a los controladores de los motores. Esta anchura configurada se proporciona a través del bloque "AXI to config" (bus verde en la figura). Además, este "Pblock" gestiona la salida de los encoders ópticos que se utilizan en el bucle de control. Por cada cambio en los canales A o B del encoder, se genera un evento AER y se envía al "IDblock" para el control de retroalimentación. Estos eventos proporcionan una polaridad que indica si el movimiento de la articulación es en sentido horario o antihorario. Por último, a partir de estos eventos, se alimentan unos contadores de 16 bits para proporcionar la posición global de las articulaciones. Estos contadores pueden leerse, a través del bus AXI, desde los scripts de PS utilizando direcciones a partir de `0x4000001:6`, o bien pueden ser convertidos a AER para el chip neuromórfico externo mediante el "IDblock", tal y como se ha mencionado en el párrafo anterior.

III. Resultados

El sistema descrito en la sección anterior está programado en VHDL y se ha sintetizado e implementado en la plataforma AvNet MMP, basada en el Zynq 7100, utilizando Vivado 2022.1. La tabla I resume los recursos necesarios. El procesamiento basado en pulsos requiere básicamente contadores ascendentes y descendentes y comparadores de diferentes resoluciones. Dado que los parámetros de configuración se asignan a registros, no se necesitan DSP ni memoria RAM en bloque, a menos que se inserte una ILA en el diseño con fines de depuración. Esta escasa necesidad de recursos hace que este controlador neuromórfico sea viable para robots con muchos grados de libertad, o para el control de múltiples robots con el mismo dispositivo.

La tabla II presenta una lista de los parámetros internos de nuestra implementación en Vivado. Tras la implementación con una frecuencia de reloj PL de 50 MHz, el parámetro Worst Negative Slack (WSN) fue de 1,47 ns. En cuanto al consumo de energía, Vivado estima un consumo total en el chip de 2,04 W, con una tasa de conmutación del 50 % y una temperatura de unión de 28,6 °C. Esta potencia total se divide en 1,8 W de potencia dinámica y 245 mW de potencia estática. De la potencia dinámica, 1,57 W son consumidos por el PS y 232 mW por el PL, siendo 106 mW para el PLL, 63 mW para los relojes, 22 mW para las E/S y 39 mW para la lógica y el enrutamiento.

IV. Conclusiones

La implementación del controlador SPID en la plataforma Zynq 7100 demuestra la viabilidad de utilizar metodologías sintéticas para salvar la brecha entre la adaptabilidad biológica y la eficiencia computacional en robótica. Al procesar la información sensorial y controlar los actuadores íntegramente en el dominio pulsante, el sistema emula con éxito los bucles sensoriomotores que se dan en los organismos biológicos. La arquitectura demuestra ser altamente eficiente en cuanto a recursos, ya que se basa principalmente en contadores ascendentes y descendentes y comparadores, sin necesidad de DSP ni RAM en bloque, lo que facilita el control de múltiples robots en un único dispositivo. Además, el sistema presenta un perfil de bajo consumo, con un consumo dinámico de solo 232 mW por parte del PL, lo que lo hace adecuado para aplicaciones de IA en el borde. La integración satisfactoria del marco con aceleradores de hardware SNN avanzados como SpiNNaker y Dynap-SE confirma su capacidad para el control de trayectorias complejo en bucle cerrado. Las investigaciones futuras continuarán optimizando los controladores de motores de impulsos para reducir aún más la latencia y el consumo de energía, abordando los retos pendientes en el campo de la robótica neuromórfica.

Agradecimientos

Este trabajo ha sido parcialmente soportado por el Ministerio de Transformación Digital y Función Pública de España mediante la subvención USE-CHIP (TSI-069100-2023-001) del programa PERTE "Cátedra Chip", financiado por la Unión Europea – Next Generation EU, por NEKOR (PID2023-149071NB-C54/AEI/10.13039/501100011033) y por la acción complementaria PAIDI 2020 (QUAL21 008 USE) de la Junta de Andalucía. D.E.T.-H. cuenta con una beca del SENACYT para cursar un doctorado en la Universidad de Sevilla.

Referencias

[1] W Grey Walter, "An imitation of life," *Scientific American*, vol. 182, no. 5, pp. 42–45, 1950.

[2] William Grey Walter, *The Living Brain*, W. W. Norton & Company, 1953.

[3] Valentino Braitenberg, *Vehicles: Experiments in Synthetic Psychology*, MIT Press, Cambridge, MA, 1984.

[4] Karel Čapek, *R.U.R. (Rossum's Universal Robots)*, Aventinum, Prague, 1920.

[5] J. Krichmar and H. Wagatsuma, *Neuromorphic and Brain-Based Robots*, Cambridge University Press, 2011.

[6] Charles Sherrington, *The integrative action of the nervous system*, CUP Archive, 1952.

[7] Arthur C Guyton, John Edward Hall, et al., *Textbook of medical physiology*, vol. 548, Saunders Philadelphia, 1986.

[8] Giacomo Indiveri and Rodney Douglas, "Neuromorphic vision sensors," *Science*, vol. 288, no. 5469, pp. 1189–1190, 2000.

[9] Chiara Bartolozzi, Giacomo Indiveri, and Elisa Donati, "Embodied neuromorphic intelligence," *Nature Communications*, vol. 13, no. 1024, 2022.

[10] Chuanxin M Niu, Kian Jalaleddini, Won Joon Sohn, John Rocamora, Terence D Sanger, and Francisco J Valero-Cuevas, "Neuromorphic meets neuromechanics, part i: the methodology and implementation," *Journal of Neural Engineering*, vol. 14, no. 2, pp. 025001, feb 2017.

[11] Jörg Conradt, Francesco Galluppi, and Terrence C. Stewart, "Trainable sensorimotor mapping in a neuromorphic robot," *Robotics and Autonomous Systems*, vol. 71, pp. 60–68, 2015, Emerging Spatial Competences: From Machine Perception to Sensorimotor Intelligence.

[12] Daniel Gutierrez-Galan, Juan P. Dominguez-Morales, Fernando Perez-Peña, Angel Jimenez-Fernandez, and Alejandro Linares-Barranco, "Neuropod: A real-time neuromorphic spiking cpg applied to robotics," *Neurocomputing*, vol. 381, pp. 10–19, 2020.

[13] Imke Krauhausen, Dimitrios A. Koutsouras, Armantas Melianas, Scott T. Keene, Katharina Lieberth, Hadrien Ledanseur, Rajendar Sheelamanthula, Alexander Giovannitti, Fabrizio Torricelli, Iain Mcculloch, Paul W. M. Blom, Alberto Salleo, Yoeri van de Burgt, and Paschalis Gkoupidenis, "Organic neuromorphic electronics for sensorimotor integration and learning in robotics," *Science Advances*, vol. 7, no. 50, pp. eabl5068, 2021.

[14] Alexander D. Rast, Samantha V. Adams, Simon Davidson, Sergio Davies, Michael Hopkins, Andrew Rowley, Alan Barry Stokes, Thomas Wennekers, Steve Furber, and Angelo Cangelosi, "Behavioral learning in a cognitive neuromorphic robot: An integrative approach," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 12, pp. 6132–6144, 2018.

[15] Pierre Daye, Sio-Hoi Ieng, and Ryad Benosman, "A theory for sparse event-based closed loop control," *Frontiers in Neuroscience*, vol. 13, 2019.

[16] Angel Jimenez-Fernandez, Gabriel Jimenez-Moreno, Alejandro Linares-Barranco, Manuel J. Dominguez-Morales, Rafael Paz-Vicente, and Anton Civit-Balcells, "A neuro-inspired spike-based pid motor controller for multi-motor robots with low cost fpgas," *Sensors*, vol. 12, no. 4, pp. 3831–3856, 2012.

[17] Daniel Bullock and Stephen Grossberg, "Neural dynamics of planned arm movements: emergent invariants and speed-accuracy properties during trajectory formation.," *Psychological review*, vol. 95, no. 1, pp. 49, 1988.

[18] Fernando Perez-Peña, Arturo Morgado-Estevez, Alejandro Linares-Barranco, Angel Jimenez-Fernandez, Francisco Gomez-Rodriguez, Gabriel Jimenez-Moreno, and Juan Lopez-Coronado, "Neuro-inspired spike-based motion: From dynamic vision sensor to robot motor open-

Tabla I: Consumo de recursos en Zynq 7100.

Recurso	Máximo disponible	Consumido	Consumido (%)
LUT de segmento	277 400	13 483	4,8
Registros de segmento	554 800	11 190	2,0
LUT como lógica	277 400	12 821	4,6
LUT como memoria	108 200	662	0,6
RAM en bloque	755	0	0,0

Tabla II: Parámetros de implementación de Zynq.

Parámetro	Valor (unidades)
Entrada PS F_{clk}	33,33 (MHz)
ARM F_{clk}	667 (MHz)
DDR3 F_{clk}	533,33 (MHz)
PL F_{clk}	50 (MHz)
WNS	1,46 (ns)
F_{clk} máxima de PL	53,95 (MHz)
Configuración de síntesis	valores predeterminados de Vivado
Configuración de implementación	valores predeterminados de Vivado

loop control through spike-vite," *Sensors*, vol. 13, no. 11, pp. 15805–15832, 2013.

[19] Alejandro Linares-Barranco, Fernando Perez-Peña, Angel Jimenez-Fernandez, and Elisabetta Chicca, "Edbiorob: a neuromorphic robotic arm with fpga-based infrastructure for bio-inspired spiking motor controllers," *Frontiers in neurorobotics*, vol. 14, pp. 590163, 2020.

[20] F. Gomez-Rodriguez, A. Jimenez-Fernandez, F. Perez-Pena, L. Miro, M.J. Dominguez-Morales, A. Rios-Navarro, E. Cerezuela, D. Cascado-Caballero, and A. Linares-Barranco, "ED-Scorbot: A robotic test-bed framework for FPGA-based neuromorphic systems," in *Proceedings of the IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics*, 2016, vol. 2016-July.

[21] Salvador Canas-Moreno, Enrique Piñero-Fuentes, Antonio Rios-Navarro, Daniel Cascado-Caballero, Fernando Perez-Peña, and Alejandro Linares-Barranco, "Towards neuromorphic fpga-based infrastructures for a robotic arm," *Autonomous Robots*, vol. 47, no. 7, pp. 947–961, 2023.

[22] Daniel Casanueva-Morato, Pablo Lopez-Osorio, Enrique Piñero-Fuentes, Juan P Dominguez-Morales, Fernando Perez-Peña, and Alejandro Linares-Barranco, "Integrating a hippocampus memory model into a neuromorphic robotic-arm for trajectory navigation," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2024, pp. 1–5.

[23] Daniel Casanueva-Morato, Chenxi Wu, Giacomo Indiveri, Juan P. Dominguez-Morales, and Alejandro Linares-Barranco, "Space-time smooth control of closed-loop neuromorphic robotic arms using spiking neural networks," in *2025 International Joint Conference on Neural Networks (IJCNN)*, 2025, pp. 1–9.

[24] Diógenes Trejos-Hernández, Elena Cerezuela-Escudero, and Alejandro Linares-Barranco, "On the simulation of a neuromorphic robotic arm with dynamic control for rapid trajectories evaluation," *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, vol. 603 LNICST, pp. 221–238, 2025.

[25] T. Iakymchuk, A. Rosado, T. Serrano-Gotarredona, B. Linares-Barranco, A. Jiménez-Fernández, A. Linares-Barranco, and G. Jiménez-Moreno, "An aer handshakeless modular infrastructure pcb with x8 2.5gbps lvds serial links," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 1556–1559.

[26] E. Piñero-Fuentes, D.A. Curra-Sosa, D. Cascado, S. Canas, A. Rios-Navarro, F. Gomez-Rodriguez, A. Jimenez-Fernandez, and A. Linares-Barranco, "A hybrid neuromorphic robotic-arm lemniscate trajectories dataset for learning-based applications," in *2024 31st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Nov 2024, pp. 1–4.

[27] Thomas Ortner, Horst Petschenig, Athanasios Vasilopoulos, Roland Renner, Špela Brglez, Thomas Limbacher, Enrique Piñero, Alejandro Linares-Barranco, Angeliki Pantazi, and Robert Legenstein, "Rapid learning with phase-change memory-based in-memory computing through learning-to-learn," *Nature Communications 2025 16:1*, vol. 16, pp. 1243–, 2 2025.

[28] A Jimenez-Fernandez, R Paz-Vicente, M Rivas, A Linares-Barranco, G Jimenez, and A Civit, "Aer-based robotic closed-loop control system," in *2008 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2008, pp. 1044–1047.

[29] A. Jimenez-Fernandez, A. Linares-Barranco, R. Paz-Vicente, G. Jiménez, and A. Civit, "Building blocks for spikes signals processing," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 2010, pp. 1–8.

[30] Daniel Casanueva-Morato, Chenxi Wu, Giacomo Indiveri, Alejandro Linares-Barranco, and Juan P Dominguez-Morales, "Toward a dynamic closed-loop neuromorphic trajectory interpolation model: Hardware emulation and software simulation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 72, no. 12, pp. 1892–1896, 2025.

Pruebas de Exactitud y Precisión del brazo robótico neuromórfico ED-Scorbot

Diógenes Enrique Trejos Hernández[1], Elena Cerezuela Escudero[2] y Alejandro Linares Barranco[3]

***Resumen*—En esta investigación, analizamos el rendimiento en cuanto a exactitud y precisión (P-ACC) del brazo robótico ED-Scorbot, controlado mediante controladores SPID (Spiking Proportional-Integral-Derivative) inspirados en la neurociencia e implementados en FPGA (Field-Programmable Gate Array). Esta implementación ofrece ventajas en términos de consumo energético, utilización del hardware y eficiencia temporal (latencia). Sin embargo, también presenta ciertas limitaciones que afectan a la precisión y la exactitud del robot. El objetivo principal de este estudio es identificar qué variaciones de parámetros —concretamente, el intervalo de transmisión de comandos (CTI) al robot y la distancia recorrida por el punto central de la herramienta (TCP) entre puntos sucesivos a lo largo de una trayectoria de referencia conducen al menor error en las pruebas P-ACC. La exactitud y la repetibilidad se evaluaron de acuerdo con las definiciones establecidas en la norma ISO 9283, que tienen en cuenta la repetición del movimiento hacia un objetivo para el cálculo de la desviación media del TCP. Inicialmente, las pruebas se realizaron de forma independiente en cada articulación, y posteriormente se ejecutaron trayectorias con multiples repeticiones, lo que dio como resultado un error mínimo de 13.17 mm para un CTI de 1 segundo y una distancia punto a punto (PTPD) de 11.8 mm.**

***Index Terms*—Neuromórfico, FPGA, ED-Scorbot, dominio pulsante, sistema de control, exactitud y precisión.**

I. Introducción

La aplicación de robots en entornos dinámicos y no estructurados, en los que los robots deben tomar decisiones autónomas basadas en información sensorial, sigue siendo un importante reto de ingeniería. Sin embargo, en los últimos años se han logrado avances significativos con los vehículos autónomos y numerosos robots de campo, en los que la enorme cantidad de datos generados por sus sensores (LIDAR, radar, cámaras, etc.) se procesa en milisegundos de forma local en ordenadores de alto rendimiento. Estos suelen ejecutar múltiples redes neuronales profundas (DNN), como las redes neuronales convolucionales (CNN) para el procesamiento de imágenes y la detección de objetos, así como las redes neuronales recurrentes (RNN), para tareas de seguimiento y predicción de trayectorias. Si bien los sistemas inteligentes actuales ya no presentan grandes limitaciones tecnológicas, su mayor complejidad conlleva un mayor consumo de energía. Los estudios revelan que entre el 5 y el 15 % de la energía mundial se dedica a la manipulación de datos (procesamiento o transmisión), una fracción que aumentará exponencialmente con el Internet de las cosas (IoT) [1]. La computación neuromórfica busca reducir el consumo de energía mediante enfoques inspirados en sistemas biológicos, como las redes neuronales pulsantes (SNN) y el protocolo de representación de eventos por dirección (AER). En este esquema, la información se procesa y transmite únicamente cuando se producen eventos neuronales (pulsos), lo que reduce significativamente el tráfico de datos y el consumo de energía [2].

El hardware neuromórfico incluye sensores de visión dinámica (DVS), cócleas de silicio y narices electrónicas, dispositivos que capturan información de forma eficiente y basada en eventos. Además, los procesadores ASIC como Loihi (Intel) y TrueNorth (IBM) son ejemplos de circuitos digitales capaces de simular millones de neuronas y sinapsis [2] [3]. Aunque estos chips ASIC son extremadamente eficientes, su principal limitación técnica es la falta de flexibilidad en la asignación de recursos, ya que sus núcleos suelen estar preconfigurados para tareas específicas. Como alternativa, los ingenieros están optando por utilizar FPGA, ya que son plataformas reconfigurables y versátiles para la implementación, la creación de prototipos [4] y la aceleración de redes neuronales convencionales [5]. Algunas de las diversas placas basadas en FPGA incorporan procesadores ARM integrados, como el Xilinx Zynq 7000 [1], lo que permite la implementación de software integrado, como versiones reducidas de Linux.

En este trabajo utilizamos el ED-Scorbot, compuesto por un brazo robótico Scorbot ER-VII, que incorpora un control SPID por articulación. Aunque este control ha demostrado ventajas en términos de eficiencia y baja latencia, presenta limitaciones en cuanto a precisión. Esto impide que el robot alcance correctamente las posiciones comandadas, debido a las latencias intrínsecas del controlador durante la estabilización de las trayectorias, resultantes de un ajuste no óptimo de las constantes PID, y a limitaciones físicas y mecánicas, como el desgaste de ciertas articulaciones debido a la antigüedad del robot [6]. También es importante señalar que, aunque el PID (y su variante neuromórfica SPID) puede estabilizar la articulación en un tiempo finito, no es tan eficaz en sistemas no lineales (brazo robótico), ya que, debido a sus ganancias fijas, no pueden contrarrestar efectos dinámicos tales como: inercia, gravedad, fuerzas de Coriolis, etc. [7] [8]. Partiendo de estas consideraciones, el objetivo de este trabajo fue reducir el error entre la posición deseada y la posición alcanzada, sin modificar la lógica programable (PL) del sistema. Para ello, se llevó a cabo un estudio de precisión y exactitud (P-ACC) de acuerdo con las definiciones de la norma ISO 9283 [9].

Laboratorio de Robótica y Arquitectura y Tecnología de computadores ETSII-EPS. I3US. SCORE., Universidad de Sevilla, Sevilla, España. e-mail: diotreher@alum.us.es.

[1]https://www.amd.com/es/products/adaptive-socs-and-fpgas/soc/zynq-7000.html, consultado el 10 de mayo de 2026

Las pruebas se llevaron a cabo controlando parámetros como los intervalos de tiempo entre el envío de comandos (CTI) y las distancias entre puntos (PTPD) en una trayectoria de referencia. El objetivo es determinar los valores óptimos de estos parámetros que minimizaran el error de posición.

El artículo esta organizado de la siguiente manera: en la sección 2 se detallan los materiales y métodos utilizados para desarrollar las pruebas de exactitud y precisión; en la sección 3 se presentan los resultados experimentales; y, por último, en la sección 4 se exponen las conclusiones.

II. Materiales y Métodos

II-A. Infraestructura Hardware

El ED-Scorbot (Fig. 1) es un brazo robótico neuromórfico compuesto por seis grados de libertad (GDL) y seis motores de 12 V CC que, mediante transmisiones, generan movimiento en los ejes de sus cinco articulaciones giratorias y abren y cierran una pinza (sexta articulación) [8]. Cada motor es accionado por un controlador PID basado en eventos (SPID), que utiliza modulación por frecuencia de pulso (PFM) para suministrar energía a los motores. A diferencia de la modulación por ancho de pulso (PWM), que se utiliza tradicionalmente en robótica, la PFM reproduce el funcionamiento de los sistemas biológicos, en los que los músculos se activan mediante ráfagas de pulsos generadas por las neuronas motoras [6].

El control de cada motor está integrado en la PL de un módulo Xilinx Mini-Module Plus 7Z100 (Zynq), lo que permite el control de cada articulación en el dominio pulsante [10].

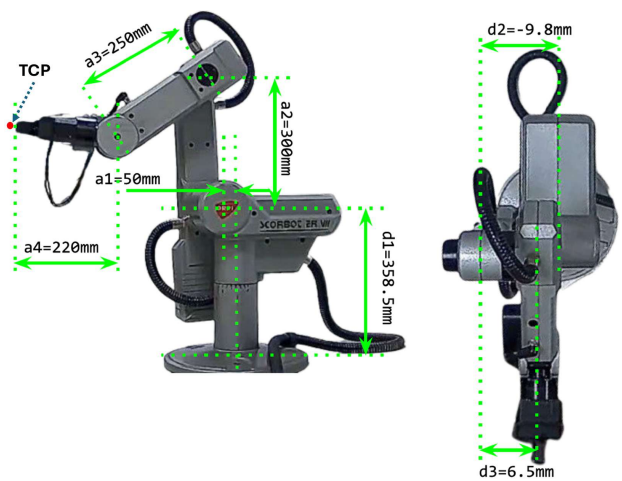


Figura 1. Esquema del Scorbot ER-VII. Vista superior (derecha). Vista lateral (izquierda). Adaptado de [11].

Para leer las posiciones de las articulaciones, el ED-Scorbot utiliza un registro de 16 bits para cada articulación. Estos registros están implementados en la PL de la Zynq y sus valores se incrementan o disminuyen (funcionando así como contadores) en función de la señal procedente del codificador óptico incremental de cada articulación. Esta información se transfiere al sistema de procesamiento (PS) de la Zynq a través del bus AXI (Advanced eXtensible Interface). Los contadores proporcionan un método de retroalimentación a través del protocolo MQTT (Message Queuing Telemetry Transport) que se ejecuta en el PS. A continuación, los valores de los contadores se publican en un topic MQTT. Para obtener más detalles sobre ED-Scorbot y su plataforma desarrollada en trabajos anteriores, visite el sitio web de Py-EDScorbotTool [2].

II-B. Descripción Cinemática

El robot tiene seis articulaciones, pero en este trabajo solo se utilizaron las cuatro primeras, ya que son suficientes para situar el TCP en el punto deseado dentro del espacio de trabajo. La tabla I presenta los límites de las articulaciones en valores de refencias digitales (SR) y sus ángulos correspondientes en grados, mientras que la tabla II ofrece la descripción cinemática del robot utilizando los parámetros de Denavit-Hartenberg.

Cuadro I
Límites superior e inferior de las articulaciones 1 a 4. (Tomado de [6]).

Articulación	Superior (SR)	Inferior (SR)	Superior (Grados)	Inferior (Grados)
J_1	487	-487	155	-155
J_2	750	-750	85	-85
J_3	383	-383	112.5	-112.5
J_4	1585	-1585	90	-90

Cuadro II
Parámetros de Denavit-Hartenberg (Adaptado de [10]).

Art.	θ_i	d_i(m)	a_i(m)	α_i(Radianes)
J_1	q_1	0.3585	0.050	$-\frac{\pi}{2}$
J_2	q_2	-0.098	0.300	0
J_3	q_3	0.0650	0.250	0
J_4	q_4	0	0.220	0

El vector de posiciones de las articulaciones se representa mediante la variable "q". Este vector contiene una lista de todas las configuraciones de las articulaciones necesarias para que el TCP pase por cada punto de una trayectoria de referencia. Las posiciones angulares de cada articulación se representaron mediante q_1, q_2, q_3, q_4, tal y como se muestra en la Tabla II; por lo tanto, $q = [q_1, q_2, q_3, q_4]$.

Cabe señalar que los SPID solo pueden recibir referencias en frecuencia de pulsos, por lo que fue necesario convertir las posiciones de las articulaciones de grados a valores SR utilizando las ecuaciones dadas en (1), derivadas de un estudio de caracterización de las articulaciones (Fig. 2). Este estudio se realizó de forma independiente para cada articulación. Durante este proceso de caracterización, se esperó un tiempo de más de dos minutos entre comandos para permitir que el controlador del SPID estabilizara la articulación en la posición objetivo.

[2]https://py-edscorbottool.readthedocs.io/en/latest/index.html(Consultado el 10 de mayo de 2026)

Estos valores en formato de SR se envían a generadores de impulsos integrados que proporcionan la señal de referencia de impulsos a cada SPID.

$$\begin{aligned} q_{SR_1} &= \lfloor (q_1 \times 3.050640635) \rfloor \\ q_{SR_2} &= \lfloor (q_2 \times 9.532888465) \rfloor \\ q_{SR_3} &= \lfloor (q_3 \times 3.220611916) \rfloor \\ q_{SR_4} &= \lfloor (q_4 \times 17.66784452) \rfloor \end{aligned} \tag{1}$$

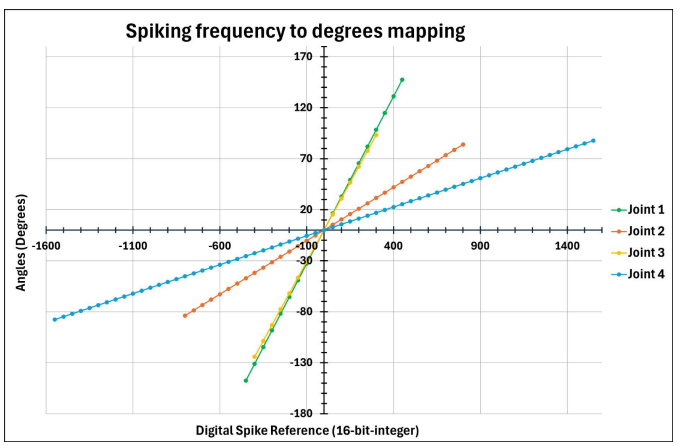


Figura 2. Caracterización de las articulaciones 1–4. Adaptado de [6]. En el eje horizontal, se indican los valores de posición de las articulaciones en SR, mientras que en el eje vertical esos mismos valores se expresan en grados.

Los resultados de (1) se obtienen tras realizar una conversión a un número entero. Cada uno de estos valores está limitado a un tamaño de 16 bits.

Los valores del contador pueden convertirse a grados utilizando las ecuaciones de (2), tal y como se describe en un trabajo anterior [6].

$$\begin{aligned} q_1 &= \left[\left(\frac{1}{125.5} \right) \times (32768 - \mathrm{COUNT}_1) \right] \\ q_2 &= \left[\left(\frac{1}{131.0} \right) \times (32768 - \mathrm{COUNT}_2) \right] \\ q_3 &= \left[\left(\frac{1}{132.5} \right) \times (32768 - \mathrm{COUNT}_3) \right] \\ q_4 &= [0.0123916 \times (32768 - \mathrm{COUNT}_4)] \end{aligned} \tag{2}$$

II-C. Software

- Las trayectorias de referencia del TCP en el espacio cartesiano se definieron utilizando MATLAB®. Los cálculos de cinemática directa (CD) y cinemática inversa (CI) se realizaron con la Robotics System Toolbox (RST). Las posiciones de las articulaciones (q en grados) obtenidas mediante CI se transformaron al dominio pulsante (1) y se almacenaron en un archivo `.json`. Tras los experimentos, se utilizó MATLAB® para analizar los datos, calcular las desviaciones del TCP en el espacio articular y evaluar los errores en el espacio cartesiano mediante CD.
  La RST necesita el modelo URDF (Unified Robot Description Format) del robot. Para alinear el modelo con el robot real, se aplicaron desplazamientos en radianes a las articulaciones 1-3: $\theta^1_{yaw} = 0.4119$, $\theta^2_{pitch} = -0.3840$ y $\theta^3_{pitch} = 0.2094$.
- **traj_exec.py**: lee el archivo `.json` que contiene las posiciones de referencia de las articulaciones y las transmite al dispositivo Zynq a través del protocolo UDP (User Datagram Protocol). Se utilizó UDP para el intercambio bidireccional de datos entre el ordenador host y los scripts que se ejecutan en la Zynq.
- **read_joints.cpp**: se ejecuta en el PS (Fig. 3), inicializa el MQTT y publica los datos de posición de las articulaciones, obtenidos de los contadores, en el tema `EDScorbot/trajectory`, permitiendo el acceso en tiempo real a los usuarios suscritos.
- **udp_sendRef.py**: se ejecuta en el PS (Fig. 3), recibe las posiciones de las articulaciones desde `traj_exec.py` via UDP y envía cada referencia al control motor correspondiente en la PL a través del bus AXI utilizando la función `sendRef`. El script también se suscribe al tema MQTT `EDScorbot/trajectory` para registrar los datos del contador y las marcas de tiempo en un archivo `.csv`.

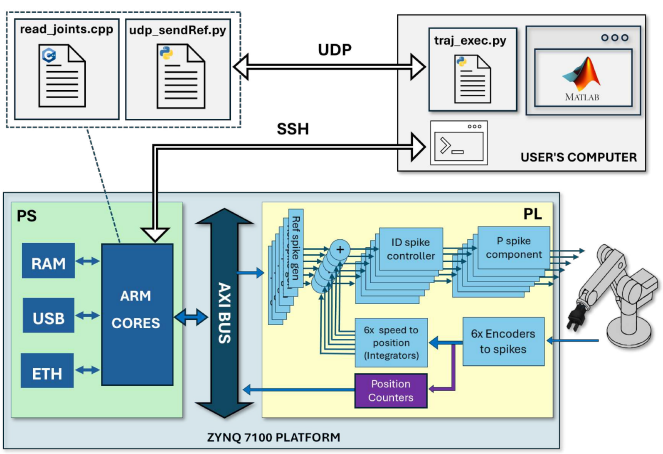


Figura 3. Diagrama de conexión entre los scripts de prueba y la plataforma ED-Scorbot. Utilizamos el protocolo SSH (Secure Shell) para establecer una sesión de terminal remota con la Zynq, lo que permite la transferencia de archivos y la ejecución de scripts.

II-D. Pruebas con las trayectorias

La ecuación (3) describe los vértices (en metros) de una trayectoria cuadrada en el espacio cartesiano. Se insertaron puntos intermedios adicionales entre los vértices, lo que dio lugar a diferentes distancias entre puntos consecutivos. Para las trayectorias cuadradas que contienen 13, 21, 37 y 69 puntos, los valores resultantes de PTPD fueron 66.7 mm, 40.0 mm, 22.2 mm y 11.8 mm, respectivamente.

$$
\begin{aligned}
x &= [\,0.50 \;\; 0.50 \;\; 0.50 \;\; 0.50 \;\; 0.50\,] \\
y &= [\,0.20 \;\; 0.20 \;\; 0.00 \;\; 0.00 \;\; 0.20\,] \\
z &= [\,0.50 \;\; 0.70 \;\; 0.70 \;\; 0.50 \;\; 0.50\,]
\end{aligned} \tag{3}
$$

Una trayectoria en lemniscata corresponde a un recorrido con forma de ocho. A diferencia de la trayectoria cuadrada, para la trayectoria en lemniscata se obtuvo una serie de puntos (en metros) utilizando las ecuaciones (4). Donde la variable t es el vector de tiempo; k denota el índice del punto de la trayectoria y n es el número total de puntos de la trayectoria. La constante a se refiere a la escala de la lemniscata. En este trabajo, utilizaremos una escala de 0.25 para que todos los puntos permanezcan dentro del espacio de trabajo del robot.

$$
\begin{aligned}
t_k &= \frac{2\pi k}{n-1}, \qquad k = 0, 1, \ldots, n-1 \\
t &= \{t_0, t_1, \ldots, t_{n-1}\} \\
x(t_k) &= 0.40 + \frac{a \sin(t_k)\cos(t_k)}{1 + \cos^2(t_k)} \\
y(t_k) &= -0.0384 + \frac{a \sin(t_k)}{1 + \cos^2(t_k)} \\
z(t_k) &= 0.35
\end{aligned} \tag{4}
$$

Posteriormente, esta trayectoria se giró primero alrededor del eje x en un ángulo de $-\frac{\pi}{3}$ y, a continuación, se giró de nuevo alrededor del eje z en un ángulo de $\frac{\pi}{3}$.

Para las trayectorias de lemniscata con n = 50, 100, 200, los valores resultantes de PTPD fueron 26.7 mm, 13.2 mm y 6.6 mm, respectivamente. Estos valores son aproximados, ya que la trayectoria es curva.

III. Resultados Experimentales

Las pruebas experimentales se llevaron a cabo en dos fases. En la primera fase, se evaluó el comportamiento de cada articulación de forma independiente para identificar sus limitaciones individuales. Para ello, se ejecutaron desplazamientos angulares de $0° \rightarrow 40° \rightarrow 0°$ en las articulaciones 1, 3 y 4, y de $0° \rightarrow -40° \rightarrow 0°$ en la articulación 2, utilizando incrementos de 1°, 2°, 5°, 8° y 10°. Cada prueba se repitió cuatro veces seguidas y se realizó con diferentes intervalos de envío de comandos (1 s, 500 ms y 250 ms).

La segunda fase constó de dos partes. En primer lugar, se ejecutó una trayectoria de referencia cuadrada en el espacio cartesiano, tal y como se describe en (3), variando el PTPD (66.7 mm, 40.0 mm, 22.2 mm y 11.8 mm) y utilizando el mismo CTI definido en la primera fase. En la segunda parte, se llevó a cabo una trayectoria en lemniscata (4) utilizando tres combinaciones diferentes de PTPD–CTI: 26.7 mm - 1 s, 13,2 mm - 500 ms y 6,6 mm - 250 ms. Las combinaciones se seleccionaron de modo que el tiempo total de ejecución de la trayectoria fuera el mismo en los tres casos, manteniendo una relación constante entre ambos parámetros. El objetivo de este análisis es evaluar si las variaciones conjuntas de PTPD y CTI producen cambios significativos en el RMSE del TCP.

Se utilizó el RMSE para calcular el error TCP (5) a partir de las coordenadas cartesianas de los puntos de la trayectoria de referencia (x_k, y_k, z_k) y las coordenadas de los puntos de la trayectoria estimada $(\hat{x}_k, \hat{y}_k, \hat{z}_k)$. El error conjunto se calculó utilizando (6).

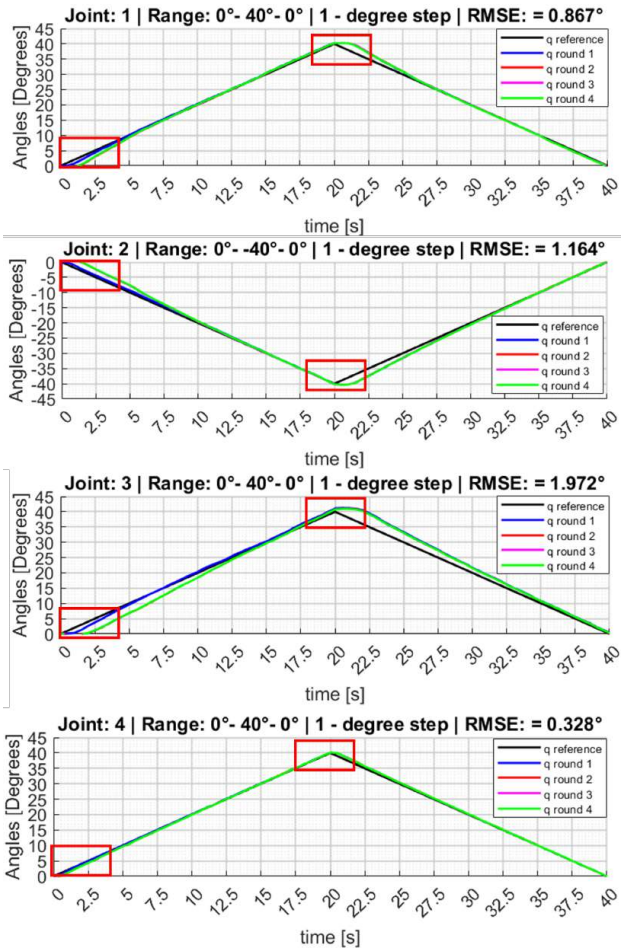


Figura 4. Prueba de cada articulación por separado, utilizando un CTI de 500 ms, un incremento de 1° y cuatro repeticiones por serie. El recuadro rojo indica las secciones en las que se produce el mayor error, debido principalmente al cambio en la dirección de rotación de la articulación. La tabla III muestra los detalles de los demás incrementos angulares y los diferentes CTI.

$$
\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{k=1}^{n}\left[(x_k - \hat{x}_k)^2 + (y_k - \hat{y}_k)^2 + (z_k - \hat{z}_k)^2\right]} \tag{5}
$$

$$
\mathrm{RMSE_J} = \sqrt{\frac{1}{n}\sum_{k=1}^{n}\left(q_j^{\mathrm{ref}}(k) - q_j^{\mathrm{meas}}(k)\right)^2} \tag{6}
$$

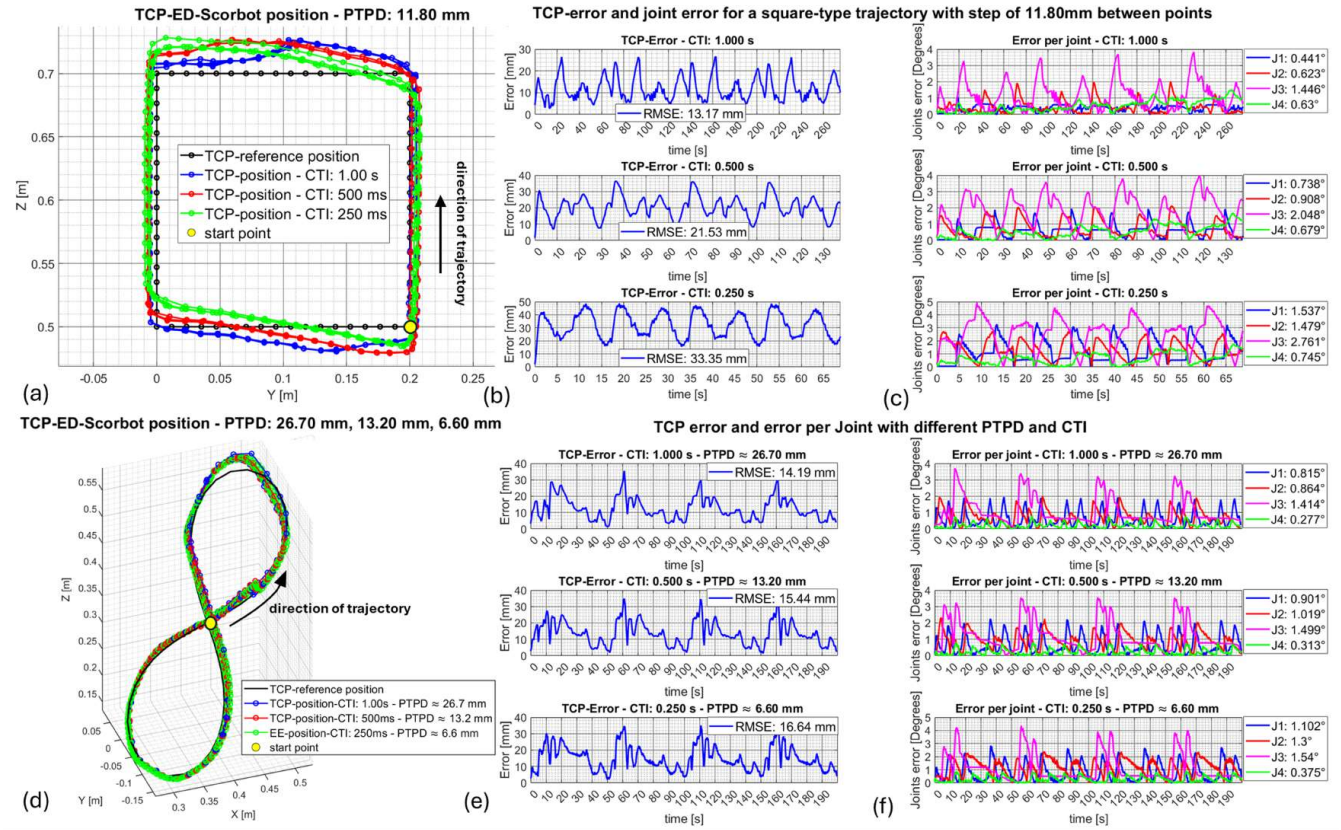


Figura 5. (a) Gráfico en 2D de la trayectoria cuadrada realizada por el robot con cuatro repeticiones consecutivas, utilizando un PTPD de 11,8 mm y valores variables de CTI. (b) y (c) Error del TCP y errores de cada articulación para la trayectoria cuadrada. (d) Gráfico en 3D de la trayectoria en lemniscata para múltiples combinaciones de PTPD y CTI con cuatro repeticiones consecutivas. (e) y (f) Error del TCP y error por articulación para la trayectoria en forma de lemniscata.

La Fig. 4 y la tabla III ilustran las limitaciones de las articulaciones del robot observadas en las pruebas individuales. La articulación 4 presenta la mayor precisión y exactitud, ya que se estabiliza casi instantáneamente; esto se debe a que su actuador solo mueve el último eslabón de la cadena y la pinza. Por el contrario, las demás articulaciones deben compensar el peso de los eslabones posteriores. A partir de la Fig. 4, también resulta evidente que la articulación 3 es la que más tiempo tarda en estabilizarse en la posición objetivo; su movimiento parece estar temporalmente desfasado, y el error aumenta cada vez que se invierte la dirección de rotación.

Los experimentos con la trayectoria cuadrada (Fig. 5(a)) indican que el robot es preciso, pero carece de exactitud a la hora de seguir la trayectoria de referencia (Fig. 5(b)(c)). Este comportamiento se debe a limitaciones tales como el desgaste mecánico de las articulaciones en particular la articulación 3 (véase la Fig. 4) y la latencia en la estabilización de la trayectoria. Una estrategia de mitigación consistió en aumentar el CTI y emitir comandos con un PTPD corto, minimizando así el error a costa de tiempos de ejecución más largos. Los resultados correspondientes para la trayectoria cuadrada se recogen en la Tabla IV.

En la prueba posterior realizada con la trayectoria de lemniscata (Fig. 5(d)), la diferencia en el RMSE entre la primera combinación (CTI = 1 s, PTPD = 26.7 mm) y la última (CTI = 0.250 s, PTPD = 6.6 mm) es de 2.45 mm, lo que indica un cambio moderado (Fig. 5(e)(f)). Sin embargo, existe una clara tendencia al aumento del error a medida que ambos parámetros se reducen simultáneamente. Este aumento del RMSE probablemente se deba principalmente a la latencia en la transmisión de comandos a velocidades más altas, a la dinámica de los actuadores, a los efectos de cuantificación introducidos por la representación basada en pulsos, o a una combinación de estos y otros factores.

Aunque el sistema responde rápidamente a los comandos, la principal limitación es la latencia del SPID a la hora de estabilizarse en las posiciones objetivo. La razón de ello es que los controladores SPID se basan en la modulación de frecuencia de pulso (PFM) para accionar los motores. La frecuencia de pulso disminuye a medida que se acerca a la

Cuadro III
RMSE (EN GRADOS) POR ARTICULACIÓN PARA DIFERENTES INCREMENTOS ANGULARES Y DIFERENTES CTI.

Articuación	CTI (s)	1°	2°	5°	8°	10°
1	1.000	0.548	0.831	1.209	1.297	1.325
2	1.000	0.616	0.837	1.062	1.058	0.984
3	1.000	1.249	1.649	2.285	2.619	2.651
4	1.000	0.229	0.255	0.215	0.277	0.318
1	0.500	0.867	1.269	2.340	2.461	3.321
2	0.500	1.164	1.582	2.069	3.277	3.079
3	0.500	1.972	2.586	3.767	5.434	6.223
4	0.500	0.328	0.351	0.268	0.316	0.351
1	0.250	1.742	2.926	5.789	7.857	9.281
2	0.250	1.951	2.892	5.698	9.420	10.130
3	0.250	3.147	4.478	7.667	10.771	11.776
4	0.250	0.440	0.468	0.934	1.884	3.545

Cuadro IV
RESULTADOS DEL RMSE PARA DIFERENTES VALORES DE CTI, n Y PTPD EN UNA TRAYECTORIA CUADRADA.

CTI (s)	PTPD	rmse-TCP	rmse-j1	rmse-j2	rmse-j3	rmse-j4
1.000	66.7 mm	28.64 mm	1.030°	1.336°	2.390°	0.570°
1.000	40.0 mm	22.61 mm	0.804°	0.990°	2.052°	0.652°
1.000	22.2 mm	18.57 mm	0.620°	0.795°	1.828°	0.652°
1.000	11.8 mm	13.17 mm	0.441°	0.623°	1.446°	0.630°
0.500	66.7 mm	49.20 mm	2.358°	2.284°	3.765°	0.591°
0.500	40.0 mm	38.72 mm	1.783°	1.730°	3.078°	0.636°
0.500	22.2 mm	28.76 mm	1.191°	1.239°	2.502°	0.693°
0.500	11.8 mm	21.53 mm	0.738°	0.908°	2.048°	0.679°
0.250	66.7 mm	85.42 mm	5.523°	3.530°	5.616°	0.825°
0.250	40.0 mm	63.22 mm	3.703°	2.639°	4.552°	0.702°
0.250	22.2 mm	45.83 mm	2.431°	2.104°	3.437°	0.747°
0.250	11.8 mm	33.35 mm	1.537°	1.479°	2.761°	0.745°

posición objetivo, lo que provoca una menor precisión de posicionamiento. De hecho, esta característica puede considerarse beneficiosa para los brazos robóticos colaborativos, ya que estos controladores hacen que el robot pierda algo de fuerza al alcanzar la posición objetivo. Su principal ventaja es que tiene un consumo energético estimado de 42 mW en la FPGA Zynq 7100 (solo para los controladores SPID); esto supone aproximadamente un tercio del consumo energético de un controlador PID digital convencional de rendimiento equivalente [12]. En comparación con el robot original, la diferencia es aún mayor, ya que el PID original se ejecutaba mediante una CPU Intel 80188, cuyo consumo de energía era del orden de 1.75 W.

IV. CONCLUSIONES

En este artículo, se realizaron pruebas P-ACC en la plataforma ED-Scorbot con controladores SPID. Las mediciones de las posiciones de las articulaciones se obtuvieron a partir de las lecturas proporcionadas por sus codificadores. Las métricas de exactitud y precisión se calcularon de acuerdo con la norma ISO 9283; sin embargo, el protocolo experimental se simplificó mediante el uso de trayectorias repetidas. Cada prueba se llevó a cabo utilizando parámetros específicos de PTPD y CTI, y se repitió cuatro veces con el fin de obtener la desviación media de la posición del TCP. En un primer momento, las pruebas se llevaron a cabo para analizar el comportamiento de cada articulación de forma individual, obteniéndose errores absolutos medios de 0.548° para J1, 0.616° para J2, 1.249° para J3 y 0.229° para J4, utilizando un CTI de 1 segundo e incrementos de 1°. Posteriormente, se ejecutaron trayectorias completas. A partir de estos experimentos, se obtuvieron valores mínimos de RMSE de 13.17 mm para una trayectoria cuadrada y de 14.19 mm para una trayectoria en lemniscata. Estos resultados muestran que el robot tiene una alta precisión, ya que durante las cuatro repeticiones, el TCP recorre consistentemente los mismos puntos, pero una exactitud limitada, debido principalmente a la naturaleza del controlador, que hace que el robot sea compatible con futuros robots colaborativos, y a las limitaciones de la articulación 3, ya que no alcanza las posiciones de referencia correctamente o las alcanza con retraso. Según el manual del Scorbot-ER-VII, con su sistema de control original basado en PID y en condiciones de funcionamiento óptimas, el manipulador alcanzó una repetibilidad posicional de ±0,2 mm [13] en la década de los 90. Aunque con la configuración neuromórfica actual el ED-Scorbot presenta mayores errores de precisión, las pruebas realizadas en este trabajo muestran que dicho error disminuye cuando se establece un intervalo de tiempo suficiente entre comandos consecutivos. Por otra parte, las pruebas de caracterización de las articulaciones han demostrado que estas pueden converger hacia las posiciones objetivo si se concede el tiempo adecuado para que el controlador estabilice la posición de la articulación (véase la Fig. 2). Sin embargo, como se ha indicado anteriormente, esta estrategia no es óptima en aplicaciones en las que el tiempo de ejecución de la tarea es una restricción crítica.

Hay varios aspectos del sistema que pueden optimizarse para lograr un rango de error aceptable, y este trabajo ayuda a identificar y contextualizar estos aspectos, como el ajuste de la articulación 3. Para validar y contrastar las mediciones internas, se requieren sensores externos, lo cual está previsto como una línea de trabajo futura.

El repositorio ED-Scorbot[3] contiene los scripts de Python y MATLAB® y el modelo URDF del robot utilizado en este trabajo.

AGRADECIMIENTOS

El Ministerio de Transformación Digital y Función Pública de España ha financiado parcialmente esta investigación mediante la subvención USECHIP (TSI-069100-2023-001) del programa PERTE ≪Cátedra Chip≫, financiado por la Unión Europea – Next Generation EU, por NEKOR (PID2023-149071NB-C54/AEI/10.13039/501100011033) y por la acción complementaria PAIDI 2020 (QUAL21 008 USE) de la Junta de Andalucía. D.E.T.H. cuenta con una beca del SENACYT para cursar un doctorado en la Universidad de Sevilla.

REFERENCIAS

[1] D. V. Christensen and et. al, "2022 roadmap on neuromorphic computing and engineering," *Neuromorphic Computing and Engineering*, vol. 2, p. 022501, 5 2022. [Online]. Available: https://iopscience.iop.org/article/10.1088/2634-4386/ac4a83https://iopscience.iop.org/article/10.1088/2634-4386/ac4a83/meta

[3] https://github.com/RTC-research-group/ED-Scorbot/ (Consultado el 10 de mayo de 2026)

[2] Y. Sandamirskaya, M. Kaboli, J. Conradt, and T. Celikel, "Neuromorphic computing hardware and neural architectures for robotics," *Science Robotics*, vol. 7, no. 67, p. eabl8419, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abl8419

[3] A. D. Hines, M. Milford, and T. Fischer, "A compact neuromorphic system for ultra–energy-efficient, on-device robot localization," *Science Robotics*, vol. 10, no. 103, p. eads3968, 2025. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.ads3968

[4] A. Linares-Barranco, F. Perez-Peña, A. Jimenez-Fernandez, and E. Chicca, "Ed-biorob: A neuromorphic robotic arm with fpga-based infrastructure for bio-inspired spiking motor controllers," *Frontiers in Neurorobotics*, vol. 14, p. 590163, 11 2020. [Online]. Available: www.frontiersin.org

[5] C. Frenkel and G. Indiveri, "Reckon: A 28nm sub-mm2 task-agnostic spiking recurrent neural network processor enabling on-chip learning over second-long timescales," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, Feb 2022, pp. 1–3.

[6] S. Canas-Moreno and et al., "Towards neuromorphic fpga-based infrastructures for a robotic arm," *Autonomous Robots*, vol. 47, no. 7, 2023.

[7] A. Barrientos, L. F. Peñin, C. Balaguer, and R. Aracil, *FUNDAMENTOS DE ROBÓTICA. 2.a Edición*. McGraw-Hill/Interamericana de de España, S.A.U., 2007.

[8] D. Trejos-Hernández, E. Cerezuela-Escudero, and A. Linares-Barranco, "On the simulation of a neuromorphic robotic arm with dynamic control for rapid trajectories evaluation," *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, vol. 603 LNICST, pp. 221–238, 2025. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-87345-4_15

[9] L. Lattanzi, C. Cristalli, D. Massa, S. Boria, P. Lépine, and M. Pellicciari, "Geometrical calibration of a 6-axis robotic arm for high accuracy manufacturing task," *The International Journal of Advanced Manufacturing Technology 2020 111:7*, vol. 111, pp. 1813–1829, 10 2020. [Online]. Available: https://link.springer.com/article/10.1007/s00170-020-06179-9

[10] E. Piñero-Fuentes, D. Curra-Sosa, D. Cascado, S. Canas, A. Rios-Navarro, F. Gomez-Rodriguez, A. Jimenez-Fernandez, and A. Linares-Barranco, "A hybrid neuromorphic robotic-arm lemniscate trajectories dataset for learning-based applications," in *2024 31st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Nov 2024, pp. 1–4.

[11] T. Ortner, H. Petschenig, A. Vasilopoulos, R. Renner, Špela Brglez, T. Limbacher, E. Piñero, A. Linares-Barranco, A. Pantazi, and R. Legenstein, "Rapid learning with phase-change memory-based in-memory computing through learning-to-learn," *Nature Communications 2025 16:1*, vol. 16, pp. 1243–, 2 2025. [Online]. Available: https://www.nature.com/articles/s41467-025-56345-4

[12] A. Jimenez-Fernandez, G. Jimenez-Moreno, A. Linares-Barranco, M. J. Dominguez-Morales, R. Paz-Vicente, and A. Civit-Balcells, "A neuro-inspired spike-based pid motor controller for multi-motor robots with low cost fpgas," *Sensors*, vol. 12, no. 4, pp. 3831–3856, 2012. [Online]. Available: https://www.mdpi.com/1424-8220/12/4/3831

[13] Eshed Robotec (1982) Ltd., *SCORBOT-ER VII User's Manual*, 2nd ed., Eshed Robotec (1982) Ltd., December 1998, iSBN: 965-420-058-2. Catalog # 100017 Rev. C.

Arquitecturas heterogéneas empotradas

FPGA-based Energy-efficient Decision Tree Ensemble Accelerator

Unai Fernandez-Urroz[1], Libe Mori[1], Unai Garciarena[1], Javier Navaridas[1], Jose A. Pascual[1]

Abstract— **The growing complexity of artificial intelligence techniques and the increasing size of modern datasets have led to a substantial rise in energy consumption in conventional implementations optimized for CPUs and GPUs. Field Programmable Gate Arrays (FPGA) represent a promising alternative, as they enable customized hardware acceleration that improves both latency and performance while maintaining high energy efficiency. This paper presents an FPGA-based architecture for accelerating tree ensemble inference, with emphasis on energy efficiency and resource optimization. The proposed system integrates a Python-based training workflow that generates a flexible configuration file, enabling adaptable deployment across varying numbers of trees and dataset characteristics. The architecture exploits fine-grained parallelism by executing decision trees concurrently and aggregating their predictions through a hardware-based majority voting scheme. Experimental evaluation on a Zynq-based platform demonstrates low resource utilization, requiring less than 4% flip-flops and 16% LUTs, while BRAM becomes the main limiting factor, reaching 70% for a 64-tree configuration. The total power dissipation is approximately 1.68 W, of which only 0.28 W corresponds to the programmable logic implementing the accelerator. Additionally, we found competitive inference latency compared to related FPGA implementations. These results confirm the suitability of the design for energy-constrained embedded AI applications.**

Keywords— **FPGA, Hardware accelerators, Machine Learning, Random Forest, Low-power, limited resources**

I. Introduction

ARTIFICIAL Itelligence (AI), particularly Machine Learning (ML), has experienced rapid growth in recent years, with increasing adoption across domains such as industry, healthcare, and education. ML refers to systems that learn from data to identify patterns and perform tasks without explicit programming. This expansion has been driven by advances in algorithms, availability of large datasets, and improvements in computing hardware. Over the past decades, multiple algorithms have been developed, with varying levels of complexity and applicability [1], including Decision Trees (DT), Random Forests (RF), Gradient Boosted Decision Trees (GBDT), Support Vector Machines (SVM), and Deep Neural Networks (DNN).

Recent advances in hardware [2][3] have facilitated the emergence of novel architectural designs that significantly improve the performance of these ML algorithms. CPUs now feature specialized cores for AI, designed to improve performance while minimizing energy consumption, while GPUs also include specialized hardware designed for AI.

[1]Dept. of Computer Architecture and Technology, University of the Basque Country (EHU), e-mail: unai.fernandezu@ehu.eus

Despite ongoing progress in the field, the increasing complexity of machine learning algorithms, together with the use of larger datasets, raises the computational requirements needed to achieve performance improvements. However, this trend toward increasingly powerful hardware and intensive computation has led to a significant rise in energy consumption. In this context, Field Programmable Gate Arrays (FPGAs) emerge as a viable alternative, offering customizable hardware architectures that can reduce both energy consumption and latency. An FPGA allows the development of custom digital systems used in embedded computer systems and prototyping [4][5].

This paper presents the design and implementation of an FPGA-based hardware accelerator for Random Forest inference, Ranger. A Random Forest [6] is a widely used machine learning algorithm that combines the predictions of multiple decision trees to produce a final output, improving robustness and generalization performance.

The Random Forest model is trained offline, and the resulting parameters are converted into a configuration file that encodes the structure of the decision trees. The proposed architecture consists of two main components: a Processing System (PS) and Programmable Logic (PL). The PS, which includes an ARM processor running PetaLinux [7], is responsible for system control and for transmitting configuration and feature data to the accelerator.

The PL implements the proposed configurable and parallel tree-ensemble accelerator called Ranger. By exploiting the inherent parallelism of tree-based models, the architecture enables concurrent execution of multiple decision trees, followed by hardware-based aggregation of partial predictions. The configuration and feature data are transferred from the PS to the PL via the AXI-Stream bus protocol [8], ensuring efficient high-speed communication between both domains and allowing the accelerator to adapt to varying numbers of trees and dataset characteristics.

Our experiments conducted show a low resource utilization for deploying the architecture with 64 trees, using only 16% of available LUTs and 3% of FFs on the Zynq-7020 used for evaluation. Compared to other similar proposals, the results on resource usage demonstrate an improvement, despite being limited by the number of available BRAMs to store all the node data. Additionally, our system exhibits a total power consumption of 1.68 W, but Ranger dissipates approximately 0.28 W. Furthermore, we found our proposal achieves competi-

Dataset:

x0	x1	x2	class
0.11	1.12	3.20	1
2.44	2.55	1.33	0
3.31	1.23	1.62	1

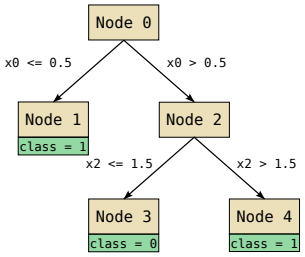


Fig. 1: Example dataset with an induced decision tree.

tive average inference times even when featuring a smaller fabric with lower frequency.

The rest of the paper is organized as follows. Section II reviews the fundamentals of decision trees and random forests and discusses related hardware accelerators for machine learning. Section III presents the design and implementation of the proposed system. Section IV details the experimental setup and evaluation results. Finally, Section V concludes the paper and outlines directions for future work.

II. Background

A. Decision trees and ensemble methods

Decision Trees (DTs) [9] are widely used supervised learning models that represent decision boundaries through a hierarchical structure of internal decision nodes and terminal leaf nodes. Internal nodes take decisions based on a given attribute, whereas leaf nodes correspond to output classes or regression values. The example DT shown in Fig. 1 consists of one root node (node 0), one internal decision node (node 2) and three leaf nodes (nodes 1, 3 and 4). The root node and decision nodes evaluate feature-based conditions to determine the traversal path by selecting the appropriate branch, while the leaf nodes store the output classes to be predicted.

As tree depth and structural complexity increase, decision trees become more expressive but also more prone to overfitting. This is a pathological condition where the model may adapt excessively to the training data, including noise, resulting in reduced generalization capability and degraded performance on unseen samples.

To mitigate this limitation, ensemble methods combine multiple decision trees to improve robustness and predictive performance. Two widely adopted ensemble strategies are bagging and boosting. Random Forest [6] is based on bagging, where multiple trees are trained independently on randomized subsets of the data, and their predictions are aggregated typically through majority voting. In contrast, Gradient Boosted Decision Trees (GBDT) build trees sequentially, with each new tree trained to correct the errors of the previous ones. XGBoost [10] is a highly optimized and regularized implementation of the GBDT framework. By combining the outputs of multiple trees, ensemble models reduce variance and improve stability, as individual trees can compensate for one another's weaknesses.

Our proposed architecture targets energy-constrained embedded systems and focuses on the efficient execution of decision tree ensembles with independently trained trees, such as Random Forest or GBDT. By exploiting their inherent parallelism, the design enables concurrent tree execution.

B. Hardware accelerators for decision tree ensembles

The increasing adoption of tree-ensemble methods in real-time and embedded applications has motivated the development of specialized hardware accelerators. In particular, FPGA-based implementations have attracted significant attention due to their ability to exploit parallelism while maintaining low-power consumption.

Several works have focused on accelerating gradient-boosted tree models (GBDT). Olmos et al. [3] present an optimized FPGA architecture for XGBoost inference, reporting substantial speed-ups compared to CPU and GPU implementations. Their design emphasizes high throughput on performance-oriented FPGA platforms. This is considered one of the baselines in our experimental work, as it features a similar architecture as Ranger.

Similarly, Kanani et al. [4] introduce LightFPGA, an automated framework that generates *ad-hoc* Verilog RTL from pre-trained LightGBM models [11]. Their approach constructs the whole trees within the FPGA logic using LUTs and registers, which may limit scalability when the number of trees or their depth is too high. Furthermore, this approach severely limits clock frequency (a mere 25 MHz in a high-performance Alveo U280 FPGA, which can run at several hundred MHz). Even with these limitations, they attained significant acceleration and lower power consumption when compared with an Intel i5 CPU running at 2.5 GHz. This serves as our second baseline, as it enables a comparison between our general model and a system that generates *ad hoc* trees tailored to specific cases.

Owaida et al. [5] proposed a flexible CPU–FPGA co-design that optimizes on-chip memory usage to support large tree ensembles. Their architecture allows processing to be distributed between the CPU and FPGA when the model size exceeds available hardware resources. The FPGA model assumes a full binary tree with no missing nodes to simplify memory allocation, which may result in a substantial amount of wasted BRAM storage when unbalanced trees are employed. In order to avoid this waste, our model only stores information for the necessary nodes.

More recently, Dinh et al. [2] presented a resource-efficient Random Forest accelerator targeting edge-computing and IoT scenarios. They introduced an architecture based on multiple Decision Tree Processors (DTP) designed for pipelined tree processing. Their design achieves improved inference speed and lower energy consumption compared to general-purpose processors, highlighting the suitability of FPGA-based solutions for low-power environments.

While existing approaches often prioritize maximum throughput or large-scale FPGA platforms, fewer works explicitly emphasize configurability and energy-efficient deployment on resource-constrained embedded systems. In this context, the architecture proposed in this paper focuses on adaptable Random Forest inference with low-power consumption, targeting embedded Zynq-based platforms.

III. Design and implementation of the hardware accelerator

As mentioned previously, the proposed system consists of two primary components, the Programmable Logic (PL) and the Processing System (PS). The PL implements the Ranger IP block, which features an architecture designed to accelerate the inference of tree ensembles, capable of adjusting internal processing modules to handle different numbers of decision trees in the ensemble. Ranger can handle different datasets, as it is configured with a sequence of node data. The PS runs a Python application on Petalinux to train the model and test it with Ranger. To configure the accelerator, the application in the PS creates a configuration file. Note, however, that Ranger could work independently of the PS by connecting the input and output FIFOs to other protocols.

Data transmission between the PS and PL is carried out through the AXI-Stream bus protocol. AXI-Stream [8] is a point-to-point protocol of the ARM AMBA family. Using this protocol allows high-performance communication between the different processing modules in the FPGA. AXI-Stream is a burst-based protocol, which makes it suitable for sending a large amount of data between the components. FIFO buffers are used to store incoming data and then forward the output, thus mitigating synchronization issues between the PS and the PL.

A. Architecture overview

This section provides a general overview of the main system proposed in this paper [1], detailing the data flow that starts with the input FIFO, passes through the main module called Ranger, and concludes with the output FIFO containing the inference results. Fig. 2 illustrates the mentioned components and also highlights the internal components of Ranger, whose functions are detailed below.

- **Ranger:** It is the main module that contains the necessary submodules to perform the inference, such as `DT_accel`, `configurator`, and `aggregator`. This module orchestrates the configuration, execution, and aggregation phases of the inference process. At the beginning of the process, it handles the configuration phase to adjust each tree accelerator properly. Then, each DT accelerator computes a partial result. Finally, this module performs the aggregation phase, which consists of getting a final prediction based on previously computed partial results.
- **Data FIFOs:** The FIFOs located at the input and output temporarily store data for communication with the PS. The data coming from the PS, which includes information for configuring the trees and features, is stored in the input FIFO. Once the computation of results is complete, the final results are stored in the output FIFO before being sent to the PS.
- **Configurator:** This submodule inside Ranger performs the configuration of the DT accelerators by sending 32-bit values containing essential information to reproduce the structure of trees to the internal memories of each tree.
- **DT_accel:** It is the submodule inside Ranger in charge of accelerating a single decision tree. To create the tree ensemble, this element is instantiated multiple times depending on the number of trees in the Random Forest.
- **Aggregator:** The aggregator computes the final result by performing a majority vote between the partial results of the DT accelerators.

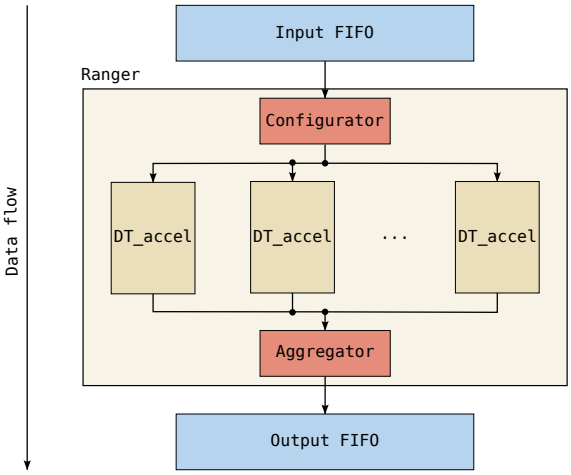


Fig. 2: General view of proposed system and internal components of Ranger.

B. Data format and configuration

As a first step, the Random Forest model is trained in the PS using a Python script with the *scikit-learn* library [12]. After training the model with the desired dataset, a configuration file is created indicating the number of trees in the ensemble and the depth of each one. This configuration file consists of a sequence of values split into two parts: the first part contains representative characteristics of each node and the second part contains the features.

Each of the trees is traversed using a method known as pre-order traversal. This technique involves visiting the nodes of the tree in a specific order: starting from the root, it first processes the left subtree, then the right subtree. On the right side of Figure 1, an example of a DT can be seen, where

[1]Code available in: `https://github.com/UnaiFernandez/Ranger-Decision-Tree-Accelerator.git`

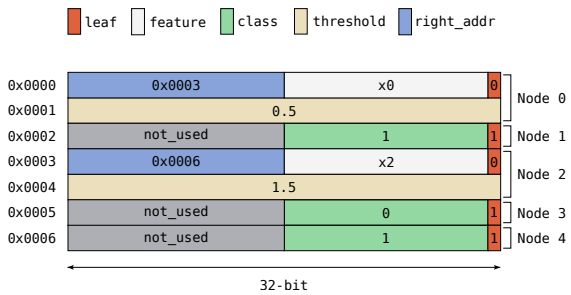


Fig. 3: Format of data stream for configuring the DT.

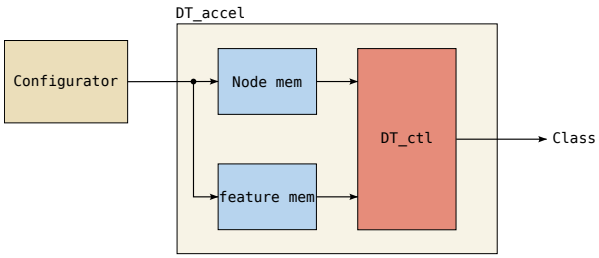


Fig. 4: Overview of the internal components of DT_accel module.

the numbering of the nodes corresponds to the order of traversal. During tree traversal, data from each node (leaf, feature, threshold, and class) is extracted and used to reconstruct the tree structure in the PL. This data is added to a sequence of 32-bit values displayed on Figure 3. As shown, decision nodes consist of two 32-bit words: the first containing the leaf, feature, and right address, and the second containing the threshold. Leaf nodes, on the other hand, contain the class value. The different values extracted from the decision trees are detailed below.

- **leaf**: The least significant bit indicates that the current node is a leaf node (1) if the value equals '1' and a decision node (0) if it equals '0'.
- **feature / class**: Bits 1 to 15 in the first word show the feature to be selected if the current node is a decision node or the class if the node is a leaf.
- **right addr**: Bits 16 to 31 in the first word represent the address of the right child node. This data is necessary to jump to the position where the right node is located in the memory. The left child address does not need to be stored, as it is located in the next position in memory.
- **threshold**: If the node is a decision node, the next word in the memory corresponds to the threshold, represented by using the IEEE 754 single-precision floating-point format [13] (1 sign bit, 8 exponent bits, and 23 mantissa bits).

As described in section III-A, the *configurator* is the component in charge of managing the configuration and feature data. At the beginning of the configuration data sequence for each tree, the number of nodes is specified to allow the *configurator* to differentiate the data for each decision tree, as all configuration data is sent to the PL in a single sequence. The configuration data is followed by the feature data, represented with the IEEE 754 standard for floating-point arithmetic [13]. All this data is managed by the *configurator* and sent to the corresponding DT accelerator.

C. Decision tree acceleration module

The *DT_accel* module in Fig. 4 is responsible for executing the DT algorithm once the configuration process has been completed. Leveraging the design capabilities and features of an FPGA, the *DT_accel* module can be instantiated multiple times to execute multiple decision trees in parallel. As illustrated in Fig. 4, each *DT_accel* module includes two internal memories that receive the data from the *configurator* and a controller for the algorithm. The internal *Node mem* stores the data related to the structure of the tree, whereas the *Feature mem* stores the features of the current sample, which are required to compute the inference. *DT_ctl* is the module responsible for managing the entire inference process.

During the execution process, the node memory data is the first to be stored. Once the *node_mem* in each DT accelerator is filled with the configuration data, the necessary features for execution are sent to the *feature_mem*. At this point, the DT controller module (*DT_ctl*) comes into play, reading data from both memories and progressing in the search for a leaf node, which specifies the corresponding class. As seen in Algorithm 1, the process begins when the enable signal is activated; otherwise, the process remains idle. When the algorithm exits the idle state, the information of the current node is collected from the node memory, which is stored as detailed in Fig. 3. If the selected node is a decision node, the selected feature value is compared to the threshold of the node to determine the address of the next node to process. In the case that the feature is less than or equal to the threshold, the address of the next node will be the address of the left child node (the next node stored in the Node memory). However, when the feature is greater, the address of the right child node will be selected by reading it from the current node. In contrast, if the selected node is a leaf node, the search process terminates, and the bits 1 to 15 of the memory encode the predicted class using.

D. Class aggregation module

Random Forest algorithm aggregates the partial results obtained from each DT in the ensemble to achieve more robust and higher-quality predictions. To determine the final result of this solution, a majority vote is conducted among all trees to identify the result that appears most frequently in the partial predictions.

Fig. 5 illustrates the detailed process to compute the final prediction, starting from the partial results of each tree. Initially, the results from each DT accelerator undergo one-hot encoding. This representation allows each class to be delineated by activating

Algorithm 1 DT prediction algorithm.

Require: *node_data*, *feature_data*, *enable*
  assert(enable = '1')
  while *done* =′ 0′ **do**
    Collect node data
    if *is leaf* **then**
      Get class value from memory and activate *done* signal
    else
      Select the next decision node
      if *feature* <= *threshold* **then**
        Select left child node
      else
        Select right child node
      end if
    end if
  end while
Ensure: *Class*, *done*

the bit at the position corresponding to the predicted class while keeping the other bits set to '0', which simplifies the process of calculating the final class. In Fig. 5, class 0 is represented as '0001' and class 1 as '0010'. Each one-hot encoded result is stored in a matrix called *class_mat*, positioned according to the DT identifier. In the *class_mat* matrix, columns represent the different possible classes, while rows indicate the results from each tree. To continue the search for the final class, the values in the columns are summed and stored in *sum_mat*. This vector contains the number of times each class has appeared in the partial results. To determine the final class, it is necessary to identify the index of the largest element within *sum_mat*. In the example illustrated in Fig. 5, the final class is determined to be 0, as it has occurred three times.

The aggregation process requires a computation time that is directly related to the number of trees in the ensemble. This factor is a limitation at the time of choosing a higher clock frequency. Therefore, this process is divided into several steps that are executed over multiple clock cycles.

IV. Evaluation of the designed architecture

We proceed now to evaluate the design of Ranger, putting it into context against other state-of-the-art designs. We first explain our experimental setup and then analyze the area and power utilization as well as the inference speed.

A. Experimental set-up

We used the Zybo-Z7 [14] development board to evaluate the design of Ranger for accelerating Random Forest inference. This development board features a Zynq-7020 SoC, which integrates an ARM Cortex-A9 processor with Xilinx 7-series FPGA logic. This architecture makes this development board suitable for embedded applications. In this project, the PS and the PL of the Zynq-7020 SoC

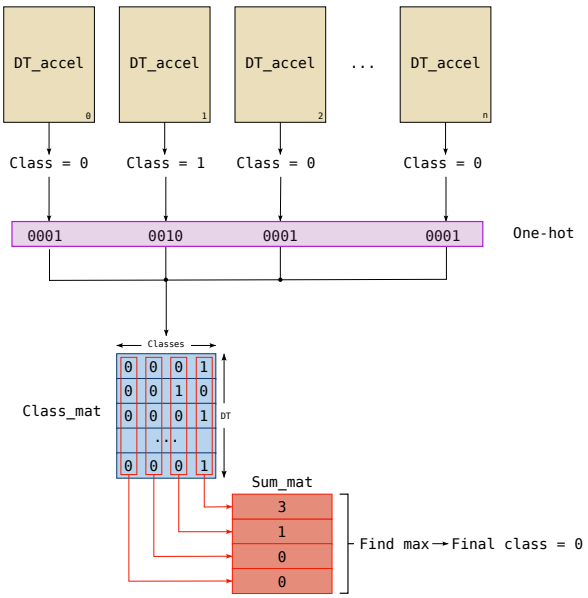


Fig. 5: Aggregation of the partial results to compute the final prediction.

are utilized. The ARM processor runs PetaLinux [7], and transmits the configuration data via the AXI-Stream bus to the PL, where the architecture described in Section III performs Random Forest inference.

To compare Ranger with the state-of-the-art (Section II), we selected two recently proposed architectures with similar objectives: LightFPGA [4] and the optimized XGBoost accelerator proposed by Olmos et al. [3]. Unfortunately, they use different datasets in their experiments, so they are not comparable with each other. For the sake of simplicity, we considered the five datasets in Table I our experimentation, as they are used in similar proposals:

Table I: Details of the datasets selected for the experimentation.

	Dataset	Features	Instances	Classes
[15]	Iris	4	150	3
[16]	Wine	13	178	3
[17]	Bank note	5	1372	2
[18]	Heart disease	13	303	2
[19]	Diabetes	8	768	2

To evaluate Ranger, the configuration file for the Random Forest has been created using the previously mentioned Python application. We trained Random Forests consisting of 16, 32, and 64 decision trees, each with a depth of 8. The training process used 80% of the dataset, leaving the remaining 20% as testing data for cross-validation.

To run the experiments, we developed a multi-threaded C application that is in charge of interfacing PS and the PL via the AXI-Stream bus. First, the application sends the DT configuration data to initialize the Random Forest model. Once this is done, it streams the testing data for inference and receives the results computed in the PL. Once the last result is computed, a cycle counter is stopped. This way, it can calculate the average inference time based on

Table V: Average inference time comparation with state-of-the-art proposals.

		Iris	Wine	Bank note	Heart disease	Diabetes
	Ranger (Zybo-Z7)	252 ns	386 ns	348 ns	433 ns	410 ns
[4]	**Alveo U280**	320 ns	320 ns	320 ns	-	-
	Intel i5 2.4 GHz	33 μs	32 μs	21 μs	-	-
[3]	**XC7K325T**	-	-	-	394 ns	205 ns
	Alveo 250	-	-	-	390 ns	200 ns
	Intel i7 8700k	-	-	-	23 μs	14 μs
	Nvidia RTX 4060	-	-	-	27 μs	25 μs

the total elapsed time and the number of inferences carried out.

B. Analysis of the results

In Table II, the available resources on the **Zybo-Z7** are displayed alongside the resources utilized with 16, 32, and 64 DTs, provided by Xilinx Vivado. Regarding LUTs and FFs, it can be seen that a small amount is used compared to the number available on the Zynq-7020. Additionally, the use of BRAM increases with the number of trees. This indicates that the scalability of the design is highly sensitive to the amount of available BRAM on the FPGA.

Table II: Resource utilization in the Zynq-7020.

Resources	Available	16 trees	32 trees	64 trees
LUTs	53200	2995 (6%)	4885 (9%)	8502 (16%)
FFs	106400	1897 (2%)	2469 (2%)	3608 (3%)
BRAM	140	26.5 (19%)	50.5 (36%)	98.5 (70%)

On the other hand, Table III presents a comparative overview of resource usage across various proposals for tree ensembles composed of 64 trees (100 in the case of LightFPGA [4]). Despite the differing architectures in the FPGAs for each proposal, it is evident that the resource usage of Ranger is generally lower than the other proposals. In the case of LightFPGA, the usage of LUTs and FFs varies greatly depending on the dataset because trees are constructed *ad hoc* for each case. In the table we present the reported results with Wine and Bank note datasets. Specifically, with the Wine dataset, resource utilization is lower than Ranger's, whereas with the Banknote dataset, the usage increases significantly.

Table III: Comparative overview of resource utilization.

	Olmos et al. [3]	**LightFPGA**[4]		**Ranger**
Target	XC7K325T	Alveo U280		Zybo-Z7
DT number	64	100		64
Clock	125 MHz	25 MHz		100 MHz
LUTs	44262	26334	7300	8502
FFs	29253	7300	8082	3608
BRAM	256	0	0	98.5

To estimate Ranger's power dissipation, we used the Power Report tool provided by Vivado [20]. This tool estimates watts based on the simulation of the implementation to ensure that the switching activity is reflected in the results. Table IV presents the findings for configurations with 16, 32, and 64 DTs. For the sake of comparison, we also show the total dissipation reported by LightFPGA[2]. As can be seen, Ranger achieves significantly lower power consumption than LightFPGA, as it does not require constructing the whole trees within the FPGA. Furthermore, we can see that static and dynamic power remain quite stable despite the increase in the number of trees. It is important to note that using a stand-alone FPGA to process data from local sensors would further improve power consumption, as the PL consumption is roughly one order of magnitude lower than that of the PS, which is only used to configure Ranger and feed it with data.

Table IV: Power dissipation breakdown on the Zybo-z7.

Power	16 trees	32 trees	64 trees	LightFPGA[4]
PS	1.400 W	1.400 W	1.400 W	-
PL–Static	0.253 W	0.255 W	0.261 W	-
PL–Dynamic	0.011 W	0.018 W	0.032 W	-
TOTAL	1.664 W	1.673 W	1.693 W	3.205-3.711 W

Table V provides a comparison of the average inference time between Ranger and the state-of-the-art proposals. Olmos et al. [3] used XC7K325T and Alveo 250 devices, whereas LightFPGA [4] used an Alveo U280. The AMD Alveo boards feature a 16 nm Ultrascale architecture designed for data centers. Meanwhile, the XC7K325T and the Zybo-Z7 used in this approach are less powerful and intended for developing embedded applications. For Ranger and the proposed architecture by Olmos, the clock frequencies are similar, at 100 MHz and 125 MHz, respectively. The number of processed trees is also different; Ranger uses 64, the XC7K325T uses 128, the Alveo 250 512 and the Alveo U280 uses 100. LightFPGA [4] works at a much lower frequency, 25 MHz. As shown in Table V, the results obtained with the Heart disease dataset are similar to those achieved with the Alveo 250 and XC7K325T. However, in the case of the Diabetes dataset, a greater difference in results is observed. This is attributed to the fact that as the number of nodes in the tree increases, Ranger requires more memory accesses because deeper trees increase memory access latency due to additional node fetches.

V. Conclusions and Future work

This paper presents an FPGA-based architecture for acceleration tree ensemble inference on embedded platforms. The proposed design supports up to 64 decision trees operating at 100 MHz,

[2]We do not show power results for Olmos et al. because they did not report them.

achieving competitive average inference latency compared to related FPGA implementations. Through a configuration-driven approach, the architecture can adapt to different datasets and ensemble sizes without requiring structural modifications, providing flexibility for diverse deployment scenarios.

Experimental results on the Zybo-Z7 platform demonstrate low resource utilization (16% LUTs and 3% FFs for 64 trees), with BRAM usage becoming the limiting factor (70%). The total power consumption is approximately 1.68 W, of which only about 0.28 W corresponds to the programmable logic. These characteristics make the proposed solution particularly suitable for energy-constrained environments where limited hardware resources and low-power operation are critical requirements.

Despite these advantages, the scalability of the design is constrained by the available BRAM resources, as tree structures are stored in on-chip memory. Future work will focus on improving memory management strategies to reduce BRAM dependency and enable support for larger ensembles. Additionally, the aggregation module currently limits the achievable clock frequency due to timing constraints. Redesigning this component to increase parallelism and reduce the critical path length could further improve inference latency and overall system performance.

Acknowledgements

This research is supported by the Basque Government through projects KK-2025/00098 and IT1504-22. It is also supported by grant CNS2023-144315 funded by the MICIU/AEI/10.13039/501100011033 and by "European Union NextGenerationEU/ PRTR". It is also supported by grant PID2023-152390NB-I00 funded by the MICIU/AEI/10.13039/501100011033 and by "FEDER funds". Dr. Javier Navaridas holds a Ramón y Cajal fellowship funded by the Spanish Ministry of Science, Innovation and Universities (RYC2018-024829-I), funded by MCIN/ AEI/10.13039/501100011033 and, as appropriate, by "ESF Investing in your future" or by "European Union NextGenerationEU/PRTR".

References

[1] Iqbal H Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN computer science*, vol. 2, no. 3, pp. 1–21, 2021.

[2] Trung Pham Dinh, Cuong Pham-Quoc, Tran Ngoc Thinh, Binh Kieu Do Nguyen, and Pham Cong Kha, "A flexible and efficient fpga-based random forest architecture for iot applications," *Internet of Things*, vol. 22, pp. 100813, 2023.

[3] Rodrigo Olmos and Andrés Otero, "Optimized xgboost architecture on fpga for high-performance and seamless deployment," in *presented at the 7th Workshop on Accelerated Machine Learning (AccML)*, 2025.

[4] Alish Kanani, Swar Vaidya, and Harshit Agarwal, "Lightfpga: Scalable and automated fpga acceleration of lightgbm for machine learning applications," in *2021 25th International Symposium on VLSI Design and Test (VDAT)*. IEEE, 2021, pp. 1–6.

[5] Muhsen Owaida, Hantian Zhang, Ce Zhang, and Gustavo Alonso, "Scalable inference of decision tree ensembles: Flexible design for cpu-fpga platforms," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.

[6] Leo Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[7] Inc. Advanced Micro Devices, *PetaLinux Tools Documentation: Reference Guide (UG1144)*.

[8] ARM, *AMBA AXI-Stream Protocol Specification*, 2021.

[9] J. Ross Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[10] "Xgboost documentation," 2025.

[11] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[12] "Scikit-learn, machine learning in python," 2025, Accessed: 2025-07.

[13] "Ieee standard for floating-point arithmetic - redline," *IEEE Std 754-2008 (Revision of IEEE Std 754-1985) - Redline*, pp. 1–82, 2008.

[14] Diligent Inc., *Zybo Z7 Board Reference Manual*.

[15] Antony Unwin and Kim Kleinman, "The iris data set: In search of the source of virginica," *Significance*, vol. 18, 2021.

[16] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decision support systems*, vol. 47, no. 4, pp. 547–553, 2009.

[17] Volker Lohweg, Jan Leif Hoffmann, Helene Dörksen, Roland Hildebrand, Eugen Gillich, Jürg Hofmann, and Johannes Schaede, "Banknote authentication with mobile devices," in *Media Watermarking, Security, and Forensics 2013*. SPIE, 2013, vol. 8665, pp. 47–60.

[18] Robert C. Detrano, András Jánosi, Walter Steinbrunn, Matthias Emil Pfisterer, Johann-Jakob Schmid, Sarbjit Sandhu, Kern H Guppy, Stella Lee, and Victor Froelicher, "International application of a new probability algorithm for the diagnosis of coronary artery disease.," *The American journal of cardiology*, vol. 64 5, pp. 304–10, 1989.

[19] Jack W Smith, James E Everhart, William C Dickson, William C Knowler, and Robert Scott Johannes, "Using the adap learning algorithm to forecast the onset of diabetes mellitus," in *Proceedings of the annual symposium on computer application in medical care*, 1988, p. 261.

[20] Inc. Advanced Micro Devices, *Vivado Design Suite User Guide: Power Analysis and Optimization (UG907)*.

SNNorlax: Accelerating the training and inference of Spiking Neural Networks on FPGA

A Esteban-Sauce[1], L Mori[1], JA Pascual[1] and J Navaridas[1],

Resumen— **Spiking neural networks are a type of neural network that closely mimic the dynamics of biological brains by encoding information as discrete spikes. Unlike traditional artificial neural networks, this event-based representation reduces unnecessary computation and improves energy efficiency, particularly when neural activity is sparse. This work proposes a hardware acceleration platform designed as a neuromorphic accelerator for embedded IoT systems. The proposed low-power FPGA-based architecture supports arbitrary spiking neural network topologies and enables both inference and on-chip learning through spike-timing-dependent plasticity implemented directly in the FPGA fabric. Experimental results show that, although the maximum number of supported synapses is constrained by the available on-chip memory, the design achieves high resource efficiency and low power consumption. In particular, the accelerator supports up to 4,096 neurons and 131,072 synapses while operating within a power envelope of only 0.667 W, making it well suited for resource-constrained embedded neuromorphic applications.**

Palabras clave— **Spiking Neural Network, FPGA, Processor, Accelerator, Neuromorphic, STDP**

I. Introduction

In recent years, Artificial Neural Networks (ANNs) have proven highly effective in addressing a variety of tasks, including image classification [1], object recognition [2], and natural language processing [3]. Traditional ANNs are composed of layers of interconnected neurons that use continuous activation functions. As a result, they require numerous computationally intensive operations to be executed periodically, regardless of the actual activity of the network. This leads to high energy consumption and limits their efficiency, especially for their use in embedded systems.

Although ANNs are inspired by the way the brain works, their operation does not realistically resemble the actual way biological neurons work. In contrast, neuromorphic systems aim to replicate the biological behaviour more closely. One of the most important neuromorphic models is the Spiking Neural Network (SNN). In SNNs, information is transmitted using discrete spikes instead of continuous values. These spikes encode information based on their spatio-temporal properties and are sparsely distributed in both spatial and time dimensions [4]. Neurons operate asynchronously and only need to perform computations when spikes occur. Because of this event-driven behaviour, the energy consumption of SNNs correlates with the activity of the network, which can significantly reduce power usage.

[1]Dept. of Computer Architecture and Technology (CAT), Universidad del País Vasco (UPV/EHU), e-mail: asier.esteban@ehu.eus, javier.navaridas@ehu.eus, joseantonio.pascual@ehu.eus and libe.mori@ehu.eus

An SNN is composed of neurons connected through synapses. Neurons receive spikes from other neurons and integrate them into their membrane potential. When this potential exceeds a given threshold, the neuron fires a spike and enters a refractory period during which it cannot respond to new inputs. The operation of the spiking neurons modelled in SNNs is illustrated in Fig. 1. Several mathematical models have been proposed to mimic this behaviour, such as Leaky-Integrate-and-Fire (LIF) [5], Izhikevich (IZH) [6] or Hodgkin Huxley (HH) [7]. While more complex models such as HH provide higher biological accuracy, they also require more computation. In this work, we use the LIF model, which provides a good balance between computation and biological plausibility and, thus, is one of the most commonly used in the literature [8], [9], [10], [11], [12], [13]. Nonetheless, the proposed simulation framework is flexible and would allow the implementation of other neuron models if required.

Learning in SNNs is achieved by modifying the sensitivity of the neuron's synapses based on neural activity, in a process known as neural plasticity [14], [4]. One of the most widely used learning mechanisms is Spike Timing-Dependent Plasticity (STDP) [15]. STDP updates synaptic strength according to the relative timing of presynaptic and postsynaptic spikes. If a presynaptic neuron fires shortly before the postsynaptic spike, the synapse is strengthened. On the contrary, if it occurs after, the synapse is weakened. The intuition behind this way of operating is to reinforce correlated firings of neurons and weaken non-correlated firings, following the Hebbian theory of learning [16]. The STDP rule enables online and unsupervised learning in SNNs, which are the main types of learning typically considered within an IoT context [17], [18].

Most current implementations of SNNs are being developed targeting conventional computing platforms, such as CPUs[13] or GPUs[10]. Although these architectures can process neurons in parallel, they do not fully exploit the event-driven nature of SNNs, where computation should depend on the activity of the network. For this reason, both industrial and academic research groups have focused on developing specialized hardware accelerators for SNN execution [13], [19], [20]. In particular, Field-Programmable Gate Arrays (FPGAs) have attracted increasing interest as a platform to simulate SNNs. FPGAs are flexible and reconfigurable devices that can support a wide range of digital systems, including general-purpose processors [21] and application-specific processors [22]. While several FPGA-based

SNN accelerators have been proposed [11], [23], [24], [12], many of them are not very well suited for low-power IoT and embedded devices due to high power consumption, limited scalability, or exceedingly restrictive network models.

In this work, we present a fully event-driven FPGA-based neuromorphic accelerator that addresses these limitations. The proposed design supports arbitrary SNN topologies, implements on-chip STDP learning, and is optimized for low power consumption. The main goal of the accelerator is to be integrated into an embedded IoT platform capable of performing both training and inference using data received from sensors and making decisions in real time. In this context, the accelerator is designed to coexist with other hardware IP blocks responsible for input preprocessing and response control based on the outputs of the SNN. The architecture is composed of a custom synaptic memory, multiple buffers to store spikes, a Neuron Processing Unit (NPU) that updates neuron states and generates spikes, and a spike dispatcher that schedules neuron updates based on the buffered spikes and the network topology.

The proposed accelerator can support networks of up to 4096 neurons and 131,072 synapses, limited by the available on-chip BRAM resources of the FPGA. Experimental results in a Zynq-7000 SoC show that the design achieves high efficiency with an estimated power consumption of 0.268 W, making it suitable for embedded and resource-constrained neuromorphic applications, particularly in IoT systems.

The rest of the paper is organised as follows. Section II provides background on the neuron model and STDP learning. Section III describes the architecture and implementation of the proposed accelerator. Section IV presents the experimental results, including resource utilization and power consumption and compares the results with some of the state of the art accelerators. Finally, Section V concludes the paper and discusses possible lines of future work.

II. Background

In this section, we discuss background knowledge required for this paper. First, we will explain how the LIF neuron model works and then, describe how STDP enables unsupervised learning in SNNs.

A. Leaky-Integrate-and-Fire Neuron model

LIF [5] is a simplified biological neuron model that is widely used in neuromorphic research [8], [9], [10], [11], [12], [13]. This neuron model represents a good balance between computational complexity and biological realism. The LIF membrane potential can be described as follows:

$$C_m \frac{dV}{dt} = g_l(V_{rest} - V) + I \tag{1}$$

where C_m is the membrane capacitance, g_l is the membrane conductance, V is the neuron's membrane potential, V_{rest} is the resting potential value and I is the input current.

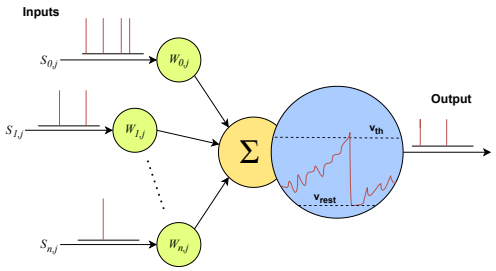


Fig. 1: Spiking neuron model employed in an SNN, showing spikes as synaptic inputs, the evolution of the membrane potential and the generation of spike outputs.

When the membrane potential of a neuron surpasses an established threshold, a spike is created and the neuron's potential (V) returns to its resting value (V_{rest}). Afterwards, there is a refractory period, T_{ref}, where the neuron does not react to any input spikes and, therefore, the membrane potential retains its resting value. When the refractory period ends, the neuron will resume updating the membrane potential following Eq. (1).

A discrete time version of Eq. (1) is required to implement the behaviour of a LIF neuron in a digital system. In particular, if a postsynaptic neuron j receives a spike from a presynaptic neuron i at a time step t, the new membrane potential value of neuron N_j is calculated with Eqs. (2) and (3). First, Eq. (2) calculates, $V_j^{leak}(t)$, the loss of membrane potential of neuron N_j from the last processed spike until the current time-step, t. Next, Eq. (3) updates the membrane potential, $V_j(t)$, at a time step t, if N_j is outside the refractory period. In this case, $V_j(t)$ will be updated based on the potential loss, $V_j^{leak}(t)$, and the weights, W_{ij}, of the input synapses, S_{ij}, that have generated a spike.

$$V_j^{leak}(t) = V_j(t-1)\left(1 - \frac{\Delta t}{\tau_m}\right) \tag{2}$$

$$V_j(t) = \begin{cases} V_{rest}, & \text{if } t \in [T_f, T_f + T_{ref}) \\ V_j^{\text{leak}}(t) + \sum_i S_{ij} W_{ij}, & \text{otherwise} \end{cases} \tag{3}$$

where, τ_m is the time constant of a resistor-capacitor (RC) circuit; Δt is the time difference between consecutive time steps; T_f is the time when the neuron fired last; T_{ref} is the time length of the refractory period; S_{ij} is a boolean array that indicates whether the presynaptic neuron, i, has fired a spike and W_{ij} is the synaptic weight of that synapse in the postsynaptic neuron j.

Although there are other methods of discretizing time, the advantage of using Eqs. (2) and (3) is that the membrane potential only needs to be updated when an input spike is received. Unlike most of the existing algorithms for processing SNNs that are clock-driven and need to update the state of each neuron every clock cycle [23], this event-driven update only computes the membrane potential of neurons that receive an input spike. In this way, each neuron is processed more efficiently, not wasting resources

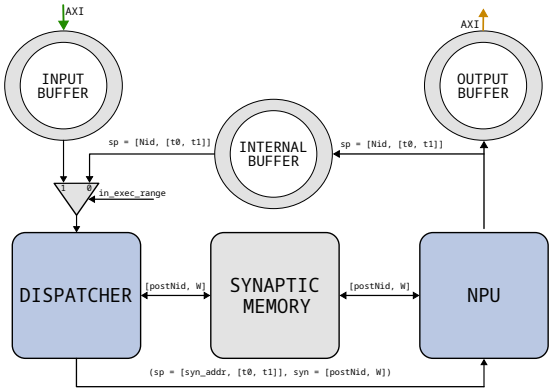


Fig. 2: Overall architecture of the accelerator.

updating neurons that are not active in the network.

B. Spike-Timing-Dependent Plasticity

STDP is a biological learning rule that adjusts the strength of synaptic connections between neurons based on the relative timing of spikes in the pre-synaptic and post-synaptic neurons [15]. The main idea behind this learning rule is to strengthen the presynaptic connections that correlate with neuron firing and to weaken those that do not. This way, the effect of the stimuli that make a neuron fire are reinforced, while the ones that do not are diminished.

Concretely, the synaptic weight change (ΔW) and the updated weight (W_{out}) are calculated as follows:

$$\Delta W = \begin{cases} A_+(W_{max} - W_{in})e^{-\frac{\Delta t}{\tau_+}}, & \text{if } \Delta t > 0 \\ 0, & \text{if } \Delta t = 0 \\ -A_-(W_{in} - W_{min})e^{\frac{\Delta t}{\tau_-}}, & \text{if } \Delta t < 0 \end{cases} \quad (4)$$

$$W_{out} = W_{in} + \Delta W \quad (5)$$

where Δt is the time difference between the pre-synaptic and post-synaptic spikes; A_+ and A_- are positive constants that determine the amplitude of weight changes; τ_+ and τ_- determine the timescales over which potentiation and depression affect decay; W_{in} denotes the synaptic weight before the update and W_{max} and W_{min} define the upper and lower bounds of the synaptic weight, which are enforced on the updated weight W_{out}.

III. Design and implementation of the accelerator

This section presents our neuromorphic accelerator, implemented in VHDL, which is capable of running SNNs with arbitrary topologies. Fig. 2 depicts the overall architecture of the proposed accelerator. The design includes dedicated buffers for input, output, and internal spikes, and an event-driven dispatcher coupled with a pipelined Neuron Processing Unit (NPU) for event processing.

A. Initialization and simulation

The execution of an SNN consists of two stages: an initialization stage where the neuron parameters and network topology are configured, and a simulation stage where the input and internal spikes are processed.

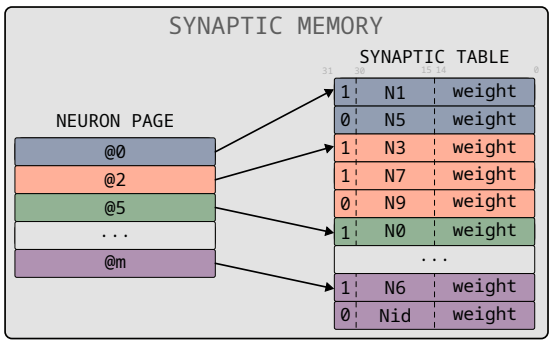


Fig. 3: Structure of the Synaptic Memory which stores the network topology and the synaptic information.

The accelerator starts in the initialization stage, where it remains idle until all the information necessary for the simulation is stored in memory. First, it receives the main network parameters, including the number of neurons and synapses, the membrane potential threshold, the refractory period and the synaptic delay. Then, synaptic information is received and written to the Synaptic Memory. Once the last synapse entry is written, the simulation stage begins.

During simulation, the accelerator receives spikes and encodes them in 32-bit words, where the most significant 16 bits store the neuron ID and the least significant 16 bits store the current time-step. These spikes are stored in the input buffer. The dispatcher selects spikes either from the input buffer, containing external stimuli to the network, or from the internal buffer, storing the spikes generated during simulation, and dispatches work to the NPU based on the network topology stored in the Synaptic Memory. The NPU updates neuron state according to the LIF model and applies STDP to update synaptic weights. If the membrane potential surpasses the configured threshold, it generates a spike, which is written to the internal buffer for further processing and to the output buffer as a result of the simulation.

B. Spike Dispatcher

During the simulation stage, the dispatcher in Fig. 2 selects the next spike to be processed by examining the input and internal buffers of the accelerator. As it operates in an event-driven manner, the simulated time advances with the time-step encoded in the processed spike. Once a spike is selected, the dispatcher proceeds to read the synaptic information stored in the Synaptic Memory and dispatches work to the NPU.

As the number of synapses in an SNN can be very high, an efficient organization of synaptic data is essential. In our implementation, we use a two-memory structure as illustrated in Fig. 3. It includes a Synaptic Table to store synaptic information and a Neuron Page memory to index into it. Synapses are stored contiguously in the Synaptic Table and sorted by presynaptic neuron to simplify sequential dispatch. Each synapse entry is currently encoded in a 32-bit word. As shown in Fig. 3, the most significant bit

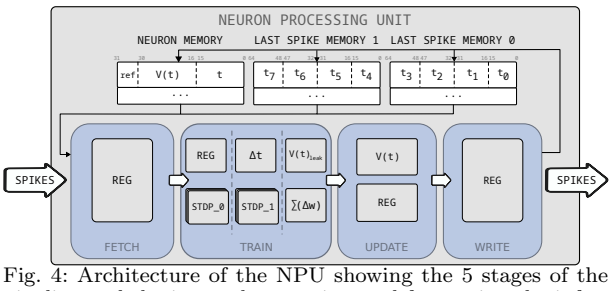


Fig. 4: Architecture of the NPU showing the 5 stages of the pipeline and the internal memories used for storing the information necessary to process the synapses.

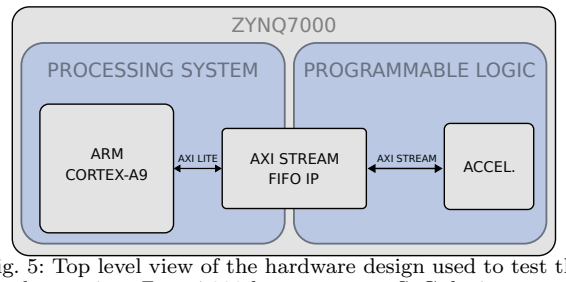


Fig. 5: Top level view of the hardware design used to test the accelerator in a Zynq 7000 heterogeneous SoC device.

Tabla I: ZYBO Z7-20 FPGA board resources.

I/O Pins	Available IOB	LUT	FlipFlop	BRAM	DSP	BUFGs
400	125	53200	106400	140	220	32

indicates whether the next entry still corresponds to the same presynaptic neuron; the next 16 bits encode the postsynaptic neuron ID; and the remaining 15 bits store the synaptic weight. The Neuron Page memory stores, for each neuron, the address of its first synapse entry within the Synaptic Table.

Taking advantage of the proposed memory scheme, the dispatcher iterates through the synapse entries of the presynaptic neuron sequentially until the most significant bit of a synapse entry is 0. In each iteration, the dispatcher passes the postsynaptic neuron ID and synaptic weight read from the Synaptic Table to the NPU. Since the first synapse requires two memory accesses (page lookup followed by synapse entry read), a pipeline bubble would be introduced: after reading a spike from the buffer, the first synapse is not available for the next two clock cycles. To hide this latency and increase throughput, the dispatcher prefetches the next spike's synapse address by reading it from the Neuron Page memory while the loop for the previous spike is still running. As a result, in most cases, the next synapse can be ready in a single clock cycle without stalling the pipeline.

C. Neuron Processing Unit

As explained in Section II-A, modelling a LIF neuron requires three values: membrane potential, the timestamp of the last processed spike and whether the neuron is in its refractory period. We store these values for all neurons in a memory composed of 32-bit words, using one word per neuron. The 15 least significant bits represent the membrane potential, the next 16 bits store the last spike timestamp, and the most significant bit indicates whether the neuron is in the refractory period.

The NPU receives from the dispatcher the postsynaptic neuron to process, the synaptic weight and the current and previous time-steps associated with the synaptic spike. This information is used to update the neuron's membrane potential and the synaptic weight following Eqs. (3) and (5). As shown in Fig. 4, processing is divided into four stages: (1) reading neuron and synapse data, (2) updating the synaptic weight according to the previous spike, (3) computing the updated membrane potential and (4) writing the updated neuron state back to memory.

If the new membrane potential computed in the third stage surpasses the configured threshold, the NPU outputs a spike containing the ID of the firing neuron and the time-step at which it must be processed. The latter is computed by adding the synaptic delay to the current time-step. The generated spike is stored in the internal buffer for further processing and also in the output buffer to be sent through the output interface.

Given that the membrane potential loss in Eq. (2), $V_j^{leak}(t)$, is independent of the synaptic weight, in order to reduce the latency of the operations to update the membrane potential, the loss is calculated in parallel with the computation of STDP in the training stage. Moreover, this operation consists of a division and a fixed-point multiplication that would increase the complexity of the hardware. To reduce it, we avoid the division by having the percentage of loss pre-calculated and scaled up by 128, so the fixed-point multiplication is done with integer values and then scaled down to the actual value with a simple shift operation.

Since the computation of STDP requires three clock cycles, see the next subsection for details, and the remaining stages can be executed in one cycle, sequential processing would achieve a rate of one neuron update every six cycles. However, because each event is independent, we apply pipelining, so that each pipeline stage can process a different postsynaptic neuron. This increases throughput to one event per clock cycle.

D. STDP implementation optimizations

The STDP rule requires updating the synaptic weight depending on the time difference between the presynaptic and postsynaptic spikes. With the data structure described above, it is easy to calculate all the post-before-pre events as they are spatially related in the memory. However, a postsynaptic spike can be caused by multiple presynaptic neurons, which are not directly accessible with this data structure.

A simple solution would be to replicate the Synaptic Memory in Fig. 3 but sorted from post to pre. However, this would consume a large amount of memory resources and substantially reduce the number of supported synapses. In addition, it would require maintaining coherence between the two copies. Taking advantage of the event-driven execution, the synaptic weights are updated only when the next presynaptic spike is processed [25]. Therefore, weight changes are not triggered by postsynaptic spikes and

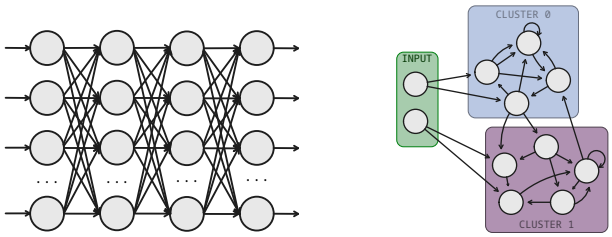
are postponed until the next presynaptic event. This removes the need for a replicated Synaptic Memory while preserving the effect of STDP rules.

With this approach, and taking into account the latest spikes from the postynaptic neurons, if their firing rate is higher than that of the presynaptic neuron, some weight updates might be missed. However, it is not feasible to buffer all the spikes until the weight of the last presynatic neuron has been updated, as this would require a large amount of memory resources.

According to [25], it is possible to approximate a buffer size that supports most weight updates while reducing on-chip memory usage. They propose an equation in which the output rate of the postsynaptic neuron must be smaller than the average input rate multiplied by the buffer size. This provides a constraint on the neural network configuration that can be simulated without significantly losing weight update contributions. In their case, after measuring these values for their tested networks, they selected a buffer size of 8. To determine an appropiate buffer size for the proposed accelerator, we have carried out several independent experiments simulating neural networks with arbitrary topologies that fit within our FPGA. Each network was instantiated with different neural parameters, weight distributions and numbers of neurons and synapses. In each simulation, we counted the number of postsynaptic spikes occurring between consecutive presynaptic spikes. Consistent with their findings, for the network configurations tested in this work, the number of synapses requiring more than 8 stored spikes is very low. Therefore, this design buffers up to the last 8 spike timestamps per neuron, providing a trade-off that minimizes on-chip memory usage while preserving most weight update contribution.

The synaptic weight change (ΔW) in Eq. (4) is calculated using exponential operations that would result in very complex hardware and high-latency data paths. To reduce the complexity of the resulting hardware, the exponential values are pre-calculated for a STDP window of 32 time-steps and stored in a Look-Up Table. Weight changes for spikes outside the specified time window are considered to have negligible influence and are therefore discarded.

With this setup, the computation of ΔW is reduced to bounding the weight and performing two multiplications. A Digital Signal Processor (DSP) slice is used to reduce power consumption, increase the resource efficiency of the design and reduce the latency of the data paths. This component reduces the resource consumption by integrating multiple accumulators and multipliers. This datapath is cloned 8 times in the design, since the weight change needs to be calculated for all the time-steps in the buffer. Then the weight changes are accumulated obtaining the new value for the synapse. In order to allow high frequency clocks, the training phase of the NPU is divided into a three-stage pipeline. The first two stages correspond to the two steps required by the DSP component to calculate the weight updates, and the final stage is used to accumulate all the updates into the new synaptic weight value. See again Fig. 4.



(a) Multilayered network. (b) Random network.

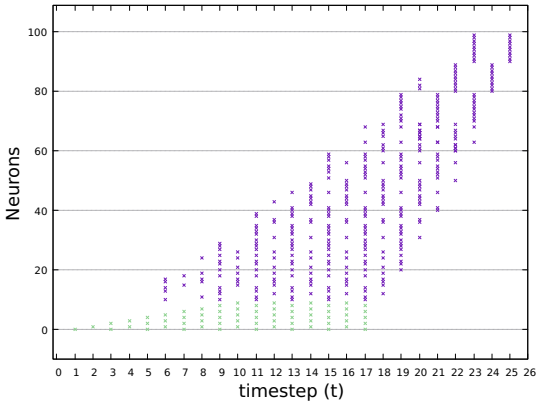
Fig. 6: Example of the topologies used in our experiments. Grey circles represent the neurons and arrows represent synapses. The tail of the arrow indicates the presynaptic neuron and the arrowhead the postsynaptic one.

IV. Experimental evaluation

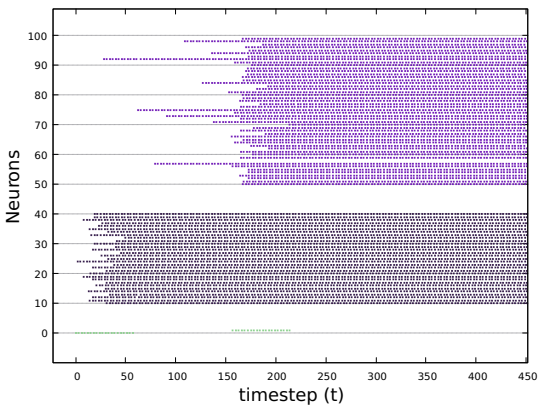
The accelerator has been tested using a hardware-software co-design approach on a Digilent ZYBO Z7-20 FPGA board. The integrated Arm Cortex A9 CPU used as the Processing System (PS) provides a 90 MHz clock for the Programmable Logic (PL) where the accelerator is located. Subsequent tests have revealed that this is not the maximum frequency that the accelerator can achieve, see below in Section IV-C. In order to allow for seamless connectivity with other possible IP Blocks implemented in the PL in our envisioned IoT platform, the input and output interfaces of the accelerator are implemented following the AMBA4 AXI Stream protocol [26]. This protocol consists of a simple valid and ready handshake that allows streaming data between two devices using 32-bit or 64-bit packets. However, as the CPU only has AXI Full or Lite ports, the AXI Stream FIFO IP of Xilinx is used as a bridge between the PS and PL as illustrated in Fig. 5.

In this setup, the PS is responsible for configuring the test networks and providing them with input spikes to feed the simulation. In addition, it is in charge of collecting the spikes generated by the accelerator and storing them in a `.csv` file, which is used to plot the spike trains in Section IV-A. The PS-PL communication is implemented by a C program that interfaces with the Xilinx FIFO IP by providing the data and the length of the streams that the IP sends to the accelerator. In addition, the program reads the output spikes of the accelerator that are stored in the FIFO IP periodically.

All the networks used for the tests share identical neural parameters. The membrane potential threshold is set to 140, the refractory period lasts 3 time-steps, and the spike propagation delay is 1 time-step. Additionally, the membrane potential exhibits a leak of 5 % per time-step. To have a more varied behaviour, we initialized synaptic weights for both networks using a uniform distribution that generates random values between 0 and 50.



(a) Propagation of spikes in a layered topology.



(b) Propagation of spikes in a clustered topology.

Fig. 7: Spike trains generated by the networks under evaluation. Spikes in green are the inputs of the simulation; those in purple are generated by the SNNs based on these inputs.

A. Examples of execution

To validate the correct operation of the accelerator, two network topologies have been considered. We start with a standard, feedforward multilayered topology, similar to the one depicted in Fig. 6a, but composed of 10 fully-connected layers with 10 neurons per layer, resulting in 100 neurons and 1000 synapses. The second network is a randomly-connected topology, like the one in Fig. 6b, but composed of 100 neurons and 912 synapses. In order to support different tests, the accelerator has been synthesized to a maximum of 4,096 neurons and 65,536 synapses, allowing much larger networks than the ones described above.

Fig. 7a shows the spikes generated by the first network. The spikes from the input neurons are drawn in green while the rest of spikes are in purple. To test the operation under an increasingly higher number of input spikes, we gradually excite each of the neurons in the input layer. Neuron 0 is excited at time-step 1, neuron 1 at time-step 2 and so on, finishing with neuron 9 at time-step 10. Once excited, the input neurons will spike periodically every 2 time-steps. All input neurons are de-excited after time-step 17, so they stop spiking altogether. As can be seen in Fig. 7a, due to the low synaptic weights, 9 spikes of the input layer are required to generate spikes in the first layer. Similarly, the second layer does not start firing until the second round of spikes from the first layer. In general, we can see that the propagation from one layer to the next takes between 1 and 3 time-steps. As the number of input spikes increases, the delay of the spike propagation across the network is reduced, and the overall activity increases. Once the input neurons are de-excited and no new spikes are introduced in the simulation, the remaining layers gradually become silent. This is the anticipated behaviour of a layered feedforward network.

The main purpose of this experiment is to demonstrate that our design is able to support arbitrary neural nets. Synaptic connectivity for the second network is generated by dividing the desired number of synapses by the number of neurons in the network, obtaining thus the connection ratio of each neuron. Each ratio is then multiplied by a randomization factor to add a variance of between -5 and 5 to the number of connection counts. The output neuron for each connection is sampled from a uniform distribution. This approach allows multiple synapses between neuron pairs, as well as recurrent self-connections.

We formed a similar network to the one shown in Fig. 6b. Following the color scheme of the figure, the green cluster represents the 10 input neurons. These neurons do not have any input synapses and each one is connected to a single cluster. The blue and purple boxes show two densely interconnected clusters with very sparse inter-cluster connectivity. In this case, the blue cluster contains 30 neurons while the purple one is made up of 50.

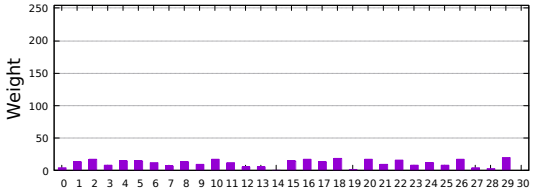
The spike train for the second network is shown in Fig. 7b. In this case, the first two out of the 10 input neurons have been excited 20 times every 3 time-steps. The first to spike is neuron 0 which is connected to the lower cluster (10-40) and 100 time-steps after it finishes neuron 1 starts exciting the upper cluster (50-100). It can be seen that some neurons of the second cluster get activated by the activity in the first one. However, it is not until neuron 1 starts firing, that the activity increases. This is because, there is not enough activity in the cluster to surpass some of the neuron's thresholds. Again, this is the expected behaviour for a randomly connected network.

B. STDP Execution test

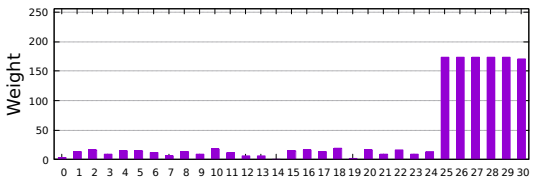
We carried out two tests to validate the implementation of the presynaptic spike driven STDP. Both tests show how a strong correlation of spikes increases the strength of the synapses while the weight of not correlated and sparse synapses is barely affected. Both experiment share the same neural parameters as before and $A_+ = 1$ and $A_- = 0{,}5$ constant values of the STDP learning rule.

As in [25], the first experiment uses a small two layer network with 30 presynaptic neurons and a single postsynaptic neuron. Initial synaptic weight were sampled from a uniform distribution ranging from 0 to 20, which represent a relatively low influence on the postsynaptic neuron.

The firing rate of the first 25 presynaptic neurons are drawn from a uniform distribution that ranges



(a) Initial weights.



(b) Final weights after 400 time-steps.

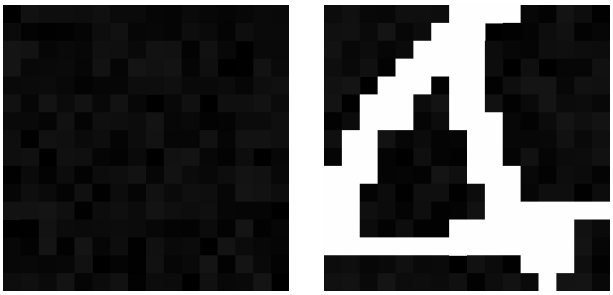
Fig. 8: Updating of weights in the STDP experiment.

from 10 to 80 time-steps per spike, while the last 5 fire simultaneously at a frequency of 3 time-steps per spike. Fig. 8 shows the measurements of the weights before (8a) and after (8b) running for 400 time-steps. Fig. 8b illustrates the expected result of the experiment where the synaptic weights of the highly correlated spikes of the last neurons increased significantly compared to the rest of the synapses. Due to the sparse activity of neurons 0-24 and the small time window selected for the STDP, their synaptic weights are not altered. However, since the last 5 neurons spike frequently and simultaneuosly, their weights increase from small values below 20 to 173, which is enough for a single synapse make the output neuron fire. Note that these values are far from the maximum synaptic weight allowed that is 255, as each spike is also considered to have low correlation with the previous spike of the output neuron.

For the second test, we used a subset of the Semeion handwritten digit dataset [27]. This dataset consist of 1593 samples of black and white 16×16 images of handwritten digits ranging from 0 to 9. In this case, each presynaptic neuron represents a pixel image, which results in 256 input neurons and a single output neuron. Each active pixel is treated as the neuron firing every 5 time-steps. Fig. 9 shows the weights of the input synapses as 16×16 grey scale images. Weights closer to 0 are represented with darker values and weights closer to 255 are drawn closer to white. After running each sample for 50 time-steps, the weights shift from a uniform distribution (left images) to a clear number (right images). In this case, we showcase the before and after for two samples of the dataset. In each case, the weights of the active synapses are increased to the maximum value that is 255, while the rest of the synaptic values in the network remain constant.

C. Hardware utilization and power dissipation

We move now to discuss the resource and power consumption of our proposal. The accelerator has been synthesized and implemented using Xilinx Vivado. The target system was a Digilent ZYBO Z7-20 FPGA board. Table I shows the resources of the XC7Z020-1CLG400C FPGA chip that resides within this evaluation board.



(a) Initial weights sample 80 (b) Final weigths sample 80

(c) Initial weights sample 167 (d) Final weigths sample 167

Fig. 9: Weight map of the synapses for two samples of the Semeion dataset before and after passing through the network.

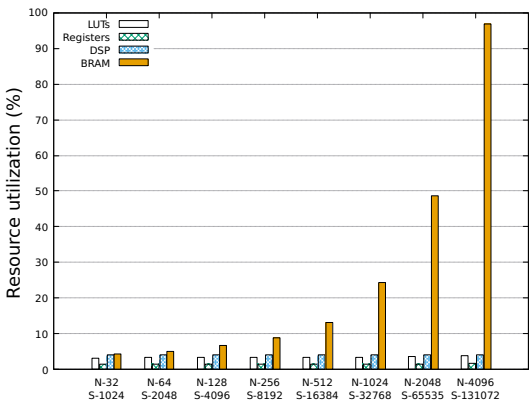


Fig. 10: Hardware resource utilization for varying maximum network size configurations.

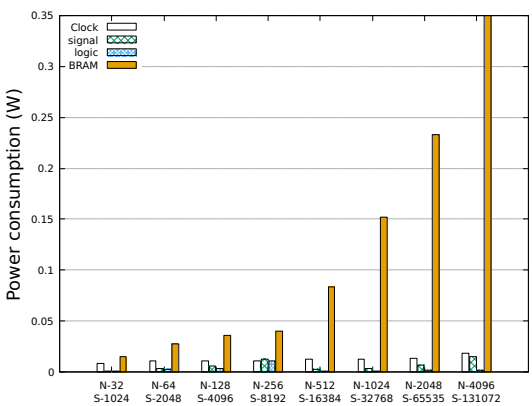


Fig. 11: Power consumption breakdown of the accelerator. Static (0.250 W) consumption not shown for the sake of clarity.

Table II shows the comparison between some of the most relevant state-of-the-art hardware accelerators and this work. It can be seen that while the accelerators that use an external memory can simulate larger networks, this work improves upon the state-

of-the-art results for platforms with on-chip memory. In addition, our design consumes significantly fewer resources per neuron and has lower power consumption than the rest of the implementations, which makes it ideal for embedded IoT applications.

To get an overview of how the network parameters affect resource and power consumption, the accelerator has been synthesized and implemented with incrementing maximum numbers of neurons and synapses. For simplicity, we increased these parameters in powers of two, starting with networks of 32 neurons and 1024 synapses until the maximum capacity of our board is reached at 4096 neurons and 131,072 synapses. Note that large configurations are limited to run at a clock frequency of 100 MHz while smaller configurations can achieve 105 MHz. This is because of the physical distance between some BRAM blocks of the synaptic memory and their output registers.

Fig. 10 shows the percentage of resources that each of the networks uses. It can be seen that even though the use of the rest of resources remains almost constant, our design relies heavily on Block RAMs, which become the limiting factor for large SNNs. This is because, in order to be as generic as possible for different networks, synapses are stored as full 32-bit words. Following this scheme, in the zybo board the accelerator can get a maximum of 2^{17} synapses before running out of BRAM to synthesize the design. However, these results show that the design leaves considerable amount of free resources (LUTs, registers, DSPs) for other IP blocks which could be used in embedded applications, e.g., sensor input processing and output analysis blocks. Moreover, we could employ part of the unused hardware to integrate a larger number of NPUs to speed up simulations, if necessary.

In order to get an accurate power estimation of the accelerator, a post-implementation timing simulation has been performed for each network size configuration. In these simulations, a random network, almost as big as the configured maximum size, like the one shown in Fig. 6b has been tested to provide a precise switching activity for the implemented logic. According to the Vivado implementation, the total power consumption of the accelerator when modelling the maximum SNN size is around 0.667 W (0.286 W of static power plus 0.386 W of dynamic power). Fig. 11 shows that, as expected from the resource consumption, the most power-hungry component is the BRAM, which can utilize up to 92 % of the total dynamic power for the largest modelled SNNs.

Overall, the utilization and power analysis show that despite the largest configuration being limited by the available BRAM on the board, this work achieves high resource efficiency and low power consumption, confirming the suitability of the proposed accelerator for low-power, resource-constrained embedded neuromorphic applications.

D. Comparison with the state of the art

The development of systems to simulate the execution of SNNs has received significant attention in recent years, especially in the field of artificial intelligence. In order to overcome the limitations of traditional processing systems like CPUs and GPUs caused by the Von Neumann bottleneck [28], many hardware-oriented implementations have been proposed.

Table II summarizes the main features of some FPGA-based implementations and compares them with our proposal. We can see that several platforms have been developed with the objective of improving the efficiency and speed of SNNs compared with standard computing devices. Unlike our proposal that focuses on neuron to neuron connectivity, the rest of accelerators only implement layered topologies, which reduces the information necessary to route spikes but limits the flexibility of the possible SNNs.

Additionally, some of the existing architectures which can execute and train a simplified SNN model are based around a clock-driven neuron update [23], [29], [30]. Such architectures can not take full advantage of the event-driven nature of SNNs, since they need to update the state of the whole network incurring in extra performance and energy overheads. In fact, the proposal of Gupta et al. [23] physically implements the network neuron and synapses, and only use 16 BRAM FIFO queues to store the output neuron synapse weights. This approach allows real parallel execution of all the neurons in the network, but greatly limits the size of the executed network.

However, Spiker+ [30] showcases a network with similar amount of synapses with far less BRAM consumption, which reduces their power consumption. This is due to the lower 4-bit resolution of the synaptic weight value. However, The membrane potential of the neurons are stored in 6-bit word Look-Up Table for each layer, increasing the number of LUTs per neuron of the accelerator. Similarly, Lopez-Asuncion et al. [29] stores the neurons potential followed by all the synaptic weights of the neurons in a 16-bit word memory. Since we use 32-bit words, their accelerator doubles the number of synapses with similar BRAM utilization compared to our proposal.

Skydiver [31] and Cerebron [32] optimize the workload of their computing units to achieve better energy efficiency. In addition, FireFly [33] uses the DSP48E2 block from Xilinx UltraScale FPGA boards to optimize the processing efficiency of the generated spikes. However, these designs focus on Convolutional SNNs, which greatly limits the possible SNN topologies that can be accelerated as opposed to our proposal which allows configuring any topology without the need to resynthesize the accelerator.

NeuroFlow [12] is an accelerator for large scale neural network simulation. It can simulate networks of 600,000 neurons both LIF and IZH in a 6-FPGA system. Nevertheless, as a large scale implementa-

tion, it is not well suited for compact embedded applications because of the high power dissipation.

In contrast, as our implementation, FPGA-NHAP [24] focuses on a smaller scale, energy-efficient solution suitable for embedded systems. Indeed, this platform externalizes the synaptic information in an SDRAM in order to increase the network size, while reducing on-chip memory usage. Nevertheless, as the access to that information has a higher latency, they use the internal BRAMs to cache the active synapses. This results in more memory consumption than our approach. Moreover, external RAMs provide additional energy consumption which may be a scarce resource in some embedded devices.

All in all, it can be seen that there is a trade-off between the memory usage and the flexibility of the designs. Even though other accelerators consume less on-chip memory per synapse, our proposal is the only one that allows simulation of more biologically plausible network topologies that are not as structured as feedforward layered networks. Furthermore, we obtain a competitive number of neurons and synapses with similar hardware utilization.

V. Conclusions and Future Work

In this work, we designed and implemented a new FPGA-based neuromorphic accelerator for low-power IoT applications, capable of simulating SNNs with arbitrary topology and support for on-chip learning with STDP. The proposed accelerator includes a NPU that updates neuron membrane potentials and synaptic weights upon spike reception and generates output spikes when the firing threshold is exceeded. By applying pipelining techniques, the processing throughput of the NPU is increased. The NPU is fed spikes by a dispatcher that selects the postsynaptic spikes based on the configured network topology. Critically, our accelerator relies on an efficient organization of network and synaptic information while minimizing FPGA resource usage.

The accelerator can communicate with external devices through the AXI Stream protocol, which is used to configure SNN parameters and network topology, provide input spikes, and retrieve simulation results. This provides simple compatibility with other IP Blocks or devices. In this particular case, we show some experimental work where the simulation is driven by the processing system of a Zynq 7000 board. However, our envisioned scenario is to use the accelerator as a stand-alone block that interfaces with the sensors and actuators of an IoT device.

By using a presynaptic spike-driven approach, the accelerator implements the STDP learning rule in an event-driven manner while reducing the memory resources required to associate presynaptic and postsynaptic spikes. This has been verified through experiments that show how synapses with spikes that are directly related increase in weight, whilst those with no such relationship remain unaffected.

Experimental results show that, as expected for SNN accelerators, the design relies heavily on on-chip memory resources. Nevertheless, it achieves very low power consumption and maintains an almost constant usage of logic resources across configurations, resulting in less than one LUT and one flip-flop per neuron in the largest configuration. Moreover, we found that the proposed accelerator is competitive when compared with state-of-the-art FPGA-based solutions in terms of resource consumption, power dissipation and flexibility, including support for online learning and arbitrary network topologies.

The current implementation includes a single NPU. As future work, we aim to extend the architecture towards a multicore design to increase processing throughput. This will require the introduction of a memory arbitration or bus protocol to prevent conflicts when multiple NPUs access neuron and synapse memories. With multiple NPUs operating in parallel, the accelerator could process several neurons simultaneously, enabling higher throughput for larger and more complex networks. Finally, to further improve the flexibility of the accelerator, we will also explore the implementation of additional neuron and plasticity models beyond LIF and STDP.

Agradecimientos

This work is supported by the Basque Government grants IT1504-22 and KK-2025/00098. It is also supported by the grants PID2023-152390NB-I00 and CNS2023-144315 funded by MICIU/AEI/10.13039/9/501100011033, "European Union NextGenerationEU/PRTR" and by "FEDER funds". Dr. Javier Navaridas holds a Ramón y Cajal fellowship funded by the Spanish Ministry of Science, Innovation and Universities (RYC2018-024829-I), funded by MCIN/ AEI/10.13039/501100011033 and, as appropriate, by "ESF Investing in your future" or by "European Union NextGenerationEU/PRTR".

Referencias

[1] Traore et al., "Deep convolution neural network for image recognition," *Ecological informatics*, vol. 48, pp. 257–268, 2018.

[2] Diwan et al., "Object detection using yolo: Challenges, architectural successors, datasets and applications," *multimedia Tools and Applications*, vol. 82, no. 6, pp. 9243–9275, 2023.

[3] Guo et al., "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[4] Wulfram Gerstner and Werner M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*, Cambridge University Press, 2002.

[5] Soleimani et al., "Biologically inspired spiking neurons: Piecewise linear models and digital implementation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 12, pp. 2991–3004, 2012.

[6] Eugene M Izhikevich, "Which model to use for cortical spiking neurons?," *IEEE transactions on neural networks*, vol. 15, no. 5, pp. 1063–1070, 2004.

[7] Denis Noble, "A modification of the hodgkin—huxley equations applicable to purkinje fibre action and pacemaker potentials," *The Journal of physiology*, vol. 160, no. 2, pp. 317, 1962.

[8] Benjamin Lindner, Lutz Schimansky-Geier, and André Longtin, "Maximizing spike train coherence or incoherence in the leaky integrate-and-fire model," *Physical Review E*, vol. 66, no. 3, pp. 031916, 2002.

[9] Stoliar et al., "A leaky-integrate-and-fire neuron analog realized with a mott insulator," *Advanced Functional Materials*, vol. 27, no. 11, pp. 1604740, 2017.

Tabla II: Results of the relevant state-of-the-art FPGA SNN hardware accelerators.

	Spiker+ [30]	FireFly [33]	Gupta et al. [23]	NHAP [24]	López-Asunción et al. [29]	SNNorlax
Target	XC7Z020	xczu7ev	Virtex-6	Kintex-7	Artix-7	ZYBO Z7-20
Clock	100 MHz	300 MHz	100 MHz	200 MHz	130 MHz	100 MHz
Algorithm	Clock-driven	Event-driven	Clock-driven	Mixed	Clock-driven	Event-driven
Model	LIF	IF/LIF	Simplified LIF	LIF/IZH	Simplified LIF	LIF
Learning	Offline	Offline	STDP	Offline	Offline	STDP
Weights	On-chip BRAM	Off-chip SDRAM	On-chip BRAM	Both	On-chip BRAM	On-chip BRAM
Neurons	922	3444	800	2842	2048	4096
Synapses	101632	20980	12544	1861632	262144	131072
LUTs/neuron	8.25	9.29	70	16.32	2.67	0.29
FFs/neuron	-	-	29	10.70	2.86	0.25
B/URAM	18/0	25/40	16/0	150/0	130/0	122.5/0
Power	0.18 W	2.55 W	-	0.53 W	-	0.667 W

[10] Eshraghian et al., "Training spiking neural networks using lessons from deep learning," *Proceedings of the IEEE*, vol. 111, no. 9, pp. 1016–1054, 2023.

[11] Daniel Neil and Shih-Chii Liu, "Minitaur, an event-driven fpga-based spiking network accelerator," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 22, no. 12, pp. 2621–2628, 2014.

[12] Kit Cheung, Simon R Schultz, and Wayne Luk, "Neuroflow: a general purpose spiking neural network simulation platform using customizable processors," *Frontiers in neuroscience*, vol. 9, pp. 516, 2016.

[13] Furber et al., "Overview of the spinnaker system architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, 2013.

[14] Joan Stiles, "Neural plasticity and cognitive development.," *Developmental neuropsychology*, vol. 18, no. 2, 2000.

[15] Natalia Caporale and Yang Dan, "Spike timing–dependent plasticity: a hebbian learning rule," *Annu. Rev. Neurosci.*, vol. 31, no. 1, pp. 25–46, 2008.

[16] Donald Olding Hebb, *The organization of behavior: A neuropsychological theory*, Psychology press, 2005.

[17] Omar Abdel Wahab, "Intrusion detection in the iot under data and concept drifts: Online deep learning approach," *IEEE Internet of Things Journal*, vol. 9, no. 20, pp. 19706–19716, 2022.

[18] Bharath Sudharsan, John G. Breslin, and Muhammad Intizar Ali, "Imbal-ol: Online machine learning from imbalanced data streams in real-world iot," in *2021 IEEE International Conference on Big Data (Big Data)*, 2021, pp. 4974–4978.

[19] Sawada et al., "Truenorth ecosystem for brain-inspired computing: Scalable systems, software, and applications," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 130–141.

[20] Mike Davies et al., "Taking neuromorphic computing to the next level with loihi3," *Intel Labs' Loihi*, vol. 2, no. 1, 2021.

[21] Höller et al., "Open-source risc-v processor ip cores for fpgas—overview and evaluation," in *2019 8th Mediterranean Conference on Embedded Computing (MECO)*. IEEE, 2019, pp. 1–6.

[22] Murat Isik, "A survey of spiking neural network accelerator on fpga," *arXiv preprint arXiv:2307.03910*, 2023.

[23] Gupta et al., "Fpga implementation of simplified spiking neural network," in *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2020, pp. 1–4.

[24] Yijun Liu, Yuehai Chen, Wujian Ye, and Yu Gui, "Fpga-nhap: A general fpga-based neuromorphic hardware acceleration platform with high speed and low power," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 6, pp. 2553–2566, 2022.

[25] Jongkil Park and Sang-Don Jung, "Presynaptic spike-driven spike timing-dependent plasticity with address event representation for large-scale neuromorphic systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 6, pp. 1936–1947, 2020.

[26] arm, "Amba axi-stream protocol specification," Available at: https://developer.arm.com/documentation/ihi0051/latest, 2021, Accessed: 08/12/2025.

[27] "Semeion Handwritten Digit," UCI Machine Learning Repository, 1998, DOI: https://doi.org/10.24432/C5SC8V.

[28] Eloína Peláez, "Parallelism and the crisis of von neumann computing," *Technology in Society*, vol. 12, no. 1, pp. 65–77, 1990.

[29] Samuel López-Asunción and Pablo Ituero Herrero, "Flexible deep-pipelined fpga-based accelerator for spiking neural networks," in *2023 38th Conference on Design of Circuits and Integrated Systems (DCIS)*, 2023, pp. 1–6.

[30] Carpegna et al., "Spiker+: a framework for the generation of efficient spiking neural networks fpga accelerators for inference at the edge," *IEEE Transactions on Emerging Topics in Computing*, 2024.

[31] Qinyu Chen, Chang Gao, Xinyuan Fang, and Haitao Luan, "Skydiver: A spiking neural network accelerator exploiting spatio-temporal workload balance," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 12, pp. 5732–5736, 2022.

[32] Qinyu Chen, Chang Gao, and Yuxiang Fu, "Cerebron: A reconfigurable architecture for spatiotemporal sparse spiking neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 10, pp. 1425–1437, 2022.

[33] Li et al., "Firefly: A high-throughput hardware accelerator for spiking neural networks with efficient dsp and memory optimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 8, pp. 1178–1191, 2023.

Deep Learning 3D Object Detection and Classification at the Edge of IoT

Inés Pérez, Rogelio Hernández, Jorge Portilla[1]

Abstract— **Modern 3D LiDAR object detection usually relies on complex architectures such as Transformers and Sparse Convolutions. While achieving high accuracy, these models introduce hardware-unfriendly behaviors, complicating deployment on edge Internet of Things (IoT) devices. This work presents a feasibility study of modern pillar-based architectures on the Hailo-8L Neural Processing Unit (NPU). The transition from PointPillars to the State of the Art PillarNeSt architecture is evaluated, with an analysis of the trade-offs between detection accuracy (mAP) and inference latency. Through benchmarking with the nuScenes dataset, performance and model precision are characterized, and the architectural bottlenecks determining real-time viability are identified. This study outlines the specific hardware constraints and optimizations required to deploy 3D perception on edge systems.**

Keywords— **3D object detection, Deep Neural Network (DNN), Light Detection and Ranging (LiDAR), Internet of Things (IoT), edge AI, Neural Processing Unit (NPU), real-time systems, heterogeneous computing, hailo-8L.**

I. Introduction

OBJECT detection and classification at the edge are critical for Internet of Things (IoT) to ensure real-time response, data privacy, and reduced bandwidth consumption. 3D Light Detection and Ranging (LiDAR) sensors are widely adopted for these tasks as they provide explicit geometric information through sparse point clouds. These capabilities are essential for self-driving applications [1], safety-critical monitoring in industrial facilities, urban intersections and railway level crossings [2]. Current edge implementations often rely on lightweight tracking algorithms [3] rather than neural networks to meet strict resource constraints. On the other hand, high-performance Deep Learning (DL) models for LiDAR perception are widely used [4] but typically require processing on power-intensive Graphics Processing Units (GPUs) or premium integrated System On Chips (SOCs), and these platforms often exceed the constraints of mass-market robotics and edge IoT applications. While alternative systems perform detection using different sensors or non-DL methods, the deployment of LiDAR-based DL on edge hardware is an emerging area of research [5], [6].

In this landscape, Neural Processing Units (NPUs) emerge as a solution for high performance neural processing. Specifically, the Hailo-8 series accelerators stand out for their low cost and power consumption, making them ideal for edge deployment. However, implementing deep learning models on this hardware poses some challenges. State of the Art (SOTA) 3D networks frequently utilize sparse convolutions and dynamic operations to manage point cloud sparsity [7], [8], [9]. These operations conflict with the static computational graphs and deterministic memory allocation requirements of NPUs like the Hailo-8L [6]. Furthermore, the transition to fixed-point INT8 quantization is a prerequisite for NPU acceleration, yet LiDAR architectures exhibit higher sensitivity to bit-width reduction than 2D vision models [10].

This study addresses the practical challenges of deploying modern 3D detection models on resource-constrained hardware by benchmarking architectures ranging from established models, such as PointPillars [11], to contemporary designs, such as PillarNeSt [12]. The main contributions of this work are as follows:

- **Feasibility Benchmarking**: a comprehensive evaluation of models on the Hailo-8L accelerator, establishing a performance study for IoT platforms.
- **Hardware-Algorithm Analysis**: an analysis of inference bottlenecks and the loss of mean Average Precision (mAP) due to quantization.

II. Related Work

DL has established a framework for 2D object detection and classification, supported by a wide range of architectures and large-scale datasets [13], [14]. For 3D perception, LiDAR is widely used because it captures explicit geometric data via sparse point clouds [15]. This precision drives the need for 3D architectures that can handle irregular spatial data.

Traditionally, 3D object detection research has been divided into three main paradigms: projection-based, point-based, and voxel-based methods [15]. Point-based approaches extract features directly from raw coordinates to avoid discretization loss; but suffer from high computational costs and irregular memory access patterns. Voxel-based methods discretize the input into a structured 3D grid. Projection-based methods map 3D data onto 2D planes, such as Bird's Eye View (BEV) or Range View (REV). Within the volumetric paradigm, pillar-based architectures, initiated by PointPillars [11], represent a specialized case that combines voxelization with projection to generate a 2D pseudo-image. This transformation allows the backbone to utilize standard 2D convolutions, which are the most

[1]Centro de Electrónica Industrial y Sistemas Multimodales, Universidad Politécnica de Madrid. Emails: ines.perez.delgado@upm.es, r.hlorite@upm.es, jorge.portilla@upm.es

optimized operations in modern accelerators. While this projection involves a loss of vertical spatial information, recent studies demonstrate that pillar-based designs achieve better recall and latency in low-density environments compared to voxel-based methods [16].

Edge hardware constraints dictate model deployment. While GPUs offer high throughput and mature software support, they exhibit high power consumption. Specialized alternatives, including Tensor Processing Units (TPUs) and NPUs, provide domain-specific acceleration for tensor operations within a limited power envelope, but they require deterministic computational graphs and specialized memory management [17]. In addition, those accelerators also necessitate quantization, transitioning from 32-bit Floating Point (FP32) to 8-bit Integer (INT8) precision. 3D LiDAR models are more sensitive to quantization than 2D vision models [10], primarily because the regression of spatial coordinates requires high numerical precision to avoid critical localization inaccuracies.

Precision loss is primarily mitigated through Quantization Aware Training (QAT) and Post-training Quantization (PTQ). QAT incorporates quantization effects during the training loop to maintain high accuracy; however, it is computationally slow and requires retraining with the full dataset. In contrast, PTQ optimizes the model post-training through calibration using a small data subset. Advanced PTQ methods rapidly optimize parameters to recover accuracy without full retraining. Although QAT typically is more accurate, specialized PTQ methods can effectively reduce accuracy degradation while providing up to a 3× inference speedup on edge hardware [18], [19].

III. Benchmarking framework

This section details the benchmarking environment utilized to evaluate pillar-based 3D object detection on edge silicon. It defines the hardware constraints, the evaluation metrics, and the selected SOTA architectures.

A. Hardware platforms and configuration

Table I: Comparison of edge AI accelerators

Platform	**TOPS**	**Watts**	**TOPS/W**	**Price**
Jetson Nano	40	7–15	3–6	$199
Coral M.2	4	2	2	$25
Hailo-8	26	3–5	5–9	$180
Hailo-8L	13	2–3	4–7	$120

Table I compares representative GPU, TPU, and NPU architectures for edge DL acceleration based on peak performance and power consumption. While GPUs like the NVIDIA Jetson Orin Nano provide up to 40 Tera Operations per Second (TOPS), they operate within 7–15W power envelopes. In contrast, TPUs such as the Google Coral M.2 are limited to 4 TOPS at 2W and lack software support for modern 3D frameworks. The Hailo-8 series provides an intermediate profile, delivering 13 to 26 TOPS with a typical power consumption under 5W. Its integration with the Raspberry Pi 5 via expansion modules has already demonstrated high efficiency in 2D vision tasks [20] and is starting to be used for 3D perception with LiDAR sensors [21].

The experimental setup uses x86 host (Intel Core i9-13900, 32 GiB RAM) connected to a Hailo-8L NPU via PCIe. The host Central Processing Unit (CPU) executes the pre-processing and post-processing tasks, while the NPU accelerates the core network.

B. Dataset and metrics

The nuScenes dataset was selected for the development and evaluation of the 3D object detection models. It utilizes a Velodyne HDL-32E LiDAR sensor and features 1.4 million 3D bounding boxes across 1,000 scenes. While pioneers like KITTI use single-frame LiDAR scans with approximately 120k points, nuScenes employs a multi-sweep technique that accumulates up to 20 consecutive frames. This process densifies the point clouds to over 300k points per sample, providing temporal context at the cost of higher computational load. In edge hardware, this increased density directly impacts inference latency, as larger activation tensors must be managed within the NPU limited memory.

Detection accuracy is measured through:

- mean Average Precision (mAP): evaluates bounding box localization precision across different distance thresholds.
- nuScenes Detection Score (NDS): combined score of mAP and five True Positive (TP) error components: translation, scale, orientation, velocity, and attribute.

Hardware performance is assessed through:

- Inference Latency: the time required for a single forward pass through the NPU, measured in milliseconds.
- Effective Throughput: the total system performance measured in Frames Per Second (FPS).

C. Model selection

Architectures were selected based on the benchmarking of SOTA models on embedded NPU platforms, which demonstrates that efficient on-device inference requires architectures that avoid sparse convolutions [6]. Following this criterion, this study adopts the pillar-based paradigm, comparing PointPillars [11] as the pillar baseline against the advanced PillarNeSt model [12] to evaluate the impact of backbone scaling and structural optimizations on edge performance.

Both architectures share a tripartite structure (Encoder, Backbone, and Head) but differ in their technical implementation:

- **Pillar Encoder:** PointPillars utilizes a PointNet with Multi-Layer Perceptrons (MLPs) and max pooling. PillarNeSt combines max and average pooling with height offsets to mitigate Z-axis information loss. This modified encoder outputs a higher-resolution 2D pseudo-image, expanding the spatial dimensions for downstream processing.
- **2D Backbone:** PointPillars relies on standard 3×3 convolutions. In contrast, PillarNeSt adopts a ConvNeXt-based backbone, utilizing 7×7 kernels and increased depth in early stages to handle 3D data sparsity.
- **Post-processing:** Both models employ Non-Maximum Suppression (NMS) to filter redundant detections on the host CPU. However, their computational overhead differs. PointPillars relies on an anchor-based head, evaluating predefined bounding boxes at every grid location. This generates a large number of initial candidates, increasing the NMS load. Conversely, PillarNeSt utilizes an anchor-free CenterPoint head that predicts object centers via heatmaps, producing fewer candidate boxes and reducing the post-processing load.

PillarNeSt achieves scalability by varying the input channels (C_{in}) and the depth of its five stages. As shown in Table II, the computational load increases from the Tiny to the Large variant. To accurately represent the hardware workload, the reported parameters and Giga Floating Point Operations per Second (GFLOPs) correspond exclusively to the subnetwork mapped to the Hailo-8L. Parameter counts were extracted directly from the model framework and the computational complexity (GFLOPs) was empirically profiled using the *fvcore* library, evaluating the exact spatial tensor dimensions processed during inference.

Table II: Architectural comparison of PillarNeSt and PointPillars

Model	C_{in}	**Config**	**Params**	**GFLOPs**
PointPillars	64	—	4.9M	29.0
PN-Tiny	48	{2, 2, 1, 1, 1}	2.5M	107.8
PN-Small	48	{3, 3, 2, 1, 1}	5.0M	246.9
PN-Base	64	{4, 4, 2, 2, 1}	11.8M	419.1
PN-Large	96	{6, 6, 4, 2, 2}	15.1M	749.6

As shown in Table II, the hardware bottleneck of the PillarNeSt architecture does not lie in its parameter count, but in the computational demand (GFLOPs) of its activation tensors. By omitting downsampling in the initial stage to preserve spatial geometry, the encoder feeds the backbone with a high-resolution feature map (e.g., $720 \times 720 \times 48$ for the Tiny version). Executing 7×7 convolutions across this spatial grid increases the required operations. The processing workload is dictated by the spatial dimensions of the input tensor rather than the volume of trainable weights. Due to these resource constraints on the Hailo-8L, this study prioritizes the PillarNeSt-Tiny variant to provide a balanced framework for evaluating the accuracy-latency trade-off against the PointPillars baseline. To ensure reproducibility, the models are deployed utilizing the open-source MMDetection3D framework for PointPillars [22] and the official repository for PillarNeSt [23].

D. Hailo Dataflow Compiler

The Data Flow Compiler (DFC) translates neural network models into the Hailo Executable Format (HEF) through a fixed compilation pipeline. This process sequentially executes model parsing, graph optimization, fixed-point quantization, and hardware resource allocation. For the quantization stage, the DFC uses a linear Min-Max strategy, mapping FP32 activation ranges to INT8 discrete levels based on a calibration dataset.

To mitigate the precision loss inherent to INT8 conversion, the DFC integrates a model optimization tool with various PTQ algorithms. These include pre-quantization techniques, such as cross-layer equalization, to advanced post-quantization optimizations, including bias correction, activation clipping, Adaptive Rounding (Adaround), and Quantization-Aware Fine-Tuning (QFT). Based on current literature regarding LiDAR quantization [18], percentile-based clipping, QFT, and Adaround are the available DFC tools that best approximate the current strategies proposed to mitigate the severe precision degradation inherent to 3D point clouds.

- Quantization-Aware Fine-Tuning (QFT): the QFT methodology [24] bridges the gap between QAT and PTQ. Utilizing a small, unlabeled calibration subset and knowledge distillation from the FP32 teacher model, it performs a rapid fine-tuning of weights, biases, and scales.
- Adaptive Rounding (Adaround): instead of standard nearest-neighbor rounding, Adaround [25] optimizes weight rounding layer-by-layer. It learns whether to round each individual weight up or down to minimize the local reconstruction loss, providing a solution for network components that fail to converge under standard quantization.

IV. Proposed methodology

This section details the specific adaptations and optimization pipelines applied to deploy the selected 3D architectures on the Hailo-8L accelerator. The methodology focuses on overcoming the limitations of static NPU graphs and mitigating the precision loss inherent to PTQ.

A. Graph partitioning and architectural transformations

To ensure compatibility with the Hailo Dataflow architecture, the detection pipeline is partitioned between the host CPU and the NPU, as illustrated in Figure 1:

- CPU Execution: handles pre-processing and post-processing. Tasks with irregular memory access or dynamic shapes, such as Pillarization, Scattering, and NMS, remain on the host. This avoids hardware stalls when dealing with a variable number of object candidates.
- NPU Execution: executes the core network, consisting of the backbone, neck, and detection head, as a static computation graph. For PointPillars, the Hailo-8L processes a $1\times64\times400\times400$ pseudo-image and produces three output tensors for classification, bounding box regression, and direction. In the PillarNeSt-Tiny variant, the NPU ingests a $1\times48\times720\times720$ canvas and outputs flattened prediction tensors for heatmaps and bounding box attributes, including position, dimensions, rotation, velocity, and IoU.

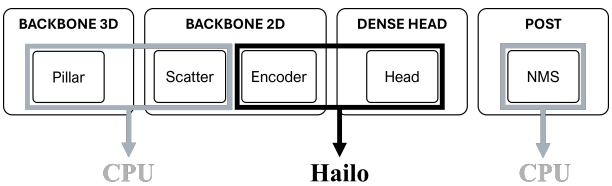


Fig. 1: Dataflow mapping and execution partitioning between the host CPU and the Hailo-8L NPU.

While the established PointPillars baseline [11] features a conventional 2D backbone that compiles natively, the advanced ConvNeXt-based design of PillarNeSt [12] requires structural modifications to eliminate compilation bottlenecks. The following graph modification is applied exclusively to the PillarNeSt models at the PyTorch level prior to ONNX exportation:

- **Linear Layer fusion:** Standard nn.Linear layers within the PillarNeSt backbone and encoder are replaced by equivalent 1×1 Conv2d operations. This transformation preserves spatial data routing and avoids flattening operations that are inefficient for the NPU dataflow.
- **LayerNorm adaptation:** The 1D LayerNorm layers native to ConvNeXt blocks are replaced by a custom 2D normalization module. This module manually computes spatial mean and variance, ensuring compatibility with the static memory buffers of the Hailo compiler.
- **Upsampling adaptation (PixelShuffle):** To replace resource-intensive ConvTranspose2d layers in the PillarNeSt neck, which often introduce zero-insertion overhead, a PixelShuffle (Depth-to-Space) strategy is implemented. For 2×2 upsampling, a 1×1 convolution expands the channel dimension by 4×, followed by deterministic re-ordering. For the 4×4 upsampling stages, a double PixelShuffle methodology is utilized, leveraging PixelUnshuffle to map pretrained weights into 1×1 convolutional kernels.

B. Optimization and quantization pipeline

Model quantization was performed using the Hailo DFC. Because 3D LiDAR models are highly sensitive to bit-width reduction, we evaluated available PTQ strategies to build a specialized 3D optimization pipeline. The compiler's framework prevents the simultaneous application of QFT and Adaround. Based on findings that global task-guided optimization is more effective than layer-wise reconstruction for 3D detection, we prioritized QFT and omitted Adaround [18]. To mitigate accuracy degradation, the following techniques were applied:

- **Densified dataset calibration:** to prevent the empty-frame bias typical of sparse nuScenes data, the calibration process was restricted to a curated, densified subset of 64 frames, ensuring each contained over 20 annotated objects.
- **Percentile-based clipping:** this technique was applied to truncate the long-tailed activation distributions, actively preventing spatial outliers from artificially expanding the INT8 dynamic ranges of the network.
- **QFT:** the compiler was configured to perform a rapid, joint fine-tuning of the quantization parameters, leveraging knowledge distillation from the FP32 teacher models to recover lost precision.

V. Results and discussion

This section analyzes the impact of quantization on 3D detection accuracy and hardware performance. Baseline sensitivity is evaluated for the architectures prior to the implementation of the Hailo 3D-specific optimization pipeline.

A. Quantization sensitivity analysis

The transition to INT8 utilizes the Hailo compiler's default Min-Max strategy. Table III illustrates the varying levels of accuracy loss observed across the evaluated architectures. PointPillars experiences a **20.9%** reduction in mAP and a **18.0%** drop in NDS. In contrast, PN-Tiny is more resilient, with its mAP decreasing by 11.9% and its NDS score by 6.3%.

Table III: Baseline Accuracy: FP32 vs. Min-Max INT8

Model	Metric	FP32	INT8	Drop
PointPillars	mAP	31.1	24.6	-6.5
	NDS	46.0	37.7	-8.3
PN-Tiny	mAP	58.8	51.8	-7.0
	NDS	65.6	61.5	-4.1

PointPillars is more sensitive to bit-width reduction than PillarNeSt. This is mainly due to its anchor-based head, where localization depends on specific coordinate offsets. In INT8, minor rounding errors in these offsets lead to 3D bounding box shifts. Furthermore, LiDAR point clouds are sparse, with up to 90% of the feature map consisting of zero pixels, creating long-tailed activation distributions.

Min-Max calibration is sensitive to spatial outliers, which can expand the dynamic range and reduce the precision available for foreground features.

PillarNeSt reduces this degradation through two main factors. Its ConvNeXt-based backbone uses normalization layers to stabilize variance, and its anchor-free head predicts probabilistic heatmaps. This heatmap-based approach is more robust to quantization noise than the direct coordinate regression used in PointPillars.

B. Optimized quantization performance

To recover the precision lost during the Min-Max calibration is applied the advanced PTQ strategies detailed in Section IV. Table IV compares the default pipeline against the optimized models.

Table IV: Impact of Advanced PTQ on Accuracy (%)

Model	Metric	FP32	Def.	Opt.	Drop
PointPillars	mAP	31.1	24.6	27.1	-4.0
	NDS	46.0	37.7	38.2	-7.8
PN-Tiny	mAP	58.8	51.8	54.3	-4.5
	NDS	65.6	61.5	63.1	-2.5

As illustrated in Fig. 2, the optimization pipeline achieves a +2.5% mAP gain for both architectures compared to the default INT8 baseline. While densified calibration and QFT mitigate the gap, FP32 accuracy is not fully recovered. This limitation arises because the strategies are constrained by the compiler optimization toolkit. Bridging the accuracy gap for 3D perception requires hardware-aware quantization at the PyTorch level prior to compilation, as suggested by recent literature [24], [10], [19].

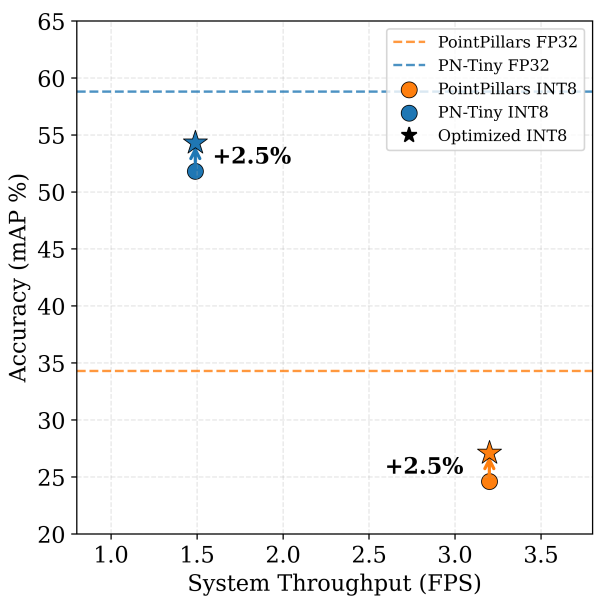


Fig. 2: Accuracy (mAP %) vs. Speed (FPS) trade-off highlighting the +2.5% improvement after INT8 optimization.

C. Hardware latency and bottlenecks

Deployment feasibility depends on end-to-end inference speed. Table V details the latency profile on the host and Hailo-8L setup. Measurements reflect a blocking execution pipeline, where pre-processing, NPU inference, and post-processing occur sequentially without concurrent pipelining.

Table V: End-to-End Latency (ms) and System Throughput (FPS)

Model	Pre	NPU	Post	Total	FPS
PointPillars	182.8	45.5	84.1	312.4	3.20
PN-Tiny	86.6	578.9	5.4	670.9	1.49

The distribution of computational load between the host CPU and the edge accelerator is visualized in Fig. 3. PointPillars is CPU-bound, with the host processor accounting for 266.9 ms (85.4%) of the total 312.4 ms cycle. This latency is driven by irregular point cloud voxelization and anchor-based NMS filtering, during which the NPU remains largely underutilized. In contrast, PillarNeSt-Tiny is NPU-bound. Its anchor-free design reduces combined pre- and post-processing to 92.0 ms, a 65.5% reduction compared to PointPillars. However, NPU inference represents 86.3% of the total latency, consuming 578.9 ms on the Hailo-8L, as the large activation tensors generated by the high-resolution input saturate the compute capacity. This bottleneck shift dictates future optimizations: PointPillars requires CPU-level multi-threading or offloading NMS, whereas PillarNeSt requires network compression to reduce the NPU workload.

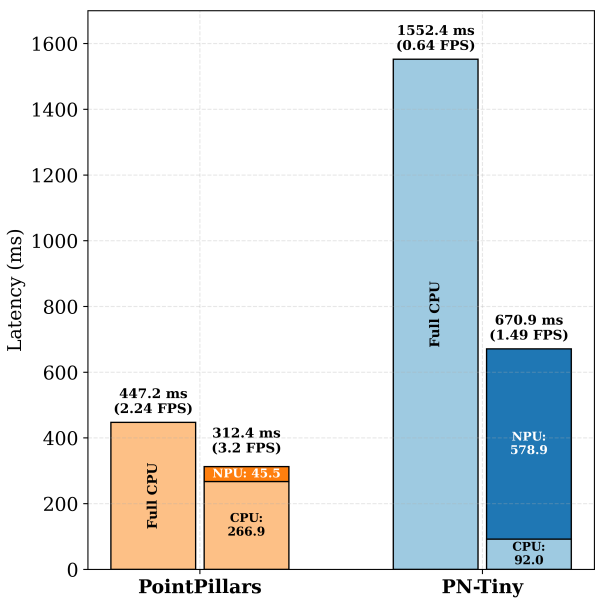


Fig. 3: End-to-end latency comparison by deployment strategy. The chart compares the total inference latency of PointPillars and PN-Tiny under two deployment configurations. The left bar for each model (Full CPU) represents the execution of the complete FP32 pipeline on the host processor. The right stacked bar illustrates the INT8 hybrid configuration, where network layers are offloaded to the Hailo-8L NPU and pre/post-processing tasks remain on the host CPU.

The implementation of the hybrid pipeline yields noticeable latency reductions compared to a full CPU execution. As shown in Fig. 3, hardware acceleration provides a 1.4× speedup for PointPillars (dropping total latency from 447.2 ms to 312.4 ms)

and a 2.3× increase for PillarNeSt-Tiny (dropping from 1552.4 ms to 670.9 ms). However, the limited speedup of Pointpillars indicates that hardware acceleration is constrained by the CPU-bound nature of the pipeline. In contrast, PN-Tiny benefits more from hardware acceleration due to its NPU-dominated workload.

The evaluation reveals a Pareto trade-off between accuracy and throughput. PointPillars maintains a higher throughput of 3.2 FPS but is limited to a maximum optimized mAP of 27.1%. Its performance is constrained by host-side sequential processing, making it suitable for systems with high-performance CPUs but limited NPU resources. PillarNeSt-Tiny exploits the NPU architecture to achieve a significantly higher mAP of 54.3% at a reduced throughput of 1.49 FPS. For IoT applications prioritizing high-fidelity perception, PillarNeSt emerges as a more suitable architecture. However, widespread deployment depends on accelerating NPU processing. Future work must investigate methods to reduce the computational cost of dense feature extraction, specifically through 4-bit compression or mixed-precision quantization. Both architectures require further research into PTQ and QAT alternatives beyond the Hailo DFC framework to achieve parity with floating-point performance.

VI. Conclusions and future work

This study demonstrates the feasibility of deploying SOTA 3D object perception models, such as the PillarNeSt architecture, on low-power accelerators like the Hailo-8L. The analysis reveals that architectural design determines the distribution of computational load and overall inference performance on heterogeneous systems. While PointPillars remains limited by CPU-bound operations, PillarNeSt successfully shifts the computational workload to the NPU.

Although the high-resolution activation tensors required by PillarNeSt demand significant computational resources, the architecture achieves a superior trade-off between detection accuracy and hardware efficiency. Specifically, the optimized INT8 model reaches a 54.3% mAP on the nuScenes dataset at 1.49 FPS. It is also concluded that effective deployment necessitates 3D-specific PTQ strategies to mitigate the accuracy degradation caused by the inherent sparsity of LiDAR point clouds.

While this work presents an initial evaluation using an x86 host, further research is required to achieve full embedded implementation. The following steps are proposed:

- **Embedded integration:** porting the complete pipeline from the x86 host to a Raspberry Pi 5 to evaluate real edge CPU bottlenecks, system latency, and thermal constraints under sustained IoT workloads.
- **Network optimization:** applying model compression and channel pruning to the PillarNeSt backbone to reduce intermediate tensor sizes and accelerate NPU execution.
- **Custom quantization:** investigating hardware-aware QAT at the PyTorch level to fully recover the FP32 accuracy lost during the PTQ process.
- **Real-World validation:** testing the quantized models using custom datasets collected directly from physical LiDAR sensors to assess domain shift and real-world generalization.

Acknowledgements

The publication is part of the INARTRANS 4.0 project PLEC2023-010343, funded by MICIU/AEI/10.13039/501100011033.

References

[1] Preeti and Chhavi Rana, "Review of Object Detection Techniques for Autonomous Vehicles: Trends and Future Prospects," in *2025 2nd International Conference On Multidisciplinary Research and Innovations in Engineering (MRIE)*, July 2025, pp. 47–51.

[2] Cristian Wisultschew, Gabriel Mujica, Jose Manuel Lanza-Gutierrez, and Jorge Portilla, "3D-LIDAR Based Object Detection and Tracking on the Edge of IoT for Railway Level Crossing," *IEEE Access*, vol. 9, pp. 35718–35729, 2021.

[3] Rogelio Hernandez, Gabriel Mujica, and Jorge Portilla, "Situational Awareness I2X Pipeline Methodology for LiDAR-based Sensor Networks at the Edge of Internet of Things," in *2024 IEEE International Conference on Evolving and Adaptive Intelligent Systems (EAIS)*, May 2024, pp. 1–8, ISSN: 2473-4691.

[4] Pooja Goswami, Ram Vaishnav, Tanishq Anand, and Prateek Dayal, "A Comprehensive Review on LiDAR Based 3D Deep Learning Object Detection Algorithms," in *2025 International Conference on Computer, Electrical & Communication Engineering (ICCECE)*, Feb. 2025, pp. 1–6, ISSN: 2768-0576.

[5] Cristian Wisultschew, Alejandro Pérez, Andrés Otero, Gabriel Mujica, and Jorge Portilla, "Characterizing Deep Neural Networks on Edge Computing Systems for Object Classification in 3D Point Clouds," *IEEE Sensors Journal*, vol. 22, no. 17, pp. 17075–17089, Sept. 2022.

[6] Shizhong Han, Hsin-Pai Cheng, Hong Cai, Jihad Masri, Soyeb Nagori, and Fatih Porikli, "FALO: Fast and Accurate LiDAR 3D Object Detection on Resource-Constrained Devices," 2025, Version Number: 1.

[7] "SECOND: Sparsely Embedded Convolutional Detection," .

[8] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li, "PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection," Apr. 2021, arXiv:1912.13192 [cs].

[9] Yukang Chen, Jianhui Liu, Xiangyu Zhang, Xiaojuan Qi, and Jiaya Jia, "VoxelNeXt: Fully Sparse VoxelNet for 3D Object Detection and Tracking," Mar. 2023, arXiv:2303.11301 [cs].

[10] Ninnart Fuengfusin, Keisuke Yoneda, and Naoki Suganuma, "Mixed Precision PointPillars for Efficient 3D Object Detection with TensorRT," Feb. 2026, arXiv:2601.12638 [cs].

[11] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom, "PointPillars: Fast Encoders for Object Detection from Point Clouds," May 2019, arXiv:1812.05784 [cs].

[12] Weixin Mao, Tiancai Wang, Diankun Zhang, Junjie Yan, and Osamu Yoshie, "PillarNeSt: Embracing Backbone Scaling and Pretraining for Pillar-based 3D Object Detection," Nov. 2023, arXiv:2311.17770 [cs].

[13] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen, "Deep Learning for Generic Object Detection: A Survey," *International Journal of Computer Vision*, vol. 128, no. 2, pp. 261–318, Oct. 2019.

[14] Paschalis Tsirtsakis, Georgios Zacharis, George S. Maraslidis, and George F. Fragulis, "Deep learning for object recognition: A comprehensive review of models and algorithms," *International Journal of Cognitive*

Computing in Engineering, vol. 6, pp. 298–312, Dec. 2025.

[15] Xiang Zhang, Hai Wang, and Haoran Dong, "A Survey of Deep Learning-Driven 3D Object Detection: Sensor Modalities, Technical Architectures, and Applications," *Sensors*, vol. 25, no. 12, pp. 3668, June 2025.

[16] Yuhao Huang, Sanping Zhou, Junjie Zhang, Jinpeng Dong, and Nanning Zheng, "Voxel or Pillar: Exploring Efficient Point Cloud Representation for 3D Object Detection," Mar. 2024, arXiv:2304.02867 [cs].

[17] Youngpyo Hong and Dongsoo Kim, "Performance and Efficiency Gains of NPU-Based Servers over GPUs for AI Model Inference," *Systems*, vol. 13, no. 9, pp. 797, Sept. 2025.

[18] Sifan Zhou, Liang Li, Xinyu Zhang, Bo Zhang, Shipeng Bai, Miao Sun, Ziyu Zhao, Xiaobo Lu, and Xiangxiang Chu, "LiDAR-PTQ: Post-Training Quantization for Point Cloud 3D Object Detection," Jan. 2024, arXiv:2401.15865 [cs].

[19] Yumi Kim, Kyungho Kim, Dongyeob Shin, Sung-Joon Jang, and Sang-Seol Lee, "Fully Integer Post-Training Quantization in Lidar-Based 3D Object Detection for Efficient Hardware Acceleration," in *2025 25th International Conference on Control, Automation and Systems (ICCAS)*, Nov. 2025, pp. 1769–1773, ISSN: 2642-3901.

[20] Rahima Khanam and Muhammad Hussain, "YOLOv11: An Overview of the Key Architectural Enhancements," Oct. 2024, arXiv:2410.17725 [cs].

[21] Itay Krispin-Avraham, Roy Orfaig, and Ben-Zion Bobrovsky, "Real-Time 3D Object Detection Using InnovizOne LiDAR and Low-Power Hailo-8 AI Accelerator," Dec. 2024, arXiv:2412.05594 [cs].

[22] MMDetection3D Contributors, "OpenMMLab's Next-generation Platform for General 3D Object Detection," July 2020, original-date: 2020-07-08T03:39:45Z.

[23] Wayne Mao, "WayneMao/PillarNeSt," Dec. 2025, original-date: 2024-01-24T12:43:59Z.

[24] Alex Finkelstein, Ella Fuchs, Idan Tal, Mark Grobman, Niv Vosco, and Eldad Meller, "QFT: Post-training Quantization via Fast Joint Finetuning of All Degrees of Freedom," in *Computer Vision – ECCV 2022 Workshops*, Leonid Karlinsky, Tomer Michaeli, and Ko Nishino, Eds., vol. 13807, pp. 115–129. Springer Nature Switzerland, Cham, 2023, Series Title: Lecture Notes in Computer Science.

[25] Markus Nagel and Rana Ali Amjad, "Up or Down? Adaptive Rounding for Post-Training Quantization," .

Estudio del impacto de políticas de planificación en un robot de análisis clínico basado en un planificador dinámico

Plácido Fernández-Cuevas, Juan M. Montes-Sánchez, Carlos Rangel-Cascajosa, Ángel Jiménez-Fernández, Saturnino Vicente-Díaz, y Antón Civit-Balcels [1] [2] [3] [4] [5] [6]

Resumen— **Los sistemas que utilizan un planificador suelen estar compuestos por tareas con una duración específica, donde el planificador es responsable de determinar qué tarea debe ejecutarse en cada momento para que los resultados de las mismas sigan siendo válidos una vez finalizadas. La optimización de estos sistemas es una labor fundamental, ya que existen diversas vías para buscar dicha mejora, como la reducción de la duración de las propias tareas, el aumento de la potencia de procesamiento o la modificación de las políticas de planificación para lograr un mejor rendimiento.**

Este trabajo busca estudiar el impacto de la optimización de las políticas de planificación en un planificador implementado en un robot de análisis biomédico. Para llevar a cabo el estudio de la forma más cercana posible a la realidad, se utiliza un gemelo digital digital twin del robot, lo que permite obtener información precisa y fiable de los resultados.

Hemos descubierto que el impacto de las políticas es bastante significativo y varía en función de la combinación de muestras que el robot debe analizar, logrando una reducción máxima de tiempo de hasta 30 minutos en un análisis de 4.5 horas. Por otro lado, las políticas también pueden empeorar los resultados, incrementando el tiempo hasta en 36 minutos en un análisis de 4.5 horas.

Palabras clave— **Políticas de planificación, Planificación dinámica, Robots biomédicos, Gemelo digital, Optimización, Simulación.**

I. Introducción

El cribado de alto rendimiento y la eficiencia son primordiales en los laboratorios de análisis biomédicos modernos. La automatización de protocolos analíticos complejos se ha vuelto esencial para reducir el error humano, aumentar la reproducibilidad y gestionar la creciente demanda de procesamiento de muestras [1][2][3]. Estos sistemas automatizados emplean plataformas robóticas complejas para realizar tareas sensibles al tiempo, como la dispensación de reactivos, el lavado de muestras y la incubación de las mismas [4].

El desafío principal en estos sistemas robóticos radica en la gestión eficiente de procesos concurrentes y recursos compartidos (por ejemplo, el brazo del robot, las sondas de dispensación) [5]. Un planificador es responsable de orquestar estas tareas, asegurando que todos los pasos analíticos se realicen en el orden correcto y, fundamentalmente, que se cumplan las estrictas restricciones temporales. El incumplimiento de estos plazos, especialmente en los procesos de incubación, puede comprometer la reacción química e invalidar todo el análisis. Esto transforma el problema en un desafío de planificación complejo en tiempo real, que a menudo puede aproximarse a una complejidad NP-completo [6].

Muchos sistemas existentes dependen de planificadores estáticos, los cuales están altamente optimizados para un protocolo específico pero carecen de la flexibilidad para gestionar tareas heterogéneas de forma dinámica [7]. Para abordar esta limitación, propusimos anteriormente un algoritmo de planificación genérico y adaptable diseñado para tomar decisiones de orden dinámicas para equipos de análisis clínico automatizados [8]. Ese trabajo introdujo el marco conceptual para un planificador dinámico basado en prioridades capaz de gestionar múltiples protocolos diferentes de forma concurrente.

Mientras que [8] introdujo el algoritmo fundamental, este trabajo presenta su implementación práctica y un estudio detallado del rendimiento basado en un robot biomédico del mundo real. Para realizar este estudio de manera fiable, repetible y no disruptiva, desarrollamos un modelo de simulación de alta fidelidad del robot y su planificador, a menudo denominado Gemelo Digital (DT). Este gemelo digital nos permite modelar con precisión el comportamiento del sistema y probar el impacto de diferentes configuraciones, una metodología que ha demostrado ser muy eficaz en la robótica industrial [9].

El objetivo principal de este trabajo es aprovechar este gemelo digital para realizar un análisis *what-if* centrado en un componente crítico, aunque a menudo pasado por alto: la política de planificación. Una política de planificación es el conjunto de reglas que utiliza el planificador para decidir "qué tarea ejecutar a continuación"de entre todas las opciones disponibles. Nuestra hipótesis es que esta lógica de decisión

[1]P. Fernández-Cuevas pertenece al Dpto. Arquitectura y Tecnolgía de Computadores, EPS / ETSII, Universidad de Sevilla, España (e-mail: pfernandez9@us.es).

[2]J. M. Montes-Sánchez pertenece al Dpto. Arquitectura y Tecnolgía de Computadores, EPS / ETSII, y SCORE lab, Universidad de Sevilla, Spain (e-mail: juanmanuelmontes@us.es).

[3]C. Rangel-Cascajosa pertenece al Dpto. Arquitectura y Tecnolgía de Computadores, EPS / ETSII, Universidad de Sevilla, España (e-mail: crangel@us.es)

[4]A. Jiménez-Fernández pertenece al Dpto. Arquitectura y Tecnolgía de Computadores, EPS / ETSII, y SCORE lab, Universidad de Sevilla, España (e-mail: angel@us.es).

[5]S. Vicente-Díaz pertenece al Dpto. Arquitectura y Tecnolgía de Computadores, EPS / ETSII, Universidad de Sevilla, España (e-mail: satur@us.es).

[6]A. Civit-Balcels pertenece al Dpto. Arquitectura y Tecnolgía de Computadores, EPS / ETSII, Universidad de Sevilla, España (e-mail: civit@us.es).

a nivel de software tiene un impacto cuantitativo en el rendimiento tan significativo como las modificaciones de hardware.

Utilizamos este modelo para estudiar el impacto de diferentes políticas de planificación en el makespan total del sistema (tiempo total de análisis). Demostramos que el impacto es muy significativo y varía según la combinación específica de muestras que se analizan. Nuestros resultados muestran que una política optimizada puede reducir el tiempo total hasta en 30 minutos en un análisis de 4,5 horas, mientras que una política mal adaptada puede empeorarlo en hasta 36 minutos, confirmando que el ajuste de la política es un vector de optimización crítico.

El resto de este capítulo se estructura de la siguiente manera: La Sección II describe brevemente el sistema biomédico y su gemelo digital. La Sección III define los escenarios experimentales y las políticas de planificación bajo estudio. La Sección IV presenta y analiza los resultados comparativos. Finalmente, la Sección V concluye el trabajo.

II. Materiales y Métodos

A. La Plataforma de Análisis Robótico

La plataforma de hardware utilizada para este estudio es un robot de análisis biomédico comercial diseñado para automatizar completamente protocolos de tinción complejos en muestras de tejido. El sistema está destinado al uso en laboratorios clínicos, donde la robustez y el procesamiento concurrente son críticos.

El robot funciona mediante cinemática cartesiana (ejes X, Y, Z) y cuenta con una amplia área de trabajo equipada con **42 cámaras de reacción independientes y calefactables**. Este diseño permite el procesamiento simultáneo de hasta 42 muestras diferentes, cada una de las cuales puede ejecutar un protocolo analítico distinto. El efector final (brazo del robot) está equipado con un carro multi-herramienta que incluye:

- Una sonda de dispensación de alta precisión (25 μL) para reactivos críticos.
- Un conjunto de bombas peristálticas para reactivos a granel (p. ej., tampones, soluciones de lavado).
- Una sonda de alto flujo para el lavado de muestras.
- Un soplador de aire para el secado de muestras.
- Una herramienta extensora para asegurar la distribución homogénea del reactivo.
- Una estación dedicada para el lavado de la sonda y el extensor para evitar la contaminación cruzada.

B. Arquitectura del Planificador

Desde la perspectiva de la planificación, todas las tareas (procesos) realizados por el robot pueden categorizarse en dos tipos fundamentales, lo cual es crítico para la optimización:

1. **Procesos Bloqueantes:** Acciones que requieren el uso exclusivo del brazo principal y las herramientas del robot (p. ej., dispensación, lavado, movimiento, extensión). Debido a las restricciones físicas del hardware, solo se puede ejecutar un proceso bloqueante a la vez.
2. **Procesos No Bloqueantes:** Acciones que no utilizan el brazo del robot. Esta categoría consiste principalmente en incubaciones. Una incubación es un periodo de espera crítico donde una muestra debe permanecer sin perturbaciones durante una duración precisa para permitir una reacción química. Múltiples procesos no bloqueantes (p. ej., 42 incubaciones simultáneas) pueden ejecutarse en paralelo.

El desafío central para el planificador es entrelazar eficientemente los procesos bloqueantes necesarios (que forman una cola de espera serial) durante el "tiempo muerto"proporcionado por las múltiples incubaciones paralelas no bloqueantes, todo ello asegurando que no se infrinja ninguna restricción temporal.

Como se propuso en nuestro trabajo anterior [8], el sistema es gestionado por un planificador dinámico basado en prioridades. Esta arquitectura (visualizada en el diagrama de clases de la Fig. 1) define una jerarquía clara orientada a objetos.

Los procesos siguen un ciclo de vida definido por una máquina de estados, como se muestra en la Fig. 2. Un proceso se crea en el estado `NOT_READY`. Solo transiciona al estado `READY` cuando su proceso precedente en la secuencia del protocolo ha alcanzado el estado `FINISHED`.

La función principal del planificador se ejecuta en un bucle continuo: escanea la lista de todos los procesos de las 42 muestras que se encuentran actualmente en el estado `READY`. De esta lista de candidatos, utiliza las **políticas de planificación** activas (el foco central de este estudio) para seleccionar el "mejor"proceso individual para pasarlo al estado `EXEC`.

C. Gemelo Digital

Para estudiar el impacto de estas políticas sin consumir reactivos físicos ni ocupar el robot real, desarrollamos y validamos un gemelo digital del sistema. Este gemelo digital es un modelo de simulación de eventos discretos de alta fidelidad que replica con precisión:

1. La cinemática del robot y la duración exacta, previamente calculada, de cada proceso bloqueante.
2. La arquitectura completa del planificador (Fig. 1) y su lógica de máquina de estados (Fig. 2).
3. La compleja lógica temporal de todos los procesos, especialmente las ventanas críticas de tiempo de inicio y fin para las incubaciones.

Este modelo de simulación ha sido validado frente a la plataforma robótica del mundo real, demostrando una capacidad para predecir los tiempos totales de ejecución con una desviación media de menos

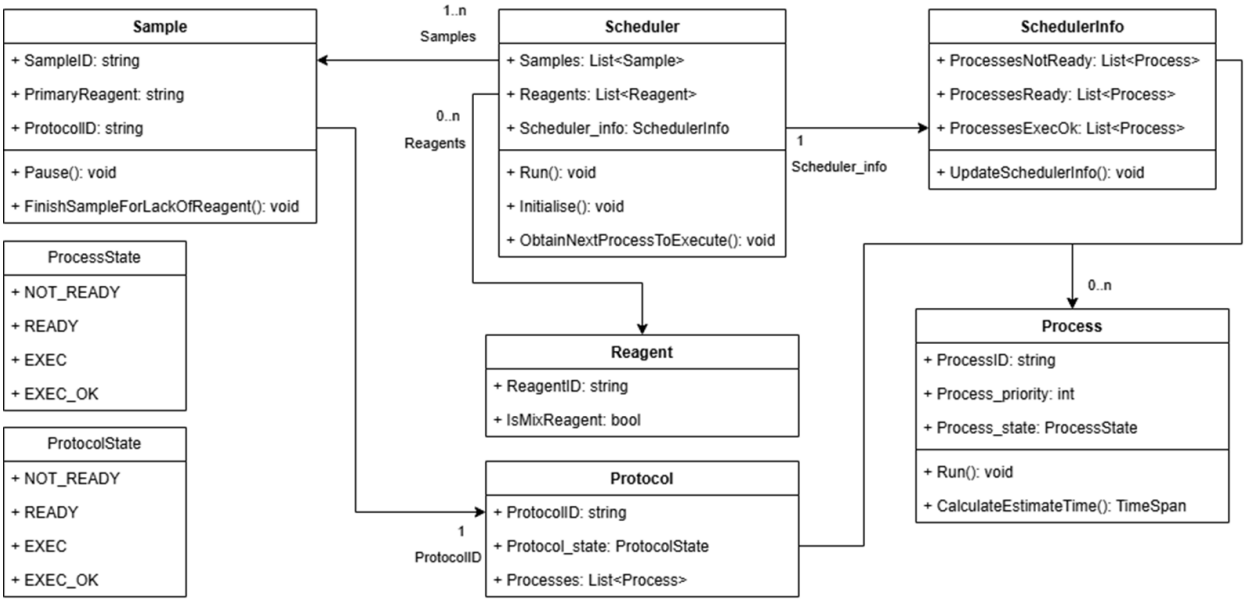


Fig. 1: Diagrama de clases UML de la arquitectura del planificador, mostrando la relación entre el Planificador, las Muestras, los Protocolos y los Procesos.

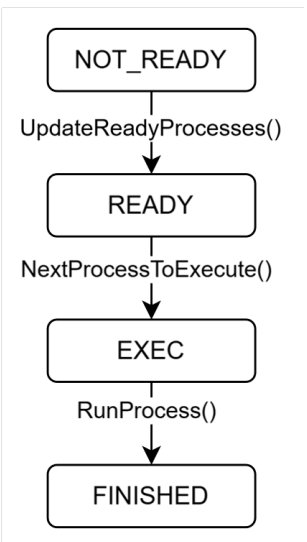


Fig. 2: Ciclo de vida de la máquina de estados para un único proceso (tarea) dentro del planificador.

del 2,5 % (y una desviación máxima observada del 2,41 %). Este modelo validado de alta fidelidad proporciona la base fiable para el análisis *what-if* de las políticas de planificación presentadas.

III. Políticas de Planificación y Diseño Experimental

Para cuantificar el impacto de la lógica de decisión del planificador, diseñamos un estudio comparativo exhaustivo. El estudio ejecuta nueve escenarios de carga de trabajo diferentes (S1-S9) frente a ocho políticas de planificación distintas (P0-P7). Todas las pruebas se realizaron utilizando el gemelo digital validado descrito en la Sección II.

A. Definición de las Políticas de Planificación

Una política de planificación es la lista jerárquica de reglas que utiliza el planificador para ordenar la cola de procesos en estado `READY` y seleccionar la siguiente tarea a ejecutar. Las ocho políticas evaluadas en este estudio utilizan la **Prioridad de Proceso (P-Prio)** como mecanismo de ordenación primario (Regla 1). Se trata de una prioridad dinámica definida en el proceso, que garantiza que las tareas críticas en el tiempo (p. ej., atender una incubación) siempre ocupen el rango más alto.

La diferencia entre las políticas radica en la secuencia de reglas de desempate (Regla 2, Regla 3, etc.) que se aplican cuando dos o más procesos comparten la misma Prioridad de Proceso.

Las heurísticas clave probadas como desempate son:

- **Mayor Espera (P-Wait):** El desempate base. Prioriza la tarea que ha estado en estado `READY` durante más tiempo.
- **Número de Paso (P-Step):** Un desempate alternativo que prioriza las tareas con un número de paso menor (es decir, aquellas que están al principio de la secuencia del protocolo).
- **Prioridad de Lavado (P-Wash):** Una regla secundaria que, ante un nivel igual de `P-Prio`, da precedencia a cualquier proceso `WashSlide`. Esta heurística está diseñada para liberar las cámaras de reacción más rápidamente.
- **Incubación Remanente más Larga (P-Inc):** Una heurística que prioriza las tareas pertenecientes a un protocolo que tiene próximamente un proceso de incubación largo.

La Tabla I enumera las ocho políticas definidas por estas combinaciones de reglas jerárquicas. **P0 (Base)** representa la configuración estándar, mientras que P1-P7 introducen diferentes lógicas de desempate.

ID de Política	Reglas Jerárquicas (Aplicadas en orden)
P0	1. P-Prio → 2. P-Wait
P1	1. P-Prio → 2. P-Step
P2	1. P-Prio → 2. P-Inc → 3. P-Wait
P3	1. P-Prio → 2. P-Inc → 3. P-Step
P4	1. P-Prio → 2. P-Wash → 3. P-Wait
P5	1. P-Prio → 2. P-Wash → 3. P-Step
P6	1. P-Prio → 2. P-Wash → 3. P-Inc → 4. P-Wait
P7	1. P-Prio → 2. P-Wash → 3. P-Inc → 4. P-Step

Tabla I: Conjunto de las ocho políticas de planificación jerárquicas evaluadas en el estudio. Todas las políticas utilizan la Prioridad de Proceso como Regla 1.

IV. Resultados Experimentales

El diseño experimental generó datos para 72 ejecuciones distintas (8 políticas × 9 escenarios). Presentamos los resultados en dos tablas para mayor claridad: la Tabla II muestra las métricas de rendimiento para escenarios homogéneos representativos, mientras que la Tabla III detalla los resultados de los escenarios complejos y heterogéneos (carga mixta).

A. Resultados de Rendimiento

Las métricas principales son el Makespan (t) (en 'HH:MM:SS') y el Total de Lavados de Jeringa (nSW). La desviación, Dif t (también en 'HH:MM:SS'), se calcula con respecto a la política **P0 (Base)** para cada escenario específico.

B. Análisis de Resultados

Los datos de las Tablas II y III revelan varios hallazgos críticos, confirmando la hipótesis de que la elección de la política tiene un impacto significativo y volátil en el rendimiento.

- **Volatilidad extrema en escenarios heterogéneos.** El hallazgo más dramático, mostrado en la **Tabla III**, es el fallo catastrófico de rendimiento de la política **P1**. En la compleja carga de trabajo **S8 (HRP+CISH)**, P1 añadió **+37 minutos y 34 segundos** al makespan. Esto demuestra que la heurística 'P-Step', aparentemente simple, puede ser destructiva, probablemente al dejar sin recursos tareas críticas y provocar cuellos de botella en cascada en un entorno mixto.
- **Ganancias significativas de las políticas orientadas a la incubación.** Por el contrario, la **Tabla III** muestra un inmenso potencial de optimización. Las políticas con mejor rendimiento fueron consistentemente **P2** y **P3**. En el **Escenario S8**, la política **P2** logró la mayor reducción de tiempo del estudio, **ahorrando 26 minutos y 51 segundos**. Esto crea una oscilación total de rendimiento de más de 64 minutos entre la mejor (P2) y la peor (P1) política, **como se visualiza en la figura 3**. El éxito de P2 y P3 indica que utilizar la **Incubación Remanente más Larga (P-Inc)** como principal desempate es una estrategia altamente efectiva para maximizar el paralelismo del sistema.
- **La heurística de "Prioridad de Lavado"(P4-P7) es subóptima.** Un hallazgo contraintuitivo se refiere a las políticas (P4-P7) que priorizaban explícitamente los procesos `WashSlide` (ver **Tabla III**). La hipótesis era que liberar las cámaras más rápido (P-Wash) mejoraría el rendimiento. Los datos no respaldan esto. En todos los escenarios heterogéneos (S7, S8, S9), las políticas 'P-Wash' fueron superadas por las políticas 'P-Inc' (P2, P3).
- **Escenarios homogéneos.** Como se muestra en la **Tabla II**, las diferencias de rendimiento fueron menos extremas en los escenarios homogéneos simples, como era de esperar. Sin embargo, la **política P0 (Base) nunca fue la opción óptima**. En S2, por ejemplo, la política P2 redujo el tiempo en más de 14 minutos, lo que indica que la lógica base 'P-Wait' es subóptima incluso para cargas de trabajo sencillas.
- **Análisis del número de lavados (nSW).** La métrica 'nSW' (mostrada en ambas tablas) proporciona una explicación clara de los cambios en el rendimiento. En el crítico **Escenario S8 (Tabla III)**, la base (P0) requirió 37 lavados. Las políticas con mejor rendimiento, **P2** y **P3**, **redujeron drásticamente el número de lavados a 28 y 26**, respectivamente. Esto sugiere fuertemente que su lógica 'P-Inc' crea una planificación más eficiente que agrupa las tareas de manera efectiva. Por el contrario, las políticas 'P-Wash' (P4-P7) no lograron reducir el número de lavados (38 lavados), reforzando la conclusión de que no son una heurística eficaz. El número total de lavados para el Escenario S8 está **representado visualmente en la figura 4**.

V. Conclusiones

Este artículo ha presentado un estudio sobre el impacto de diferentes políticas de planificación en el rendimiento de un planificador dinámico y colaborativo aplicado a un robot de análisis biomédico. Utilizando un gemelo digital de alta fidelidad validado, realizamos un estudio experimental exhaustivo,

	S2 (42 HRP)			S5 (42 AP)		
ID de Política	**Tiempo (t)**	**nSW**	**Dif t**	**Tiempo (t)**	**nSW**	**Dif t**
P0	04:07:57	23	–	04:30:14	7	–
P1	03:55:34	16	-12:23	04:32:41	7	+02:27
P2	03:53:44	16	**-14:13**	04:28:10	5	-02:04
P3	03:55:34	16	-12:23	04:32:41	7	+02:27
P4	04:06:13	20	-01:44	04:28:10	5	-02:04
P5	04:06:51	20	-01:06	04:32:41	7	+02:27
P6	04:02:22	20	-05:35	04:28:10	5	-02:04
P7	04:02:46	20	-05:11	04:32:41	7	+02:27

Tabla II: Resultados comparativos para Escenarios Homogéneos (S2, S5).

	S7 (HRP+AP)			S8 (HRP+CISH)			S9 (HRP+AP+CISH)		
ID de Política	**Tiempo (t)**	**nSW**	**Dif t**	**Tiempo (t)**	**nSW**	**Dif t**	**Tiempo (t)**	**nSW**	**Dif t**
P0	04:52:54	38	–	04:43:13	37	–	05:39:49	41	–
P1	04:54:38	33	+01:44	05:20:47	33	**+37:34**	06:11:28	38	**+31:39**
P2	04:41:30	31	**-11:24**	04:16:22	28	**-26:51**	05:28:19	40	-11:30
P3	04:45:22	31	-07:32	04:16:28	26	-26:45	05:27:40	40	**-12:09**
P4	04:50:25	29	-02:29	04:41:39	38	-01:34	05:35:48	43	-04:01
P5	04:52:21	30	-00:33	04:47:03	38	+03:50	05:34:57	43	-04:52
P6	04:46:13	30	-06:41	04:21:40	38	-21:33	05:28:50	43	-10:59
P7	04:48:47	30	-04:07	04:20:01	38	-23:12	05:28:28	43	-11:21

Tabla III: Resultados comparativos para Escenarios Heterogéneos (Mixtos) (S7, S8, S9). Se resaltan las desviaciones más extremas.

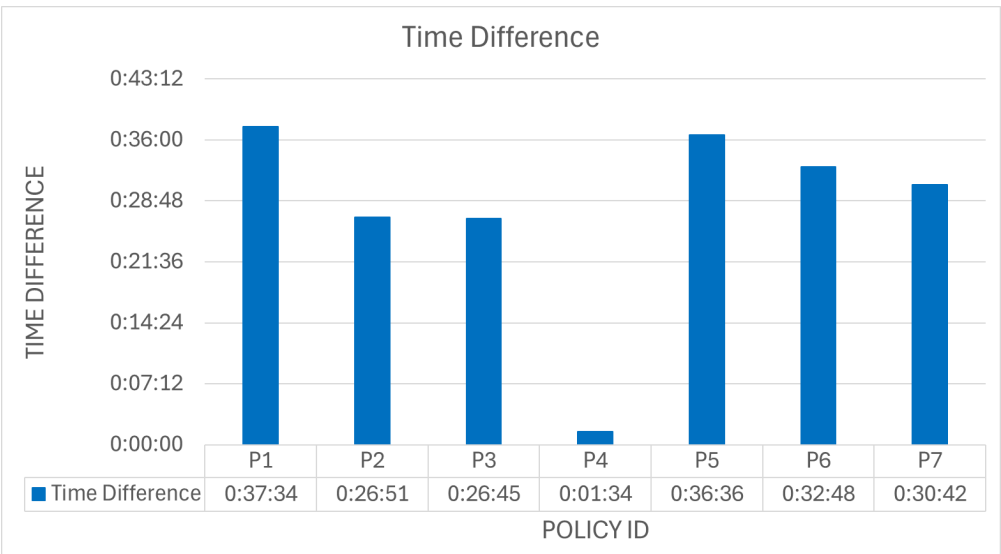


Fig. 3: Comparación visual de la desviación del makespan (en minutos) para el **Escenario S8 (HRP+CISH)**. El gráfico muestra claramente el fallo de +37 min de P1 y la optimización de -27 min de P2, demostrando el impacto crítico de la elección de la política.

probando ocho políticas de planificación distintas en nueve escenarios de carga de trabajo heterogéneos y homogéneos.

Nuestros resultados demuestran que la elección de la política de planificación es un vector de optimización crítico y de alto impacto, no un detalle menor de implementación. El rendimiento del sistema es altamente volátil y depende directamente de la lógica de planificación utilizada.

Esto fue más evidente en las cargas de trabajo complejas y heterogéneas. En el Escenario S8 (HRP+CISH), observamos una oscilación total del rendimiento de más de 64 minutos para la misma carga de trabajo, simplemente cambiando las reglas de desempate del planificador. Una heurística mal elegida (Política P1, basada en el "Número de Paso") resultó en una degradación catastrófica del rendimiento, añadiendo más de 37 minutos a la base. Por el contrario, una heurística optimizada (Política P2, priorizando el "Tiempo de Incubación") logró

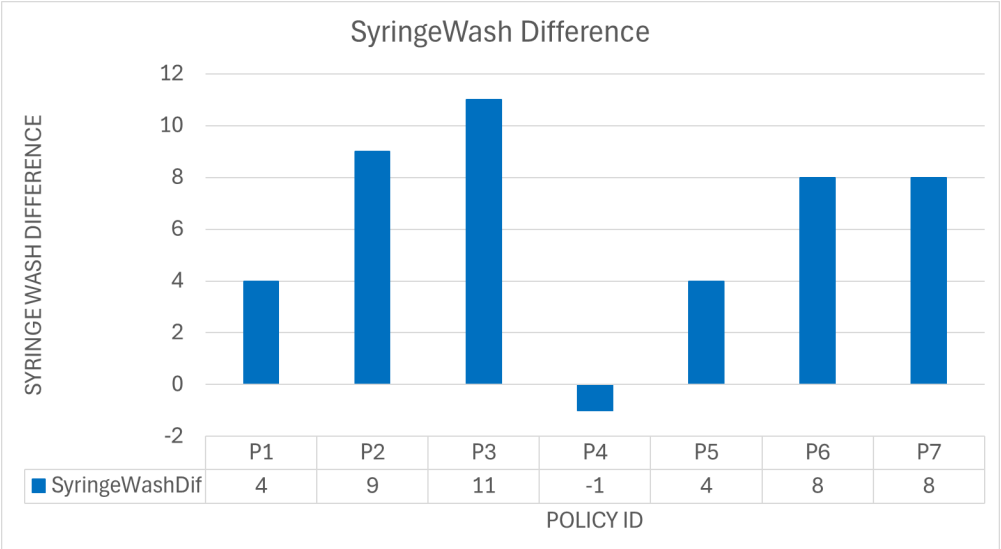


Fig. 4: Número total de lavados (nSW) para el **Escenario S8 (HRP+CISH)** a través de las diferentes políticas de planificación. Esta figura destaca cómo las políticas eficientes como P2 y P3 logran reducciones significativas en los ciclos de lavado en comparación con la base y otras políticas.

una reducción masiva de 27 minutos en el makespan total.

También encontramos una fuerte correlación entre el makespan y la eficiencia operativa, medida por el número total de lavados (nSW). Las políticas más exitosas (P2, P3) lograron sus ahorros de tiempo mediante la creación de una planificación más inteligente que agrupaba las tareas de manera efectiva, reduciendo el número total de lavados hasta en un 30 % (de 37 a 26) en el escenario S8. De forma contraintuitiva, las políticas que priorizaban explícitamente el lavado (P4-P7) no lograron alcanzar esta eficiencia y resultaron subóptimas.

La conclusión clave es que, para sistemas robóticos complejos y multiprotocolo, la política de planificación es un factor crítico para el rendimiento (throughput). Confiar en una política "base"simple (como P0) es subóptimo. Las mayores ganancias de rendimiento se encuentran al utilizar heurísticas (como P-Inc) que gestionan inteligentemente la restricción principal del sistema: la ejecución paralela de incubaciones no bloqueantes.

Este trabajo abre tres vías significativas para la investigación futura:

1. **Validación Física:** El siguiente paso inmediato es desplegar la política más exitosa (P2 o P3) en la plataforma robótica física. Esto proporcionará una validación final en el mundo real de los hallazgos del gemelo digital y confirmará el ahorro de tiempo en un entorno de laboratorio clínico.
2. **Exploración de Nuevas Heurísticas:** Este estudio se limitó a combinaciones de cuatro heurísticas primarias. El trabajo futuro debería centrarse en el descubrimiento y análisis de políticas de planificación completamente nuevas. Esto podría incluir heurísticas más avanzadas, como reglas basadas en minimizar el tiempo de desplazamiento del brazo robótico, priorizar tareas en la ruta crítica del protocolo o agrupar tareas por tipo de reactivo para reducir aún más la contaminación cruzada y los ciclos de lavado.
3. **Selección Dinámica de Políticas:** Nuestros resultados demuestran que la política óptima depende de la carga de trabajo (por ejemplo, P2 destaca en S8, pero no necesariamente en S2). El objetivo final es crear un **planificador adaptativo** que seleccione o construya dinámicamente la mejor política en tiempo real. En lugar de estar fijado a una sola política (como P2), el planificador podría analizar el lote de muestras entrante (por ejemplo, "mezcla alta"frente a "homogénea") y seleccionar el conjunto óptimo de reglas jerárquicas (por ejemplo, P-Inc + P-Wait) para ese trabajo específico. Esta es una aplicación prometedora para el aprendizaje automático, donde un agente de aprendizaje por refuerzo podría ser entrenado para seleccionar la mejor heurística en cada punto de decisión para minimizar el makespan global.

Agradecimientos

Agradecimientos

Este trabajo ha sido financiado por el Proyecto PREDICAR (ref. PID2023-149777OB-I00) de la Agencia Estatal de Investigación, Gobierno de España.

Este proyecto ha recibido financiación de la empresa Vitro S.A. como proyecto de innovación y desarrollo con referencia P022-21/E03 "Generic Planning Algorithm Adaptable to Automated Clinical Analysis Equipment (VitroPlanner)".

Referencias

[1] Richard F Venn, *Principles and practice of bioanalysis*, CRC Press, 2008.

[2] Jose A Ramos-Vara, "Technical aspects of immunohistochemistry," *Veterinary pathology*, vol. 42, no. 4, pp. 405–426, 2005.

[3] Ellen Jensen, "Technical review: In situ hybridization," *The Anatomical Record*, vol. 297, no. 8, pp. 1349–1353, 2014.

[4] Robin A Felder, "The Clinical Chemist: Masahide Sasaki, MD, PhD (August 27, 1933–September 23, 2005)," *Clinical Chemistry*, vol. 52, no. 4, pp. 791–792, 04 2006.

[5] Esther Bischoff, Fabian Meyer, Jairo Inga, and Sören Hohmann, "Multi-robot task allocation and scheduling considering cooperative tasks and precedence constraints," in *2020 IEEE international conference on systems, man, and cybernetics (SMC)*. IEEE, 2020, pp. 3949–3956.

[6] Jan Karel Lenstra and AHG Rinnooy Kan, "Complexity of scheduling under precedence constraints," *Operations Research*, vol. 26, no. 1, pp. 22–35, 1978.

[7] Khushboo Singh, Mahfooz Alam, and Sushil Kumar Sharma, "A survey of static scheduling algorithm for distributed computing system," *International Journal of Computer Applications*, vol. 129, no. 2, pp. 25–30, 2015.

[8] Plácido Fernández-Cuevas, Saturnino Vicente-Diaz, and Antón Civit-Balcels, "Generic planning algorithm adaptable for order decision to automated clinical analysis equipment," in *Decision Science Alliance International Summer Conference*. Springer, 2024, pp. 194–200.

[9] Fei Tao, Meng Zhang, Ang Liu, and Andrew YC Nee, "Digital twin in industry: State-of-the-art," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2405–2415, 2019.

[10] Hegen Xiong, Shuangyuan Shi, Danni Ren, and Jinjin Hu, "A survey of job shop scheduling problem: The types and models," *Computers & Operations Research*, vol. 142, pp. 105731, 2022.

[11] Rafal Szczepanski, Krystian Erwinski, Mateusz Tejer, Artur Bereit, and Tomasz Tarczewski, "Optimal scheduling for palletizing task using robotic arm and artificial bee colony algorithm," *Engineering Applications of Artificial Intelligence*, vol. 113, pp. 104976, 2022.

[12] Kaushlendra Sharma, Md Mehedi Hassan, Saroj Kumar Pandey, Mimansha Saini, Rajesh Doriya, Manoj Kumar Ojha, Anurag Sinha, Chetna Kaushal, Anupam Kumar Bairagi, and Naglaa F Soliman, "Cloud based multi-robot task scheduling using pmw algorithm," *IEEE Access*, 2023.

[13] Muhammad Muzaffar Iqbal, Muhammad Fahad Zia, Karim Beddiar, and Mohamed Benbouzid, "Optimal scheduling of grid transactive home demand responsive appliances using polar bear optimization algorithm," *IEEE Access*, vol. 8, pp. 222285–222296, 2020.

[14] Ateeq Ur Rehman, Zahid Wadud, Rajvikram Madurai Elavarasan, Ghulam Hafeez, Imran Khan, Zeeshan Shafiq, and Hassan Haes Alhelou, "An optimal power usage scheduling in smart grid integrated with renewable energy sources for energy management," *IEEE Access*, vol. 9, pp. 84619–84638, 2021.

[15] Yi Li, Yong Wang, Yuchao Jin, Xinzhou Cheng, Lexi Xu, and Guanghai Liu, "Research on wireless resource management and scheduling for 5g network slice," in *2021 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE, 2021, pp. 508–513.

[16] Chao Lu, Ren Gao, Lvjiang Yin, and Biao Zhang, "Human–robot collaborative scheduling in energy-efficient welding shop," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 1, pp. 963–971, 2023.

[17] Saeid Alirezazadeh and Luís A Alexandre, "Dynamic task scheduling for human-robot collaboration," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 8699–8704, 2022.

[18] Boonhatai Kruekaew and Warangkhana Kimpan, "Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning," *IEEE Access*, vol. 10, pp. 17803–17818, 2022.

[19] Robert George and Yutaka Kanayama, "A rate-monotonic scheduler for the real-time control of autonomous robots," in *Proceedings of IEEE International Conference on Robotics and Automation*. IEEE, 1996, vol. 3, pp. 2804–2809.

[20] Lui Sha and John B Goodenough, "Real-time scheduling theory and ada," *IEEE Computer*, vol. 23, no. 4, pp. 53–62, 1990.

[21] David L Ripps, *An implementation guide to real-time programming*, Prentice-Hall, Inc., 1989.

[22] Chung Laung Liu and James W Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[23] Kyuseok Im, Sergey Mareninov, M Fernando Palma Diaz, and William H Yong, "An introduction to performing immunofluorescence staining," *Biobanking: methods and protocols*, pp. 299–311, 2019.

[24] John MS Bartlett and David Stirling, "A short history of the polymerase chain reaction," *PCR protocols*, pp. 3–6, 2003.

[25] Berndt Brehmer, "Dynamic decision making: Human control of complex systems," *Acta psychologica*, vol. 81, no. 3, pp. 211–241, 1992.

[26] Liu Renke, Rajesh Piplani, and Carlos Toro, "A review of dynamic scheduling: context, techniques and prospects," *Implementing Industry 4.0: The Model Factory as the Key Enabler for the Future of Manufacturing*, pp. 229–258, 2021.

[27] J Hempenius, J Wieling, JHG Jonkman, OE de Noord, PMJ Coenegracht, and DA Doornbos, "Development of a laboratory robotic system for automated bioanalytical methods—i. the determination of theophylline in human plasma: A comparison between the robotized and manual method," *Journal of pharmaceutical and biomedical analysis*, vol. 8, no. 4, pp. 313–320, 1990.

[28] Enrique J deAndrés Galiana, Juan Luis Fernández-Martínez, and Stephen T Sonis, "Design of biomedical robots for phenotype prediction problems," *Journal of Computational Biology*, vol. 23, no. 8, pp. 678–692, 2016.

[29] Richard G Damian and Manesh J Shah, "Dynamic scheduling," 5 1993.

[30] Samuel Natelson, "Automated clinical laboratory," 1 1972.

[31] Arezou Mohammadi and Selim G Akl, "Scheduling algorithms for real-time systems," *School of Computing Queens University, Tech. Rep*, 2005.

[32] Alan Burns, "Scheduling hard real-time systems: a review," *Software Engineering Journal*, vol. 6, no. 3, pp. 116–128, 1991.

[33] Shichao Chen, Qijie Li, Mengchu Zhou, and Abdullah Abusorrah, "Recent advances in collaborative scheduling of computing tasks in an edge computing paradigm," *Sensors*, vol. 21, no. 3, pp. 779, 2021.

[34] Teofilo Gonzalez and Sartaj Sahni, "Preemptive scheduling of uniform processor systems," *Journal of the ACM (JACM)*, vol. 25, no. 1, pp. 92–101, 1978.

[35] J.R. Jackson, *Scheduling a production line to minimize maximum tardiness*, Research report. Office of Technical Services, 1955.

[36] James R Jackson, "An extension of johnson's results on job idt scheduling," *Naval Research Logistics Quarterly*, vol. 3, no. 3, pp. 201–203, 1956.

[37] Selmer Martin Johnson, "Optimal two-and three-stage production schedules with setup times included," *Naval research logistics quarterly*, vol. 1, no. 1, pp. 61–68, 1954.

[38] Wayne E Smith et al., "Various optimizers for single-stage production," *Naval Research Logistics Quarterly*, vol. 3, no. 1-2, pp. 59–66, 1956.

[39] Peter Brucker, "Scheduling algorithms," *Journal-Operational Research Society*, vol. 50, pp. 774–774, 1999.

[40] Khaldoun Senjab, Sohail Abbas, Naveed Ahmed, and Atta ur Rehman Khan, "A survey of kubernetes scheduling algorithms," *Journal of Cloud Computing*, vol. 12, no. 1, pp. 87, 2023.

[41] Edward G Coffman Jr, Zhen Liu, and Richard R Weber, "Optimal robot scheduling for web search engines," *Journal of scheduling*, vol. 1, no. 1, pp. 15–29, 1998.

[42] Matthew Gombolay, Ronald Wilcox, and Julie Shah, "Fast scheduling of multi-robot teams with temporospatial constraints," 2013.

[43] Massimiliano De Benedetti, Fabrizio Messina, Giuseppe Pappalardo, and Corrado Santoro, "Jarvsis: a distributed scheduler for iot applications," *Cluster Computing*, vol. 20, pp. 1775–1790, 2017.

[44] P Th Zacharia and NA Aspragathos, "Optimal robot task scheduling based on genetic algorithms," *Robotics and Computer-Integrated Manufacturing*, vol. 21, no. 1, pp. 67–79, 2005.

[45] Toby J Teorey and Tad B Pinkerton, "A comparative analysis of disk scheduling policies," *Communications of the ACM*, vol. 15, no. 3, pp. 177–184, 1972.

[46] "Vitrostainer webpage," 2025.

[47] Fernández Enríquez Antonio, CASCALES BARRIO

Jesús, JIMÉNEZ GONZÁLEZ Daniel, VICENTE DÍAZ Saturnino, JIMÉNEZ FERNÁNDEZ Angel Francisco, FERNÁNDEZ CUEVAS Plácido, SÁNCHEZ CUEVAS Pablo, FAJARDO CRUZ Alberto, LÓPEZ DE MINONDO Fernando, RODRIGUES MARTINS Mario Jorge, Olmo Sevilla Asunción, DE LAS HERAS DE LÓZAR Francisco, DOMÍNGUEZ BRAVO María Eugenia, RODRÍGUEZ ANTUÑA Luis, and MONTES SÁNCHEZ Juan Manuel, "Computer-implemented apparatus, method and system for performing histological or immunoenzymatic staining and hybridisation on tissue sections or on cell smears," 10 2024.

[48] Weihao Wang, Chutong Gao, and Leyuan Shi, "Robust optimization on unrelated parallel machine scheduling with setup times," *IEEE Transactions on Automation Science and Engineering*, vol. 20, no. 1, pp. 346–360, 2023.

[49] Mingjie Fan, Xinchao Zhao, Xingquan Zuo, and Lingjuan Ye, "A budget-constrained workflow scheduling approach with priority adjustment and critical task optimizing in clouds," *IEEE Transactions on Automation Science and Engineering*, vol. 22, pp. 6907–6921, 2025.

[50] Andrea Casalino, Andrea Maria Zanchettin, Luigi Piroddi, and Paolo Rocco, "Optimal scheduling of human–robot collaborative assembly operations with time petri nets," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 1, pp. 70–84, 2021.

[51] John A Stankovic, "Real-time and embedded systems," *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 205–208, 1996.

[52] Guillem Bernat, Alan Burns, and Albert Liamosi, "Weakly hard real-time systems," *IEEE transactions on Computers*, vol. 50, no. 4, pp. 308–321, 2001.

[53] Seohyun Jeon, Jaeyeon Lee, and Jaehong Kim, "Multi-robot task allocation for real-time hospital logistics," in *2017 IEEE International conference on systems, man, and cybernetics (SMC)*. IEEE, 2017, pp. 2465–2470.

[54] Bikranta Adhikari, Shivanjali Ranashing, Benjamin A Miller, Vesna D Novak, and Chao Jiang, "Learning dynamic patient-robot task assignment and scheduling for a robotic rehabilitation gym," in *2022 International Conference on Rehabilitation Robotics (ICORR)*. IEEE, 2022, pp. 1–6.

[55] Weiwei Wan, Takeyuki Kotaka, and Kensuke Harada, "Arranging test tubes in racks using combined task and motion planning," *Robotics and Autonomous Systems*, vol. 147, pp. 103918, 2022.

[56] Yao Fehlis, Paul Mandel, Charles Crain, Betty Liu, and David Fuller, "Accelerating drug discovery with artificial: a whole-lab orchestration and scheduling system for self-driving labs," *arXiv preprint arXiv:2504.00986*, 2025.

[57] Catarina Rema, Pedro Costa, Manuel Silva, and Eduardo J Solteiro Pires, "Task scheduling with mobile robots—a systematic literature review," *Robotics*, vol. 14, no. 6, pp. 75, 2025.

[58] An Zou, Yuankai Xu, Yinchen Ni, Jintao Chen, Yehan Ma, Jing Li, Christopher Gill, Xuan Zhang, and Yier Jin, "A survey of real-time scheduling on accelerator-based heterogeneous architecture for time critical applications," *arXiv preprint arXiv:2505.11970*, 2025.

Inteligencia artificial en sistemas empotrados y reconfigurables

Semantic RAID: Lightweight Fault Tolerance via Semantic Redundancy for Distributed Edge Inference

Radu Spaimoc[1], Lluis Mas[2], Jordi Vilaplana[3], Josep Rius[4], Didac Florensa[5], Jordi Mateo[6]

Abstract — ***Edge computing enables deploying machine learning models close to data sources, reducing latency and communication overhead. However, edge environments are inherently unreliable, and node failures can disrupt predictive services at inference time. Existing fault-tolerance approaches typically rely on full model replication or cloud fallback, which incur significant storage or latency costs and are not well suited to decentralized, resource-constrained deployments.***

We propose Semantic RAID, a lightweight approach to fault tolerance based on semantic redundancy. Instead of replicating entire models, nodes store compact class-responsibility vectors that summarize the output behavior of peer models. These semantic fragments are distributed across nodes using a striping strategy and used at inference time to approximate the predictions of a failed node through collaborative aggregation in a shared logit space.

The recovery mechanism combines peer predictions with a lightweight prior derived from the failed node, using temperature-controlled scaling and entropy-based confidence weighting. This formulation enables inference-time recovery without access to model parameters or training data, and is designed to operate under heterogeneous, non-IID data distributions.

Experiments on CIFAR-10 with controlled non-IID partitions show that Semantic RAID matches naive ensemble performance using the same peers while reducing redundancy overhead by several orders of magnitude compared to replication-based approaches. When combined with selective prediction, the method improves over full-coverage aggregation baselines, particularly under heterogeneous data, while maintaining near-complete coverage. However, under strong heterogeneity, the best individual survivor remains competitive, highlighting a trade-off between aggregation and specialization.

These results demonstrate that semantic redundancy provides a practical and storage-efficient alternative to model replication for fault-tolerant edge intelligence, enabling decentralized recovery of predictive functionality under node failures.

Keywords — ***Edge AI, fault-tolerant systems, distributed machine learning, semantic redundancy, collaborative inference, selective prediction, non-IID data, resource-constrained systems***

I. Introduction

Edge computing has emerged as a key paradigm for deploying intelligent services close to data sources, enabling low-latency decision making and reducing communication with centralized cloud infrastructures. This paradigm is particularly relevant in applications such as autonomous systems, industrial monitoring, and IoT networks, where real-time inference and data locality are critical.

However, edge environments are inherently unreliable. Nodes may fail due to hardware limitations, energy constraints, or connectivity issues, leading to interruptions in predictive services. In such settings, maintaining the availability of machine learning functionality at *inference time* becomes a fundamental challenge.

A common approach to fault tolerance is to replicate models across multiple nodes or to rely on cloud fallback mechanisms. While effective, these strategies incur significant storage overhead or introduce latency that may be unacceptable for time-sensitive applications. More recent approaches leverage collaborative inference or knowledge distillation to reduce redundancy, but they are primarily designed for training or model compression, and typically assume that all models remain available at inference time.

This reveals a gap in current edge intelligence systems: the lack of a lightweight, decentralized mechanism to approximate predictive functionality when a node becomes unavailable, without requiring full model replication, retraining, or centralized coordination.

In this paper, we explore the hypothesis that predictive behavior can be approximated through *semantic redundancy*, where nodes store compact summaries of each other's output behavior instead of full models. Concretely, we focus on class-level statistics that capture how a model distributes its predictions across classes. These summaries are inexpensive to store, independent of model architecture, and can be shared across nodes with negligible overhead.

Inspired by redundancy principles in RAID systems, we distribute these lightweight semantic fragments across nodes using a striping strategy. Unlike classical RAID, recovery is not algebraic. Instead, when a node fails, its predictions are approximated at inference time by combining the outputs of surviving models in a shared logit space, guided by a lightweight semantic prior associated with the failed node. The goal is not exact reconstruction, but a practical approximation of predictive behavior under strict storage constraints.

To this end, we propose *Semantic RAID*, a fault-tolerant architecture for edge AI systems. Each node

[1]Department of Computer Engineering, University of Lleida, e-mail: radu.spaimoc@udl.cat
[2]Department of Computer Engineering, University of Lleida, e-mail: lluis.mas@udl.cat
[3]Department of Computer Engineering, University of Lleida, e-mail: jordi.vilaplana@udl.cat
[4]Department of Computer Engineering, University of Lleida, e-mail: josep.riustorrento@udl.cat
[5]Department of Computer Engineering, University of Lleida, e-mail: didac.florensa@udl.cat
[6]Department of Computer Engineering, University of Lleida, e-mail: jordi.mateo@udl.cat

stores a compact *class-responsibility vector* summarizing the label distribution of its local data (on the order of tens of bytes), which is distributed across peers. During recovery, predictions from surviving models are aggregated in logit space using (i) temperature-controlled scaling, (ii) prior modulation derived from the stored semantic fragment, and (iii) confidence-based weighting based on predictive entropy. This enables inference-time approximation of a failed node without access to its parameters or training data.

The main contributions of this work are:

- A formulation of **semantic redundancy** for distributed edge inference, enabling approximation of node predictions at inference time without full model replication.
- A **lightweight semantic representation** based on class-responsibility vectors, requiring only a few tens of bytes per protected node and independent of model architecture.
- A **logit-space aggregation mechanism** that combines peer predictions with a lightweight semantic prior using temperature scaling and entropy-based weighting.
- A comprehensive empirical evaluation demonstrating that the proposed method matches naive ensemble performance using the same peers, while reducing redundancy overhead by several orders of magnitude. We further analyze the trade-off between accuracy and coverage under selective prediction, and show that under strong heterogeneity the best individual survivor can outperform aggregation-based reconstruction.

Importantly, the semantic prior alone does not improve performance over naive aggregation and can introduce small degradations. Instead, it acts as a lightweight structural bias within the aggregation framework, whose effectiveness depends on its interaction with peer predictions and the aggregation mechanism.

II. State of the Art

Distributed edge systems are increasingly used to execute machine learning tasks close to data sources, enabling low latency and improved privacy. However, edge devices are typically constrained in compute, memory, and communication resources. As a result, much of the existing literature focuses on improving efficiency during *training and deployment*, rather than on maintaining predictive functionality at inference time under node failures.

A. Model Compression and Knowledge Distillation

A large body of work addresses resource constraints in edge environments through model compression and knowledge distillation (KD) techniques [1]. Methods such as pruning and quantization reduce model size, while KD enables transferring knowledge from larger models to smaller ones.

In distributed settings, KD has also been used to reduce communication overhead. For example, Wu et al. [2] propose a communication-efficient federated learning framework based on distillation, where models exchange softened outputs instead of parameters.

These approaches improve communication efficiency and model compactness, but they assume that models remain available after training and do not provide mechanisms to approximate predictions when a node fails at inference time.

B. Semantic Representations in Federated Learning

Recent work explores the use of compact semantic representations, such as prototypes or class-level summaries, to align models trained on non-IID data. For example, Zhang et al. [3] propose a prototype-based KD framework for personalized federated learning, while Wu et al. [4] introduce global prototype distillation to guide local updates.

Other approaches explore dataset distillation to construct compact shared datasets. Arazzi et al. [5] demonstrate that lightweight data representations can approximate the original data distribution.

These methods show that compact semantic information can effectively summarize model behavior and facilitate knowledge transfer. However, they are primarily designed for training or model alignment, rather than for reconstructing predictions at inference time.

More broadly, this line of work aligns with the emerging concept of semantic communication in edge systems, where compact representations capture task-relevant information instead of raw data [6]. Our approach builds on this intuition, but focuses on inference-time use of such summaries.

C. Collaborative Inference and Fault Tolerance

A smaller set of works addresses resilience in distributed inference. For instance, Wang et al. [7] propose enhancing local models through shared statistical signatures to enable surrogate nodes to replace failed services in edge environments. However, such approaches rely on training-time adaptation and data exchange, and do not address inference-time approximation under complete node failure.

Furthermore, Jamshidi et al. [8] propose a *Modifier Module* that aggregates predictions from peer models to compensate for node failures without duplicating full models. This demonstrates that collaborative inference can improve robustness with limited overhead.

However, existing approaches typically rely on additional learned components, assume partial model availability, or do not explicitly address the case where an entire node is missing. In particular, they do not provide a mechanism to guide aggregation using lightweight, node-specific information. Moreover, they do not address how to approximate predictions when an entire node becomes unavailable at inference time without prior replication.

D. Coded and Redundant Inference

Recent work has explored coding-based approaches for fault-tolerant inference. For example, Jhunjhunwala et al. [9] propose an erasure-coded framework that reconstructs predictions using redundancy across model components.

While such approaches provide principled recovery guarantees, they typically require structured encoding schemes, additional computation, or modifications to model architectures. These requirements limit their applicability in lightweight, heterogeneous edge environments.

E. Limitations of Existing Approaches

Overall, existing work falls into three main categories: (i) model compression and KD, which reduce resource usage but do not provide inference-time resilience; (ii) semantic or prototype-based methods, which enable compact knowledge sharing but focus on training; and (iii) collaborative or coded inference approaches, which improve robustness but either require additional components or introduce computational overhead.

In particular, current methods either rely on full model availability, incur significant storage costs through replication, or depend on training-time mechanisms that are not applicable at inference time. Ensemble-based approaches provide strong predictive performance, but require maintaining multiple full models, leading to storage costs that scale linearly with the number of replicas. Crucially, none of these approaches provide a lightweight mechanism to approximate the predictions of a completely unavailable node at inference time without full replication.

F. Positioning of This Work

In contrast to prior approaches, we focus explicitly on *inference-time approximation under complete node failure*. Our method distributes lightweight summaries of model behavior across nodes and approximates the predictions of a failed node through collaborative aggregation in a shared output space.

The proposed formulation is related to logit adjustment techniques [10], where class-dependent priors are incorporated into model outputs. In our setting, the class-responsibility vector acts as a lightweight prior that biases aggregation without requiring access to model parameters. Importantly, this prior does not improve performance in isolation, but provides a structured bias within the aggregation mechanism.

Furthermore, the use of entropy-based abstention connects our approach to selective prediction frameworks [11], which improve reliability by allowing models to abstain on uncertain inputs.

Unlike prior methods, our approach does not require retraining, additional modules, or full model replication. Instead, it leverages lightweight semantic summaries (on the order of tens of bytes per node) to guide logit-space aggregation, matching naive ensemble performance using the same peers while significantly reducing redundancy overhead.

III. Semantic RAID Model

We consider a distributed edge system composed of nodes:

$$N = \{N_1, N_2, \ldots, N_n\} \tag{1}$$

Each node N_j maintains a local dataset D_j and a predictive model $f_j : \mathcal{X} \to \mathcal{Y}$. We assume a shared label space $\mathcal{Y}$ across nodes. Data distributions are typically non-IID.

Our objective is to approximate the predictive functionality associated with a failed node N_j using only surviving models and lightweight semantic information, without full model replication or centralized coordination.

A. Semantic Representation

Instead of storing model parameters or latent representations, each node N_j is represented by a lightweight semantic vector:

$$R_j \in \mathbb{R}^{|\mathcal{Y}|} \tag{2}$$

where $R_j[c]$ denotes the empirical proportion of class c in the local dataset D_j.

To ensure numerical stability and avoid zero-probability classes, a small smoothing constant $\epsilon > 0$ is added:

$$R_j[c] = \frac{n_c + \epsilon}{\sum_{c'}(n_{c'} + \epsilon)} \tag{3}$$

where n_c is the number of samples of class c in D_j. The vector is normalized such that:

$$\sum_{c \in \mathcal{Y}} R_j[c] = 1 \tag{4}$$

The storage cost of R_j is linear in the number of classes and independent of model size, making it suitable for resource-constrained environments.

B. Semantic Striping

To distribute redundancy, each node shares its semantic representation with a subset of peers:

$$\mathcal{S}_j \subset N \setminus \{N_j\}, \quad |\mathcal{S}_j| = s \ll n \tag{5}$$

This striping strategy reduces storage complexity from $O(n^2)$ to $O(n)$.

Unlike classical RAID, recovery is not algebraic. Instead, the system produces a surrogate prediction associated with the failed node through collaborative inference using surviving nodes.

C. Similarity-Aware Allocation

Stripe formation is guided by similarity between nodes. We define similarity as a combination of:

- agreement in output space (logit cosine similarity),
- similarity of label distributions (Jensen–Shannon divergence).

This ensures that nodes grouped in the same stripe operate in a compatible prediction space, supporting more reliable surrogate prediction under node failure.

D. Approximation in Logit Space

Upon failure of node N_j, a subset $\mathcal{R} \subseteq \mathcal{S}_j$ of surviving nodes collaborates to produce a surrogate prediction.

Each node $N_k \in \mathcal{R}$ produces logits:

$$L_k(x) = f_k(x) \tag{6}$$

Approximation is performed in the shared output space by combining these logits with the semantic prior R_j:

$$\tilde{L}_k(x) = \frac{L_k(x)}{T} + \frac{\log R_j}{T_p} \tag{7}$$

where $T > 0$ controls the sharpness of peer predictions and $T_p > 0$ controls the influence of the semantic prior.

The final prediction is obtained as a weighted aggregation:

$$\hat{L}(x) = \sum_{k \in \mathcal{R}} w_k(x)\, \tilde{L}_k(x) \tag{8}$$

E. Confidence-Based Gating

Weights are computed dynamically based on prediction confidence:

$$w_k(x) = \frac{\exp\left(-H_k(x)\right)}{\sum_{l \in \mathcal{R}} \exp\left(-H_l(x)\right)} \tag{9}$$

where $H_k(x)$ is the entropy of the predictive distribution obtained from $\tilde{L}_k(x)$:

$$H_k(x) = -\sum_{c \in \mathcal{Y}} p_k(c \mid x)\, \log p_k(c \mid x) \tag{10}$$

This mechanism favors nodes with more confident predictions. However, its effectiveness depends on the level of data heterogeneity; in highly non-IID settings, uniform aggregation may perform comparably.

If gating is disabled, weights are uniform.

F. Selective Abstention

To avoid low-confidence predictions, we define an entropy-based abstention criterion:

$$H(x) = -\sum_{c \in \mathcal{Y}} p(c \mid x)\, \log p(c \mid x) \tag{11}$$

The system abstains when:

$$H(x) > \tau \tag{12}$$

In practice, the threshold τ is selected to balance accuracy and coverage, ensuring that the system maintains a minimum fraction of answered queries.

G. Hyperparameter Selection

Hyperparameters are selected using a held-out validation set through a two-stage procedure.

First, the logit temperature T is selected to maximize validation accuracy without abstention, isolating its effect on prediction sharpness.

Second, the prior temperature T_p and abstention threshold τ are jointly selected to maximize coverage-adjusted accuracy:

$$\text{score} = \text{accuracy} \times \text{coverage} \tag{13}$$

subject to a minimum coverage constraint. In practice, this constraint is not binding, as coverage remains close to 100% across all evaluated settings.

H. Optimization Objective

The objective of the system is to approximate the predictive behavior associated with the failed node:

$$\min_{\hat{f}_j} \mathbb{E}_{x \sim \mathcal{D}} \left[\mathcal{L}\big(\hat{f}_j(x), f_j(x)\big) \right] \tag{14}$$

$$\min_{\hat{f}_j} \ \mathbb{E}_{x \sim \mathcal{D}} \left[\mathcal{L}\big(\hat{f}_j(x), f_j(x)\big) \right] \tag{15}$$

This objective reflects a design goal rather than a directly computable quantity at inference time, since f_j is unavailable after failure. In practice, the system approximates this objective through structured aggregation of peer predictions guided by lightweight semantic priors.

IV. System Architecture

We present the system architecture of Semantic RAID for distributed edge environments. The design reflects the lightweight nature of the proposed method, avoiding complex encoding procedures and relying on shared semantic priors and collaborative inference.

Each node N_i consists of three main components: (i) Local Intelligence Unit, (ii) Semantic Storage Unit, and (iii) Recovery Engine. A lightweight coordination mechanism governs stripe allocation and failure handling.

A. Local Intelligence Unit

The Local Intelligence Unit maintains the node's predictive model f_i and performs local training and inference on dataset D_i.

In addition, the unit exposes model outputs $f_i(x)$ in the shared output space, enabling direct collaboration between nodes without requiring alignment of internal feature representations.

B. Semantic Storage Unit

Each node maintains a lightweight semantic vector for nodes in its stripe:

$$\{R_j \mid N_j \in \mathcal{S}_i\} \tag{16}$$

where R_j is the class-responsibility vector of node N_j.

This representation requires only $|\mathcal{Y}|$ scalar values per node and is independent of model architecture. Storage overhead is therefore negligible compared to full model replication.

C. Stripe Allocation Mechanism

Nodes are organized into stripe sets:

$$\mathcal{S}_j \subset N \setminus \{N_j\}, \quad |\mathcal{S}_j| = s \ll n \tag{17}$$

Stripe allocation is based on similarity between nodes, combining:

- similarity of label distributions,
- agreement in output space (logit similarity).

This ensures that nodes within a stripe produce compatible predictions, supporting more reliable surrogate prediction under node failure.

D. Failure Detection and Orchestration

Failures are detected through lightweight heartbeat or timeout mechanisms.

Upon detecting the failure of node N_j, the system:

1. identifies available nodes in $\mathcal{S}_j$,
2. selects a subset $\mathcal{R} \subseteq \mathcal{S}_j$,
3. initiates the recovery process.

This process is decentralized and can be implemented using local coordination or precomputed routing tables.

E. Recovery Engine

The Recovery Engine produces a surrogate prediction associated with the failed node in the shared output space.

Each node $N_k \in \mathcal{R}$ produces logits:

$$L_k(x) = f_k(x) \tag{18}$$

These logits are combined with the semantic prior R_j:

$$\tilde{L}_k(x) = \frac{L_k(x)}{T} + \frac{\log R_j}{T_p} \tag{19}$$

Final predictions are obtained through weighted aggregation:

$$\hat{L}(x) = \sum_{k \in \mathcal{R}} w_k(x)\, \tilde{L}_k(x) \tag{20}$$

F. Confidence-Based Gating

Weights are computed as:

$$w_k(x) = \frac{\exp\left(-H_k(x)\right)}{\sum_{l \in \mathcal{R}} \exp\left(-H_l(x)\right)} \tag{21}$$

This mechanism favors more confident predictions. However, its effectiveness depends on the level of data heterogeneity; in highly non-IID settings, uniform aggregation may perform comparably.

If disabled, weights become uniform, resulting in a standard ensemble with a shared prior.

G. Selective Abstention

The system evaluates prediction entropy:

$$H(x) = -\sum_{c \in \mathcal{Y}} p(c \mid x)\, \log p(c \mid x) \tag{22}$$

The system abstains when:

$$H(x) > \tau \tag{23}$$

H. Computational Considerations

The architecture is designed for efficiency:

- no encoding or distillation step is required,
- storage is limited to class-responsibility vectors,
- aggregation operates in logit space with low overhead,
- similarity is computed only during stripe allocation.

This makes Semantic RAID suitable for resource-constrained edge deployments.

V. Experimental Setup

We evaluate Semantic RAID in a distributed edge setting with heterogeneous data distributions and simulated node failures.

A. Dataset and Partitioning

Experiments are conducted on CIFAR-10. The dataset is split into training, validation, and test sets, with 2,000 samples used for validation and 8,000 samples reserved for testing. The validation set is used exclusively for hyperparameter selection, while all reported results are computed on the held-out test set.

To simulate non-IID conditions, the training data is partitioned across $n = 5$ nodes using a Dirichlet distribution with parameter α. Smaller values of α correspond to stronger heterogeneity. We evaluate $\alpha \in \{10.0, 1.0, 0.5, 0.1\}$.

Each node trains the same convolutional neural network architecture with two convolutional layers followed by fully connected layers. Models are trained locally using cross-entropy loss and the Adam optimizer.

B. Failure Model

We simulate single-node failures. For each configuration, one node is removed and its predictions are approximated using the remaining nodes in its stripe.

Results are averaged over all possible failed nodes and five independent trials with different random seeds.

C. Stripe Configuration

Nodes are organized into stripes of size $s = 3$. For each node, stripes are constructed by selecting the most similar peers according to a similarity score combining label distribution similarity and logit-space agreement.

Recovery is performed using the surviving nodes within the assigned stripe.

D. Baselines

We compare against standard recovery strategies:

- **Upper bound**: original model before failure
- **Naive ensemble**: uniform averaging of raw logits from surviving models, without semantic prior, gating, or temperature scaling

- **Best survivor**: best-performing individual surviving node
- **Mean survivor**: average performance of surviving nodes
- **Worst survivor**: worst-performing surviving node

All baselines operate at full coverage (no abstention), enabling direct comparison with RAID(noτ).

E. Semantic RAID Variants

We evaluate two configurations:

- **RAID(noτ)**: full-coverage aggregation without abstention
- **RAID(τ)**: selective prediction with entropy-based abstention

For RAID(τ), accuracy is computed over accepted predictions and reported together with coverage. The threshold τ is selected on the validation set under a minimum coverage constraint of 0.70.

F. Hyperparameter Selection

Hyperparameters are selected using a two-stage procedure on the validation set.

First, the logit temperature T is selected to maximize validation accuracy without abstention.

Second, the prior temperature T_p and abstention threshold τ are jointly selected to maximize coverage-adjusted accuracy:

$$\text{score} = \text{accuracy} \times \text{coverage} \tag{24}$$

subject to a minimum coverage constraint of 0.70.

In practice, this constraint is not binding, as the selected configurations maintain near-complete coverage ($> 99.9\%$) across all evaluated settings.

We evaluate $T \in \{0.5, 1.0, 2.0\}$, $T_p \in \{0.5, 1.0, 2.0, 4.0\}$, and $\tau \in \{1.0, 1.5, 2.0, 2.5, 3.0\}$.

G. Out-of-Distribution Evaluation

To evaluate abstention behavior, we use SVHN as an out-of-distribution dataset. We report the fraction of samples rejected by the model (abstention rate) as a proxy for out-of-distribution sensitivity.

H. Evaluation Protocol

All results are reported as mean ± standard error over five trials. For each trial, models are retrained and partitions are resampled.

RAID(noτ) is used for direct comparison with baselines under full coverage. RAID(τ) is reported separately as a selective prediction setting, where accuracy must be interpreted jointly with coverage.

VI. Results

A. Recovery under Data Heterogeneity

Figure 1 shows recovery accuracy as a function of data heterogeneity.

Table I reports the exact numerical values corresponding to Figure 1.

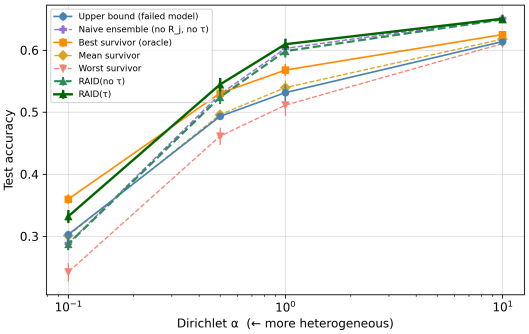


Fig. 1: Recovery accuracy as a function of data heterogeneity. RAID(noτ) closely matches naive ensemble performance, while RAID(τ) improves over full-coverage aggregation baselines through selective prediction at near-full coverage.

α	Upper	NaiveEns	Best	Mean	Worst	RAID(noτ)	RAID(τ)
10.0	0.614	**0.651**	0.625	0.618	0.611	0.649	0.651
1.0	0.532	0.603	0.568	0.540	0.511	0.598	**0.610**
0.5	0.493	0.529	0.531	0.496	0.461	0.524	**0.545**
0.1	0.303	0.288	**0.360**	0.301	0.242	0.288	0.332

Table I: **Recovery accuracy under node failure.** Results are averaged over all failed nodes and trials. RAID(noτ) matches naive ensemble performance using the same peers, while RAID(τ) improves over full-coverage aggregation baselines through selective prediction. Under strong heterogeneity ($\alpha = 0.1$), the best individual survivor remains stronger due to specialization.

RAID(noτ) achieves performance comparable to the naive ensemble across all values of α. The gap remains small (within ∼0.2–0.5%), showing that semantic striping preserves most of the predictive benefit of ensembling despite using only lightweight semantic fragments.

Under moderate to strong heterogeneity, RAID(τ) consistently improves over full-coverage aggregation baselines such as RAID(noτ) and naive ensemble. For example, at $\alpha = 0.1$, accuracy increases from 28.8% (naive ensemble) to 33.2%, while maintaining coverage above 99.9%.

However, the best individual survivor achieves 36.0% at $\alpha = 0.1$, outperforming aggregation-based methods. This highlights a trade-off between robustness through aggregation and specialization under strong heterogeneity.

These results indicate that selective prediction becomes increasingly beneficial as heterogeneity increases, allowing the system to avoid unreliable predictions without sacrificing coverage.

B. Comparison to Baselines

To ensure a fair comparison, we focus on RAID(noτ), which operates under full coverage.

Figure 2 shows the accuracy gap relative to baseline methods. RAID(noτ) consistently matches naive ensemble performance within a small margin across all heterogeneity levels, consistent with the values reported in Table I.

Compared to individual nodes, RAID provides clear improvements over the mean and worst survivor baselines. However, under strong heterogeneity ($\alpha = 0.1$), the best individual survivor outperforms aggregation-based methods, indicating that combining predictions may dilute specialized knowledge.

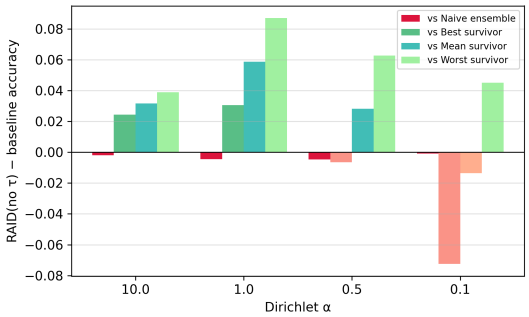


Fig. 2: Recovery gap with respect to baseline methods. RAID(noτ) closely matches naive ensemble performance and consistently outperforms weaker baselines.

Overall, these results confirm that collaborative inference provides robust performance across settings, while highlighting the limits of aggregation under extreme non-IID conditions.

C. Effect of Semantic Prior and Gating

We isolate the contribution of the semantic prior and the gating mechanism.

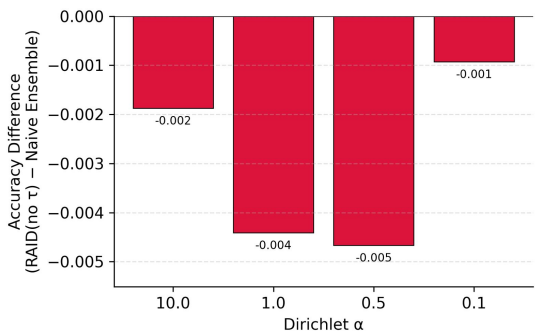


Fig. 3: Contribution of the semantic prior (R_j) compared to naive ensemble using the same peers.

Figure 3 shows that the semantic prior alone does not consistently improve over naive ensembling. Across all settings, the difference remains small and is slightly negative on average (up to $\sim$0.5%).

This indicates that the class-responsibility vector acts as a weak prior that does not, by itself, enhance predictive accuracy.

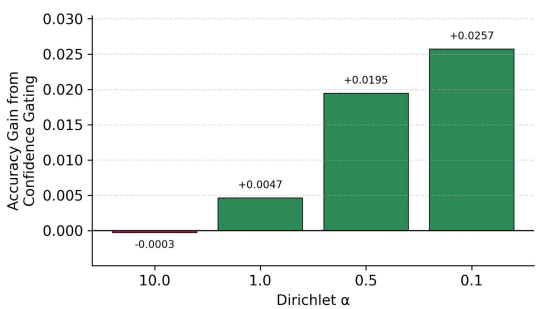


Fig. 4: Effect of confidence-based gating compared to uniform aggregation.

In contrast, Figure 4 shows that confidence-based gating becomes increasingly beneficial as heterogeneity increases. While its effect is negligible in near-IID settings, it provides gains under strong heterogeneity, reaching up to +2.6% at $\alpha = 0.1$.

Method	Storage overhead per node	Relative overhead
Full replication	$\sim$2129 KB	1×
Semantic RAID	0.078 KB	$< 10^{-4}$×

Table II: **Redundancy overhead comparison.** Semantic RAID reduces redundancy overhead by more than four orders of magnitude compared to full model replication.

These gains arise primarily from selective prediction (i.e., abstaining on uncertain samples) rather than from weighting alone.

D. Confidence and Coverage

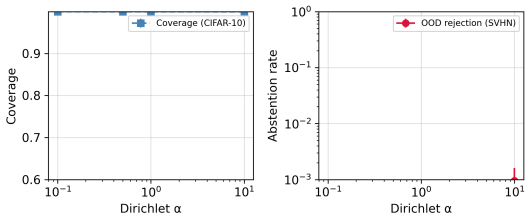


Fig. 5: Coverage and abstention as a function of heterogeneity.

Figure 5 shows the relationship between coverage and abstention. The selected threshold maintains near-complete coverage (above 99.9%) across all settings.

At the same time, abstention rates on SVHN remain very low (below 0.1%), indicating that the entropy-based criterion provides limited out-of-distribution detection capability.

In practice, the abstention mechanism acts as a mild confidence filter that selectively removes uncertain predictions without significantly reducing coverage.

E. Storage Efficiency

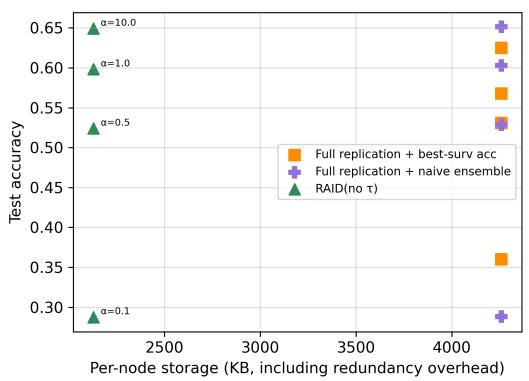


Fig. 6: Storage versus accuracy trade-off. Semantic RAID achieves comparable accuracy to full replication while requiring several orders of magnitude less redundancy overhead.

Figure 6 compares storage cost and accuracy. A semantic fragment requires only tens of bytes per protected node, compared to approximately 2 MB for a full model.

Table II highlights the corresponding reduction in redundancy overhead.

Despite this drastic reduction, RAID(noτ) achieves performance comparable to naive ensemble baselines, demonstrating that semantic redundancy

can effectively replace full model replication in edge environments.

VII. Discussion

The experimental results provide several insights into the behavior and limitations of Semantic RAID in distributed edge environments.

A. Semantic Redundancy vs. Model Replication

A central finding is that lightweight semantic fragments can effectively replace full-model redundancy with minimal performance loss. RAID(noτ) consistently matches naive ensemble performance within a small margin (typically below 0.5%), despite requiring several orders of magnitude less *redundancy overhead*.

This demonstrates that full model replication is not strictly necessary to preserve predictive functionality under node failure. Instead, distributing compact summaries of model behavior is sufficient to approximate ensemble-level performance when combined with collaborative inference.

B. Limits of Collaborative Reconstruction

A key limitation emerges under strong heterogeneity. In highly non-IID settings ($\alpha = 0.1$), the best individual survivor outperforms aggregation-based reconstruction (36.0% vs 33.2%).

This indicates that collaborative aggregation may dilute specialized knowledge when node distributions are highly skewed. In such cases, selecting or routing to the most relevant peer could be more effective than uniform aggregation. This highlights an inherent trade-off between robustness through aggregation and preservation of specialized expertise.

C. Role of the Semantic Prior and Gating

The results show that the semantic prior alone does not improve performance over naive ensembling. Across all settings, its effect remains small and slightly negative (on the order of 0.1–0.5%), indicating that class-responsibility vectors act as weak priors in isolation.

In contrast, confidence-based gating plays a more important role under heterogeneous conditions. While its effect is negligible in near-IID settings, it becomes increasingly beneficial as heterogeneity grows, reaching gains of up to +2.6% at $\alpha = 0.1$.

However, these gains are primarily attributable to selective prediction (i.e., abstaining on uncertain samples) rather than to weighting alone. This suggests that while adaptive weighting is helpful, the current entropy-based mechanism remains heuristic and may not fully capture peer reliability in all cases.

D. Behavior under Data Heterogeneity

The behavior of Semantic RAID depends strongly on the level of data heterogeneity. Under near-IID conditions ($\alpha = 10$), all methods perform similarly, and the impact of both the semantic prior and gating is minimal.

As heterogeneity increases, the importance of adaptive mechanisms becomes more pronounced. In particular, selective prediction plays a key role in improving performance, while naive aggregation becomes less reliable.

At the same time, extreme heterogeneity exposes limitations of aggregation-based approaches, as specialized models may outperform collaborative reconstruction. This suggests that future systems may benefit from hybrid strategies combining aggregation with instance-level routing.

E. Selective Inference and Reliability

Introducing the confidence threshold τ allows the system to operate as a selective classifier. In this setting, RAID(τ) consistently improves over full-coverage aggregation baselines while maintaining near-complete coverage (above 99.9%).

These gains are especially pronounced under strong heterogeneity, where selective prediction helps filter unreliable outputs. In practice, the mechanism acts as a lightweight reliability filter that improves prediction quality with negligible impact on coverage.

However, the entropy-based criterion provides limited out-of-distribution detection capability, as reflected by the very low rejection rates on SVHN. This indicates that the mechanism is better suited for filtering low-confidence in-distribution samples than for detecting distributional shifts.

F. Storage–Accuracy Trade-off

One of the main advantages of Semantic RAID is its efficiency in redundancy. By replacing full-model replication with class-responsibility vectors, the system reduces redundancy overhead by more than four orders of magnitude.

Importantly, this reduction has minimal impact on performance relative to naive ensemble baselines. This highlights a favorable storage–accuracy trade-off, making the approach particularly suitable for resource-constrained edge deployments.

G. Limitations and Future Work

Several limitations should be noted. First, the current semantic representation is simple and does not improve reconstruction beyond naive aggregation. Exploring richer semantic summaries or learned representations is a promising direction.

Second, while gating improves performance under heterogeneity, the current entropy-based formulation remains heuristic. More principled or learned aggregation mechanisms could further enhance robustness.

Third, the abstention mechanism provides only weak out-of-distribution detection. Incorporating more advanced uncertainty estimation or OOD detection methods is an important avenue for future work.

Fourth, the results suggest that routing to the most relevant peer may outperform aggregation under strong heterogeneity. Designing hybrid strategies

that combine routing and aggregation is an important direction.

Finally, the evaluation focuses on single-node failures and a relatively small number of nodes. Extending the approach to larger-scale systems and multiple simultaneous failures remains an open challenge.

Overall, the results show that Semantic RAID provides a lightweight and effective approximation to replicated inference, while highlighting the importance of adaptive aggregation and the challenges posed by heterogeneous data distributions.

VIII. Conclusion

We introduced Semantic RAID, a lightweight approach for fault tolerance in distributed edge intelligence systems based on semantic redundancy. Instead of replicating full models, the proposed method distributes compact class-responsibility vectors across nodes and approximates the predictive functionality of a failed node through collaborative inference in a shared output space.

Experimental results show that Semantic RAID closely matches the performance of naive ensemble baselines across a range of heterogeneity levels (within $\sim$0.5%), while reducing redundancy overhead by several orders of magnitude compared to full model replication. This demonstrates that full replication is not strictly necessary to maintain competitive inference performance in distributed settings.

When combined with selective prediction, the method further improves over full-coverage aggregation baselines under heterogeneous data distributions by abstaining on a small fraction of uncertain inputs. These gains arise from a favorable accuracy–coverage trade-off, rather than improvements in full-coverage aggregation alone.

At the same time, the results highlight important limitations. The semantic prior does not, by itself, improve over naive ensembling, and the current gating mechanism is not always sufficient to reliably identify the most informative peers under strong non-IID conditions. In such scenarios, simpler strategies such as selecting the best available peer can outperform collaborative aggregation (e.g., 36.0% vs 33.2% at $\alpha = 0.1$).

Overall, the findings indicate that lightweight semantic representations are sufficient to preserve the benefits of ensembling with negligible redundancy overhead, but that effective aggregation remains a key challenge in heterogeneous environments.

Future work will explore richer semantic representations, more expressive aggregation and routing strategies, and improved uncertainty estimation methods. Extending the approach to larger-scale systems and multiple simultaneous failures is also an important direction.

In summary, Semantic RAID provides a storage-efficient alternative to replication-based fault tolerance, offering a practical trade-off between memory usage, accuracy, and reliability in decentralized edge systems.

References

[1] Serhiy O. Semerikov, Tetiana A. Vakaliuk, Olha B. Kanevska, Oleksandr A. Ostroushko, and Anatolii O. Kolhatin, "Edge intelligence unleashed: a survey on deploying large language models in resource-constrained environments," *Journal of Edge Computing*, vol. 4, no. 2, pp. 179–233, 2025.

[2] Chuhan Wu, Fangzhao Wu, Lingjuan Lyu, Yongfeng Huang, and Xing Xie, "Communication-efficient federated learning via knowledge distillation," *Nature Communications*, vol. 13, pp. 2032, 2022.

[3] Jianqiao Zhang, Caifeng Shan, and Jungong Han, "FedGMKD: An efficient prototype federated learning framework through knowledge distillation and discrepancy-aware aggregation," in *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024.

[4] Shu Wu, Jindou Chen, Xueli Nie, Yong Wang, Xiancun Zhou, Linlin Lu, Wei Peng, Waseef Menhaj, and Yao Nie, "Global prototype distillation for heterogeneous federated learning," *Scientific Reports*, vol. 14, pp. 12057, 2024.

[5] Marco Arazzi, Mert Cihangiroglu, Serena Nicolazzo, and Antonino Nocera, "Secure federated dataset distillation," *Engineering Applications of Artificial Intelligence*, vol. 160, pp. 111911, 2025.

[6] Wanting Yang, Zi Qin Liew, Wei Yang Bryan Lim, Zehui Xiong, Dusit Niyato, Xuefen Chi, Xianbin Cao, and Khaled B. Letaief, "Semantic communication meets edge intelligence," *IEEE Wireless Communications*, vol. 29, no. 5, pp. 28–35, 2022.

[7] Qiyuan Wang, Saleh ALFahad, Jordi Mateo Fornes, Christos Anagnostopoulos, and Kostas Kolomvatsos, "Resilient edge predictive analytics by enhancing local models," *Open Computer Science*, vol. 14, no. 1, pp. 20230116, 2024.

[8] Omid Jamshidi, Mahdi Abbasi, Abbas Ramazani, Atefeh S. Shahraki, Zhaocheng Ren, and Amir Taherkordi, "Modifier guided resilient cnn inference enables fault-tolerant edge collaboration for iot," *Scientific Reports*, vol. 15, pp. 45458, 2025.

[9] Divyansh Jhunjhunwala, Neharika Jali, Gauri Joshi, and Shiqiang Wang, "Erasure coded neural network inference via fisher averaging," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, 2024.

[10] Aditya Krishna Menon, Sadeep Jayasumana, Ankit Singh Rawat, Himanshu Jain, Andreas Veit, and Sanjiv Kumar, "Long-tail learning via logit adjustment," in *International Conference on Learning Representations (ICLR)*, 2021.

[11] Stephen Bates, Anastasios Angelopoulos, Lihua Lei, Jitendra Malik, and Michael I. Jordan, "Distribution-free, risk-controlling prediction sets," *Journal of the ACM*, vol. 68, no. 6, 2021.

[12] Haokun Chen, Ahmed Frikha, Denis Krompass, Jindong Gu, and Volker Tresp, "FRAug: Tackling federated learning with non-iid features via representation augmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

Generación eficiente y confiable de SVA basada en LLM locales mediante el uso de TGTS

Javier Romero, M. Soledad Escolar, Jesús Barba,
Fernando Rincón, Julián Caba, Melisa G. Kuzman[1]

***Abstract*—La creciente complejidad de los diseños de hardware modernos ha hecho que la verificación basada en aserciones sea esencial; sin embargo, el desarrollo manual de Aserciones en SystemVerilog (SVA) sigue siendo un importante cuello de botella. Aunque los enfoques recientes basados en Large Language Models (LLM) muestran resultados prometedores, a menudo dependen de APIs comerciales o de modelos computacionalmente intensivos, lo que genera preocupaciones de costo y privacidad en entornos industriales. Este artículo introduce SVA-TGTS, un marco de extracción estructurada con fiabilidad sintáctica que separa la lógica combinacional y secuencial de los módulos RTL y guía a los LLM para generar reglas utilizando Sistemas de transición protegidos y tipificados (TGTS), una representación intermedia que permite la conversión sistemática en aserciones. Comparamos tres estrategias de prompting (simple, compleja y basada en TGTS) en ocho LLM de código abierto (con entre 7B y 33B de parámetros), evaluando la corrección sintáctica y el tiempo de generación. Los resultados muestran que TGTS alcanza más del 85% de corrección sintáctica, superando significativamente al prompting compleja con few-shot, al tiempo que valida eficientemente el comportamiento del diseño.**

***Index Terms*—Typed Guarded Transition Systems, SystemVerilog Assertion, Large Language Model.**

I. Introducción

La creciente escala y heterogeneidad de los sistemas digitales modernos han intensificado los desafíos de la verificación de hardware. A medida que los diseños aumentan en complejidad, la verificación se ha convertido en el coste dominante en el ciclo de vida del hardware. Estudios de la industria como [1] informan que la verificación representa entre el 60%–70% del esfuerzo total de ingeniería, siendo los diseños son cada vez más complejos y con ello más iteraciones antes de funcionar correctamente. La Verificación Basada en Aserciones (ABV) aborda estos desafíos al permitir a los diseñadores especificar formalmente el comportamiento utilizando SystemVerilog Assertions (SVA), integrando propiedades directamente en el RTL para detectar errores de forma temprana y reducir el tiempo de depuración.

[1]*Dpto. Tecnologías y Sistemas de Información. Escuela Superior de Informática*. Universidad de Castilla-La Mancha. Ciudad Real, España. emails: {javier.romero, soledad.escolar, jesus.barba, fernando.rincon, julian.caba, melisagisele}@uclm.es

A pesar de su efectividad, el desarrollo manual de aserciones sigue siendo un importante cuello de botella. Los ingenieros deben analizar el código a nivel de Transferencia de Registros (RTL) para derivar propiedades que capturen tanto el comportamiento combinacional como secuencial, un proceso que requiere mucho trabajo, es propenso a errores y difícil de escalar. Como resultado, la adopción de ABV está limitada no solo por las herramientas formales, sino también por el coste de la creación de aserciones, lo que motiva enfoques automatizados para generar aserciones directamente a partir de RTL.

Los avances recientes en LLMs ofrecen nuevas oportunidades para automatizar tales tareas. Trabajos previos muestran que los LLMs pueden generar SVA a partir de lenguaje natural o código, pero su despliegue práctico enfrenta limitaciones clave [2], [3]. En primer lugar, la confidencialidad de la Propiedad Intelectual (IP) restringe el uso de modelos basados en la nube, ya que el código RTL es altamente sensible. Esto crea una fuerte necesidad de soluciones locales que operen dentro de infraestructuras seguras. En segundo lugar, los costes computacionales y financieros siguen siendo prohibitivos. El *fine-tuning* de LLMs requiere recursos significativos de GPU, a menudo con costes de decenas de miles de dólares por iteración [4]. Además, estos enfoques dependen de grandes conjuntos de datos específicos del dominio (por ejemplo, pares RTL-a-SVA), que rara vez están disponibles fuera de grandes corporaciones. En consecuencia, las Pequeñas y Medianas Empresas (PYMES) quedan efectivamente excluidas, lo que refuerza la necesidad de métodos que eviten el reentrenamiento y permitan el uso de LLMs más pequeños que puedan desplegarse en hardware convencional.

Para abordar estos desafíos, proponemos SVA-TGTS, un marco estructurado y consciente de la sintaxis para la generación automatizada de SVA que opera completamente en infraestructura local. El enfoque se basa en Sistemas de Transiciones con Guardas (GTS) [5], utilizados como una representación intermedia entre RTL y SVA. Argumentamos que la brecha semántica entre RTL y las aserciones temporales es demasiado grande para una generación directa con modelos pequeños, lo que a menudo conduce a errores de sintaxis y alucinaciones.

En su lugar, descomponemos la tarea en dos etapas: extracción estructurada a partir de RTL y generación de reglas TGTS. Estas reglas siguen un paradigma de condición-a-acción y se traducen de manera determinista en SVA válidas.

Este trabajo se centra en aserciones para construcciones fundamentales de flujo de control, que forman la columna vertebral de los diseños digitales y son fuentes frecuentes de errores. Esto establece una base para extender el enfoque a comportamientos temporales más complejos.

En resumen, este trabajo conecta la generación de código basada en IA con los requisitos de verificación industrial. Las principales contribuciones incluyen: (i) una representación intermedia basada en TGTS para la generación de aserciones, (ii) una estrategia de *prompting* estructurada que permite a LLMs pequeños igualar a modelos más grandes sin *fine-tuning*, y (iii) una canalización completa desde SystemVerilog hasta aserciones sintácticamente correctas. Al restringir la generación mediante reglas de transición tipadas, SVA-TGTS mitiga problemas comunes como desajustes de señales y dependencias alucinadas. Hasta donde sabemos, este es el primer enfoque que aprovecha los Sistemas de Transiciones con Guardas para estabilizar LLMs en una canalización de verificación local sin entrenamiento previo (*zero-shot*), operacionalizando métodos formales para la Inteligencia Artificial (IA) generativa.

II. Trabajos relacionados

La integración del Aprendizaje Automático (ML) y la IA en los flujos de Automatización de Diseño Electrónico (EDA) ha evolucionado de ser un tema de investigación a una tendencia industrial clave. Recientes estudios sobre modelos base de circuitos [6] destacan la adopción de la IA a lo largo de todo el ciclo de vida del hardware, desde la síntesis de alto nivel hasta la corrección automática de errores y el análisis de seguridad del hardware. Esta tendencia se ve reforzada por el uso de LLMs como agentes capaces de interpretar código RTL y generar artefactos de diseño auxiliares.

El auge de la síntesis de aserciones basada en LLM ha intensificado el interés de investigación. Trabajos iniciales como AutoSVA [7] demostraron que la información estructural del diseño podía traducirse en plantillas de aserciones reutilizables, incluso sin modelos generativos modernos. A medida que los LLM maduraron, métodos como STELLAR [8] introdujeron el *prompting* guiado por estructura para alinear mejor la intención del diseño con las propiedades formales, mientras que AssertionForge [9] utilizó representaciones estructuradas de especificaciones y RTL para inferir SystemVerilog Assertions adecuadas. También han surgido enfoques multi-agente, ejemplificados por SVAgent [10] para propiedades orientadas a la seguridad. Colectivamente, estos trabajos demuestran la viabilidad de la generación de lenguaje natural o RTL-a-SVA, aunque persisten desafíos en el manejo de estructuras de control profundas, comportamientos temporales complejos y diseños a escala industrial.

Dentro de este panorama en rápida evolución, el marco de trabajo VERT [11] se centra explícitamente en la generación de SVA basada en LLM. VERT realiza un *fine-tuning* de LLMs de código abierto sobre un corpus de pares HDL–SVA para mejorar la calidad de las aserciones. Aunque influyente, el ajuste fino conlleva un coste computacional sustancial y reduce la portabilidad. Además, el conjunto de datos sigue siendo limitado para representar patrones de hardware industrialmente intrincados. Como resultado, VERT exhibe una generalización reducida cuando se enfrenta a lógica de control compleja o propiedades temporales de larga duración. En contraste, SVA-TGTS emplea ingeniería de *prompts* basada en Sistemas de transición protegidos y tipificados (TGTS), evitando el *fine-tuning* al tiempo que admite lógica de control profundamente anidada. Al restringir las salidas a las reglas de TGTS en lugar de SVA puro, el enfoque es más determinista, eficiente y adecuado para el despliegue privado utilizando modelos de lenguaje de gran tamaño.

El uso de TGTS (Typed Guarded Transition Systems) se fundamenta en métodos formales consolidados. Las Acciones Atómicas Guardadas, base de los TGTS, sustentan lenguajes como Bluespec SystemVerilog (BSV) [12], donde el modelo Guardia $\rightarrow$ Acción garantiza un diseño de hardware modular y analizable. De manera similar, el *framework* Kami [13] emplea transiciones guardadas dentro de Coq para la verificación de procesadores. Mientras que estos enfoques se orientan al diseño manual y a la prueba formal, SVA-TGTS reutiliza este formalismo como un andamiaje semántico para la IA generativa.

III. Antecedentes sobre TGTS

Los Sistemas de Transición Guardados (GTS) [5] se introdujeron originalmente como un formalismo basado en estados/eventos para el análisis de fiabilidad, que generaliza varios marcos de modelado clásicos, incluyendo diagramas de bloques y redes de Petri. En un GTS, el sistema representado se define como una tupla de seis elementos $\mathcal{GTS} = \langle V, E, T, \iota, H, B \rangle$, donde:

- **V**: es un conjunto de variables.
- **E**: es un conjunto de eventos.
- **T**: es un conjunto de transiciones donde cada transición $t \in T$ se define mediante la terna $t = \langle G, e, P \rangle$, donde G es la guardia o precondición de la transición, definida como una expresión booleana construida sobre V, e es un evento, y P

es una instrucción construida sobre V llamada postcondición de la transición, es decir $t : G \xrightarrow{e} P$.
- ι: es la asignación inicial.
- **H** and **B** son un par de instrucciones que representan la cabecera (head) y el cuerpo (body) de la aserción, respectivamente.

La semántica de un $\mathcal{GTS}$ viene dada por un grafo de transición potencialmente infinito $\mathcal{G} = \{S, Q\}$, donde S denota el conjunto de todas las valoraciones de estado alcanzables, es decir, todas las asignaciones de variables que pueden obtenerse desde el estado inicial a través de una secuencia de transiciones habilitadas, y $Q \subseteq S \times E \times S$ es la relación de transición. Cada elemento $(\sigma, e, \tau) \in Q$ representa una transición desde el estado σ al estado τ etiquetada por el evento e, y se escribe $\sigma \xrightarrow{e} \tau$. El grafo G se define como la estructura más pequeña que satisface las siguientes condiciones: i) el estado inicial pertenece al grafo, es decir, $\iota \in S$; ii) para cualquier estado $\sigma \in S$ si existe una transición $t = \sigma \xrightarrow{e} \tau$ tal que $\tau = B^{\omega}(H(P(\sigma))) \in S$ y $(\sigma, e, \tau) \in Q$, donde ω denota la aplicación iterativa de B hasta alcanzar un punto fijo τ formalmente $\tau = (B^{\omega} \circ H \circ P)(\sigma)$.

Los TGTS extienden los GTS introduciendo una disciplina de tipos estática sobre las variables de estado, las guardias y las actualizaciones de transición, garantizando la corrección de tipos en todas las ejecuciones. Específicamente, en un TGTS, esta construcción se restringe a estados bien tipados, y se requiere que todas las funciones de transición preserven el tipado, asegurando que S solo contenga valoraciones bien tipadas. Sea V un conjunto de variables y $\Gamma : V \to \mathcal{T}$ una función que asigna un tipo a cada variable. Un estado σ es una valoración tipada si, para cada $x \in V$ se cumple que $\sigma(x) \in Val(\Gamma(x))$, donde $Val(\cdot)$ denota el conjunto de valores asociados a un tipo dado. Considere $\Gamma(x) = Bool$ y $\Gamma(y) = \mathbb{N}$. Un estado $\sigma = \{x = true, y = 3\}$ está bien tipado, mientras que $\sigma' = \{x = 5, y = 3\}$ no lo está, ya que viola la restricción de tipado $\sigma'(x) \in Val(Bool)$.

Como ejemplo ilustrativo, considere un TGTS simple que modela un circuito de conmutación (*toggle*) con una señal de habilitación (*enable*). El sistema se define sobre las variables $V = \{en, out\}$ de tipo booleano. El evento *toggle* se habilita cuando *en* es verdadero, y su efecto es invertir el valor de *out*, preservando *en*. La regla del TGTS se define de la siguiente manera:

$$\frac{\sigma(en) = true}{\sigma \xrightarrow{toggle} \sigma'}$$

Utilizamos la forma estándar para describir la regla, mediante la notación $\frac{guards}{postcondition}$, donde la parte superior especifica las precondiciones o guardias necesarias para que la regla sea aplicable, y la parte inferior describe las postcondiciones resultantes inducidas por la transición.

IV. Pipeline de SVA-TGTS

Esta sección describe el flujo de procesamiento propuesto para SVA-TGTS, cuyo objetivo es generar SVA fiables y formalmente fundamentadas a partir de especificaciones TGTS mediante un proceso guiado por LLM. El flujo de SVA-TGTS opera en cuatro etapas, como se ilustra en la Figura 1: 1) el módulo RTL de entrada se analiza para separar los bloques combinacionales de los secuenciales, extrayendo puertos, parámetros, variables internas y funciones; 2) para cada bloque extraído, se construye un prompt estructurado, específico para la naturaleza combinacional o secuencial del mismo, y se envía a un modelo LLM alojado localmente en Ollama, dándole instrucciones para generar reglas TGTS (pares simples de condición-acción que capturan el comportamiento funcional del bloque); 3) estas reglas TGTS se post-procesan para convertirse en SystemVerilog Assertions, con un análisis de sensibilidad que determina las condiciones adecuadas de reloj (clocking) y reinicio (reset) según el tipo de bloque; 4) las aserciones generadas se ensamblan en un módulo de aserciones completo que preserva la interfaz del módulo original. El módulo de aserciones resultante se evalúa posteriormente utilizando Siemens Questa 2023.4 [14] para verificar la corrección sintáctica, registrando además el tiempo de generación.

A. Extracción RTL y separación de bloques

La primera etapa analiza sistemáticamente el módulo RTL de SystemVerilog de entrada para aislar dos categorías distintas de lógica: 1) **bloques combinacionales**: bloques `always` con sensibilidad `always_comb`, que producen aserciones evaluadas de forma asíncrona; y 2) **bloques secuenciales**: bloques `always` con `always_ff` o `always_latch`, disparados por eventos de reloj `posedge` o `negedge`, que producen propiedades síncronas con condiciones de reinicio (reset). De cada módulo, ya sea combinacional o secuencial, se extraen los siguientes elementos:

- **Puertos y parámetros**: Interfaz del módulo y parámetros de configuración.
- **Variables internas**: Wires y registros declarados dentro del módulo.
- **Funciones internas**: Funciones definidas por el usuario llamadas dentro del módulo.
- **Asignaciones condicionales**: Cada rama de las sentencias `case`.

Esta extracción se realiza usando un parseador ligero que identifica estructuras sintácticas. La separación de lógica combinacional y secuencial es crítica porque cada bloque se procesa usando *prompts* diferentes según su tipo.

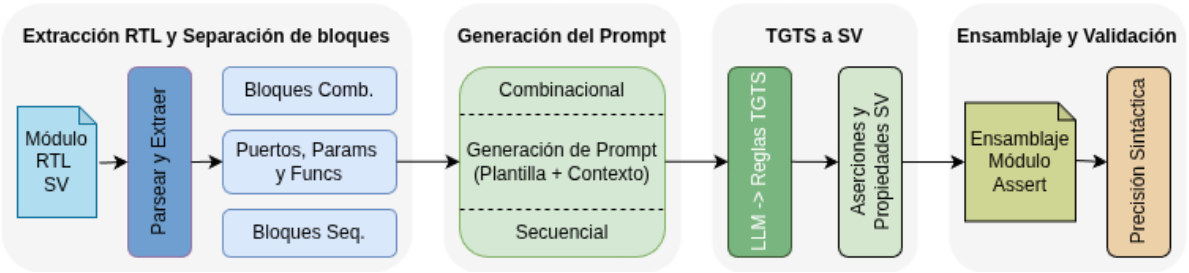


Fig. 1: Flujo de procesamiento de SVA-TGTS en cuatro etapas: Extracción de RTL y Separación de Bloques, Generación de Prompts para LLM, de TGTS a SV y Ensamblaje y Validación.

B. Generación del Prompt

Para cada bloque extraído, construimos un *prompt* que incluye tres partes: 1) la estructura de asignación condicional del bloque; 2) las señales relevantes (por ejemplo, puertos, variables internas y funciones); y 3) una instrucción para generar reglas TGTS con el siguiente formato:

```
RULE <nombre_regla>
  WHEN <precondición>
  THEN <asignación>
```

Fig. 2: Plantilla para generar reglas TGTS.

Aquí, `<precondición>` es una expresión booleana que debe cumplirse para que la regla se active, y `<assignment>` define la actualización de estado aplicada cuando la regla se dispara.

Utilizamos dos plantillas de *prompt* para distinguir entre bloques combinacionales y secuenciales:

- **Prompt combinacional**: Instruye al LLM para generar reglas sin consideraciones temporales. Cada regla corresponde a una ruta a través de la lógica condicional.
- **Prompt secuencial**: Instruye al LLM para preservar la sensibilidad al flanco de reloj y al reinicio (*reset*) en las reglas generadas, codificando las dependencias temporales en las condiciones `WHEN`.

El *prompt* se mantiene intencionadamente simple, confiando en la extracción estructurada para proporcionar el contexto suficiente. No se utilizan ejemplos *few-shot* ni técnicas de *chain-of-thought*, y la tarea del LLM se reduce a traducir las asignaciones condicionales al formato uniforme de TGTS.

C. TGTS a SV

Cada regla TGTS generada se traduce sistemáticamente en una SVA a través de una etapa de post-procesamiento que realiza la normalización de las reglas y el análisis sintáctico. En este proceso, la guardia o precondición de la regla TGTS se mapea al antecedente de la aserción, mientras que la asignación define el consecuente, preservando la semántica de transición original. La salida en bruto del LLM se limpia mediante expresiones regulares para eliminar texto superfluo y asegurar que las cláusulas `WHEN` y `THEN` de las reglas TGTS queden correctamente aisladas. Las reglas generadas que no sean TGTS válidas también se descartan en este proceso. Para los bloques combinacionales, la aserción se construye como una propiedad inmediata, como se ilustra en la Figura 3:

```
property <uuid>_assert_<id>;
    @(<lista_sensibilidad>)
    <precondición> |-> <resultado>;
endproperty
assert property (<uuid>_assert_<id>)
else $error("Error message");
```

Fig. 3: Lógica combinacional: La aserción se construye como una propiedad inmediata

De `<precondición>` se deriva de la cláusula `WHEN` de la regla, y `resultado` de la cláusula `THEN`. La lista de sensibilidad se genera a partir de todas las variables que pueden cambiar durante la ejecución; específicamente, los puertos de entrada y las variables internas que aparecen en el antecedente, además de cualquier puerto interno presente en el consecuente. Para los bloques secuenciales, la aserción se construye como una propiedad con reloj y reinicio (reset), tal como se representa en Figura 4.

```
property <uuid>_assert_<id>;
    @(<sensibilidad_reloj>)
    disable iff (<activacion_reset>)
    <precondición> |-> <resultado>;
endproperty
assert property (<uuid>_assert_<id>)
else  $error("Error message");
```

Fig. 4: Lógica secuencial: La aserción se construye como una propiedad con reloj y reinicio (reset).

De `<precondición>` y `resultado` se derivan,

de manera similar a los bloques combinacionales, de las cláusulas `WHEN` y `THEN`, respectivamente. Las señales de reloj y reinicio (reset) se infieren de la lista de sensibilidad del bloque original.

La Figura 5 ilustra un ejemplo sencillo de las tres primeras etapas del flujo propuesto. El marco superior izquierdo muestra un módulo SystemVerilog, del cual se extrae un conjunto de reglas TGTS, como se representa en el marco superior derecho. Estas reglas se traducen luego en las SVA correspondientes, que se muestran en el marco inferior.

D. Ensamblaje y Validación

Ensamblamos todas las aserciones generadas que son válidas en un único módulo SystemVerilog, incluyendo las mismas variables internas, funciones, parámetros y puertos (estos últimos se modifican todos a tipo de entrada, `input`). El módulo de aserciones ensamblado se compila utilizando `vlog` en Siemens Questa. Los errores de sintaxis se capturan y registran, pero no se realiza ninguna simulación funcional en esta etapa, ya que el enfoque principal es validar la corrección sintáctica de las aserciones generadas. El éxito de la compilación indica que las aserciones son sintácticamente válidas y que su ámbito (scope) es correcto.

V. Validación Experimental

El marco de trabajo SVA-TGTS se implementó en Python, con la inferencia del LLM gestionada a través de Ollama para garantizar un despliegue local que preserve la privacidad. Todos los experimentos se realizaron en un sistema equipado con una GPU NVIDIA L4, 1 TB de RAM DDR5 y un procesador AMD EPYC 9355P de 32 núcleos. Para cada módulo de aserciones generado, registramos estas dos métricas: 1) **tiempo de generación:** tiempo de reloj en segundos desde el envío del prompt hasta el ensamblaje completo del módulo de aserciones; 2) **estado de compilación:** resultado de aprobado/fallido en Questa, incluyendo los errores de sintaxis en caso de presentarse.

Los experimentos se evaluaron utilizando ocho LLMs de código abierto, los cuales emplean un número de parámetros que oscila entre 7B y 33B. Los modelos utilizados son: CodeGemma [15] con 7B, DeepSeek-Coder-v2 [16] con 16B, DeepSeekCoder [17] con 33B, Qwen2.5-Coder [18] con 7B, 14B y 30B, Qwen3-Coder [19] con 32B y Yi-Coder [20] con 9B. Todos los modelos se desplegaron localmente a través de Ollama con la configuración de inferencia por defecto y sin aplicar ajuste fino a ninguno de ellos.

A. Dataset

Para evaluar el marco de trabajo SVA-TGTS propuesto, se construyó un dataset curado de 200 módulos SystemVerilog específicamente para la experimentación. Todos los módulos se diseñaron con un estilo de codificación uniforme y sin incluir bucles. Esta estandarización garantiza que la fase de extracción se centre en la estructura condicional relevante para la generación de reglas TGTS, sin variaciones sintácticas superfluas que pudieran confundir la evaluación.

El dataset está organizado en cuatro categorías que representan niveles crecientes de complejidad: 1) **Módulos combinacionales (40% de los módulos)**: Módulos simples que contienen únicamente lógica combinacional. Estos incluyen sentencias `case` anidadas, constructos `casez`, rangos y múltiples casos mapeados a la misma salida; 2) **Módulos secuenciales (40% de los módulos)**: Módulos simples que contienen únicamente lógica secuencial (bloques `always` síncronos). De manera similar, incluyen sentencias `case` anidadas, constructos `casez`, rangos y múltiples casos por salida; 3) **Máquinas de Estados Finitos Mealy (15% de los módulos)**: Módulos que combinan lógica combinacional y secuencial a través de múltiples bloques `always`, representando la interacción entre la lógica de estado siguiente y la lógica de salida típica de los diseños de FSM tipo Mealy; y 4) **Módulos relacionados con la industria (5% de los módulos)**: Módulos complejos con múltiples bloques combinacionales y secuenciales, cada uno implementando funcionalidades distintas inspiradas en aplicaciones industriales reales, como un decodificador Huffman para JPEG, controladores de memoria o incluso conversión de espacio de color de vídeo, que sirven como una prueba exigente para la robustez de la generación de aserciones.

B. Experimentos

La evaluación experimental se diseñó para responder a dos preguntas de investigación primordiales: **RQ1**): *¿Cómo se compara la estrategia de prompting basada en TGTS con otros enfoques de prompting en términos de corrección sintáctica?*; y **RQ2**): *¿Cómo afecta el tamaño del modelo a la calidad de la generación de aserciones en diferentes complejidades de diseño?*. Para asegurar que las mejoras atribuidas a TGTS sean significativas, primero establecemos dos condiciones de prompting de referencia:

El valor de referencia Simple (S) utiliza un prompt mínimo que consiste únicamente en el módulo RTL en bruto y una instrucción para "generar el módulo de aserciones correspondiente", formateado dentro de un bloque de código SystemVerilog. Esta configuración refleja el uso más simple y comúnmente intentado de los LLM para la generación de SVA, y deliberadamente no proporciona pistas estructurales ni ejemplos, asegurando que su rendimiento no sea subestimado.

```
module case_simple_example (
    input  logic       addr,
    input  logic       sub_op,
    input  logic [1:0] mode,
    input  logic       data,
    output logic       result
);
  always_comb
    begin case (addr)
      1'b0: case (sub_op)
              1'b0:   result = data;
              1'b1:   result = ~data;
            endcase
      1'b1: casex (mode)
              2'b0x, 2'b1x:
                result = data & sub_op;
              default: result = 1'b0;
            endcase
    endcase end
endmodule
```

```
RULE uuid_1_simple_example
  WHEN (addr == 1'b0 && sub_op == 1'b0)
  THEN result == data

RULE uuid_2_simple_example
  WHEN (addr == 1'b0 && sub_op == 1'b1)
  THEN result == ~data

RULE uuid_3_simple_example
  WHEN (addr == 1'b1 && mode == 2'b0x)
  THEN result == (data & sub_op)

RULE uuid_4_simple_example
  WHEN (addr == 1'b1 && mode == 2'b1x)
  THEN result == (data & sub_op)

RULE uuid_5_simple_example
  WHEN (addr == 1'b1 && !(mode == 2'b0x || mode == 2'b1x))
  THEN result == 1'b0
```

```
module case_simple_example_asserts  (input logic addr, input logic sub_op, input logic [1:0] mode,
  input logic data, input logic result);
  uuid_1_simple_example: assert property (@(addr, sub_op, data)
    (addr == 1'b0 && sub_op == 1'b0) |-> result == data) else $error("Error uuid_1_simple_example");
  uuid_2_simple_example: assert property (@(addr, sub_op, data)
    (addr == 1'b0 && sub_op == 1'b1) |-> result == ~data) else $error("Error uuid_2_simple_example");
  uuid_3_simple_example: assert property (@(addr, mode, sub_op, data)
    (addr == 1'b1 && mode == 2'b0x) |-> result == (data & sub_op)) else $error("Error uuid_3_simple_example");
  uuid_4_simple_example: assert property (@(addr, mode, sub_op, data)
    (addr == 1'b1 && mode == 2'b1x) |-> result == (data & sub_op)) else $error("Error uuid_4_simple_example");
  uuid_5_simple_example: assert property (@(addr, mode)
    (addr == 1'b1 && !(mode == 2'b0x || mode == 2'b1x)) |-> result == 1'b0)
    else $error("Error uuid_5_simple_example");
endmodule
```

Fig. 5: Ejemplo de SVA-TGTS: El marco superior izquierdo muestra el código original; el marco superior derecho contiene las reglas TGTS generadas a partir del código original; el marco inferior presenta el módulo de aserciones generado.

El valor de referencia Complejo (C) aumenta el RTL con un prompt estructurado que contiene (i) una definición de rol, (ii) una lista explícita de reglas obligatorias para la generación de aserciones, (iii) cinco pares (few-shot) de (RTL, SVA) y (iv) restricciones de formato y una plantilla para el módulo final. Esto refleja los prompts de mejores prácticas diseñados manualmente en trabajos previos y representa una referencia significativamente más sólida que la Simple, evitando al mismo tiempo cualquier entrenamiento o ajuste fino.

Finalmente, la estrategia TGTS (T) sustituye el RTL completo por bloques combinacionales y secuenciales extraídos e instruye al modelo para que emita únicamente reglas TGTS siguiendo una gramática estricta. Esto reduce la ambigüedad y produce una representación intermedia determinista a partir de la cual se derivan las SVA. Las tres estrategias se evalúan en ocho LLM para aislar el efecto de la estructura del prompt y el tamaño del modelo en la calidad de las SVA.

C. Resultados

1) Análisis de precisión sintáctica: Los resultados de la Tabla I demuestran claramente que la estrategia basada en TGTS supera de manera consistente tanto al enfoque simple como al complejo en todas las categorías de referencia. Una inspección más detallada revela comportamientos distintos dependientes del modelo: En la estrategia simple, los modelos más pequeños como CodeGemma-7B y Yi-Coder-9B fallan frecuentemente al generar aserciones sintácticamente válidas, obteniendo un 0% de correctas en varias categorías, particularmente en los bancos de pruebas combinacionales e industriales. La estrategia compleja mitiga parcialmente estos problemas, con mejoras notables en la lógica secuencial (por ejemplo, Qwen2.5-Coder-14B alcanzando un 87.5%), aunque sigue siendo inconsistente y a menudo tiene un rendimiento inferior en tareas combinacionales, lo que indica que el aumento de la complejidad del prompt por sí solo no restringe suficientemente el espacio de generación. Cabe destacar que todas las variantes de Qwen2.5-Coder (7B, 14B y 32B) logran un 100% de corrección sintáctica en los bancos de pruebas combinacionales, manteniendo un rendimiento sólido y estable en las categorías secuenciales y de FSM. Incluso los modelos más débiles se benefician sustancialmente:

TABLA I: Análisis de Corrección Sintáctica y Tiempo de Generación por categoría de subdataset en la estrategias S, C y T en pass@1.

LLM	Combinacional			Sequencial			FSM			Industria		
	S	C	T	S	C	T	S	C	T	S	C	T
CodeGemma-7B	0.0	0.0	**82.5**	0.0	41.25	**92.5**	0.0	**43.33**	**43.33**	0.0	0.0	**20.0**
DeepSeek-Coder-V2-16B	3.75	22.5	**87.5**	37.5	78.75	**85.0**	46.67	73.33	**90.0**	10.0	10.0	**50.0**
Qwen2.5-Coder-7B	5.0	0.0	**100.0**	35.0	68.75	**97.5**	36.67	70.0	**73.33**	0.0	30.0	**60.0**
Qwen2.5-Coder-14B	52.5	0.0	**100.0**	72.5	**87.5**	**87.5**	63.33	**96.67**	**96.67**	40.0	40.0	**80.0**
Qwen2.5-Coder-32B	18.75	0.0	**100.0**	78.75	**95.0**	91.25	90.0	80.0	**100.0**	**60.0**	40.0	70.0
DeepSeek-Coder-33B	23.75	21.25	**97.5**	36.25	**93.75**	72.5	23.33	**76.67**	50.0	20.0	30.0	**40.0**
Qwen3-Coder-30B	10.0	0.0	**96.25**	57.5	**96.25**	**96.25**	56.67	90.0	**93.33**	30.0	40.0	**60.0**
Yi-Coder-9B	1.25	10.0	**82.5**	0.0	60.0	**92.5**	0.0	23.33	**43.33**	0.0	10.0	**20.0**

TABLA II: Tiempo de generación (segundos) por modelo y subdataset para las estrategias S, C, y T S en pass@1.

LLM	Combinacional			Sequencial			FSM			Industria		
	S	C	T	S	C	T	S	C	T	S	C	T
CodeGemma-7B	**3.77**	9.31	4.45	**7.60**	10.09	11.16	34.84	**16.40**	37.17	50.37	**22.08**	89.98
DeepSeek-Coder-V2-16B	6.82	6.38	**4.57**	**9.15**	12.72	12.29	**27.00**	35.26	33.28	89.51	**58.58**	100.45
Qwen2.5-Coder-7B	6.01	6.30	**3.13**	**7.23**	11.16	10.57	**23.35**	28.32	90.97	62.36	**38.28**	91.10
Qwen2.5-Coder-14B	15.37	13.27	**6.58**	**16.02**	29.38	33.17	51.10	49.64	**48.56**	151.74	**107.81**	189.15
Qwen2.5-Coder-32B	41.06	27.00	**15.31**	**47.57**	55.12	136.38	121.59	142.02	**104.94**	284.41	**245.07**	446.50
DeepSeek-Coder-33B	123.94	33.02	**16.10**	123.36	**39.70**	43.20	**95.12**	99.12	134.16	234.30	**151.67**	335.95
Qwen3-Coder-30B	6.13	4.84	**3.14**	**8.01**	11.73	48.09	**21.49**	22.46	231.85	314.44	**33.96**	581.66
Yi-Coder-9B	11.23	8.44	**4.45**	13.97	11.21	**11.16**	23.68	**20.54**	37.17	**24.10**	32.74	89.98

CodeGemma-7B mejora del 0% al 82.5%, y Yi-Coder-9B muestra ganancias similares, lo que resalta que TGTS reduce efectivamente la dependencia de la escala del modelo. Además, los modelos más grandes como Qwen3-Coder-30B y DeepSeek-Coder-33B exhiben un comportamiento más consistente entre categorías bajo TGTS, evitando la variabilidad observada en las estrategias anteriores. En general, estos resultados validan la hipótesis central de este trabajo: la introducción de una representación formal y estructurada como TGTS cierra eficazmente la brecha semántica entre RTL y SVA, permitiendo que incluso los LLM con recursos limitados produzcan resultados fiables y sintácticamente correctos.

2) Análisis de tiempo de ejecución: Los resultados del tiempo de ejecución en la Tabla II revelan una sobrecarga moderada asociada con la estrategia TGTS, particularmente para modelos más grandes y bancos de pruebas más complejos, como los casos de FSM e industriales. Por ejemplo, en la categoría industrial, Qwen3-Coder-30B exhibe un aumento sustancial en la latencia bajo TGTS (581,66 segundos) en comparación con la estrategia compleja (33,96 segundos), mientras que Qwen2.5-Coder-32B también aumenta de 245,07 segundos (compleja) a 446,50 segundos (TGTS). Sin embargo, este incremento no es uniforme en todas las configuraciones y, en varios casos, TGTS logra tiempos de ejecución competitivos o incluso menores, particularmente en tareas combinacionales donde modelos como Qwen2.5-Coder-7B (3,13 segundos) y Qwen3-Coder-30B (3,14 segundos) alcanzan su latencia mínima bajo el enfoque propuesto. Los modelos más pequeños también se mantienen relativamente estables, con CodeGemma-7B y Yi-Coder-9B mostrando solo aumentos marginales en comparación con las estrategias anteriores, lo que sugiere que la sobrecarga es más pronunciada en arquitecturas más grandes. Es importante destacar que una mayor corrección sintáctica a menudo se correlaciona con una mayor latencia de generación en los bancos de pruebas complejos e industriales, lo que indica un compromiso entre la profundidad del razonamiento y el coste computacional. No obstante, este intercambio es favorable: la mejora en la corrección lograda por TGTS reduce significativamente la necesidad de prompting iterativo, regeneración o depuración manual, aspectos que no se capturan en las métricas de tiempo bruto. Al considerar el flujo de trabajo de verificación completo, la naturaleza estructurada de TGTS permite un post-procesamiento determinista y resultados más fiables, lo que probablemente conduce a una reducción del tiempo total de respuesta a pesar de pasos de inferencia individuales ligeramente más largos. Por lo tanto, la sobrecarga de tiempo observada está justificada por las ganancias sustanciales en robustez y corrección, lo que refuerza a TGTS como una solución práctica para una verificación eficiente en las propias instalaciones (on-premise).

Para cuantificar aún más la relación entre la inversión computacional y la calidad de los resultados, la Tabla III presenta el "Ratio de Eficiencia" ponderado,

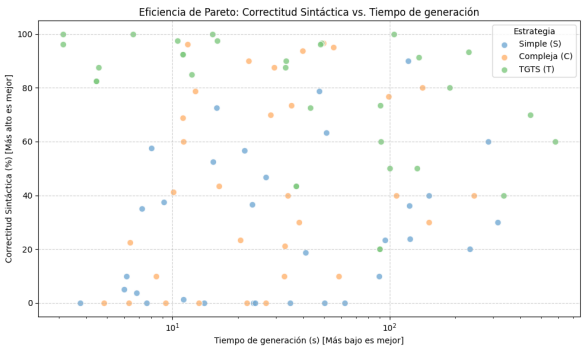


Fig. 6: Análisis de la frontera de Pareto: Corrección sintáctica frente al tiempo de generación en todos los experimentos. La región superior izquierda representa el equilibrio óptimo entre una alta precisión y una baja latencia.

definido como el tiempo promedio requerido para lograr un incremento del 1% en la corrección sintáctica. Para asegurar que las métricas reflejen la distribución real de las entradas del dataset, las cifras se calculan utilizando un promedio ponderado: Combinacionales (40%), Secuenciales (40%), FSM (15%) e Industriales (5%).

TABLA III: Ratio de Eficiencia: Tiempo promedio (s) por cada 1% de corrección sintáctica en todas las categorías.

Estrategia	Corrección media (%) ↑	Tiempo medio (s) ↓	Cost (s / 1% Sintaxis)
Simple (S)	28.6%	37.4s	1.31
Compleja (C)	45.4%	26.6s	0.59
TGTS (T)	**86.6%**	**43.7s**	**0.50**

Esta métrica revela que la estrategia TGTS (T) proporciona el equilibrio óptimo entre el rendimiento y el gasto de recursos. A diferencia de los métodos de referencia, TGTS logra el coste computacional más bajo por unidad de mejora sintáctica (0,50 s/%), lo que la hace más eficiente tanto que la estrategia Compleja (0,59 s/%) como que el enfoque Simple (1,31 s/%). Esto indica que el razonamiento estructurado inherente a TGTS no solo añade latencia, sino que está convirtiendo de manera activa y eficiente el tiempo de inferencia adicional en mejoras sintácticas medibles, a una tasa que supera a las estrategias de prompting más sencillas.

La relación entre estas variables se resume visualmente en la frontera de Pareto mostrada en la Figura 6. Al graficar los resultados brutos de todos los experimentos anteriores frente al tiempo de generación en una escala logarítmica, observamos que la Estrategia T define consistentemente el límite superior del gráfico, o la "frontera eficiente". Aunque los puntos de datos brutos para la categoría Industrial se desplazan hacia la derecha, reflejando el mayor esfuerzo computacional requerido para la lógica compleja, los resultados de TGTS en la mayoría de los casos (combinacionales y secuenciales) se agrupan en la esquina superior izquierda. Esta separación vertical confirma que la estrategia TGTS proporciona un salto significativo en la precisión (alcanzando un promedio ponderado del 86,6%) que justifica plenamente el desplazamiento horizontal marginal en el coste de tiempo. Al dominar el cuadrante superior izquierdo, TGTS demuestra ser la solución más robusta para la fiabilidad zero-shot en la generación de RTL convencional.

D. Discusión

Los datos sugieren que el recuento de parámetros no es el único determinista del éxito; Qwen2.5-Coder 7B superó a modelos más grandes como DeepSeek-Coder 33B tanto en sintaxis (100% frente a 97,5%) para tareas combinatorias. Esto valida que las estrategias de pipeline dirigidas como TGTS pueden cerrar la brecha entre los modelos de pequeña escala y los complejos requisitos de verificación de hardware. Los modelos más grandes (32B–33B) generalmente lograron una mayor corrección sintáctica en bancos de pruebas complejos, pero las ganancias de rendimiento no fueron proporcionales al aumento del tiempo de inferencia. La variante Qwen2.5-Coder 14B logró resultados competitivos (80% de casos correctos en bancos de pruebas industriales, 96,67% en FSM) con requisitos de recursos significativamente menores que el modelo 32B (189 s frente a 446 s de tiempo promedio de generación en bancos de pruebas industriales). Esta observación respalda la viabilidad de los modelos de tamaño moderado para el despliegue práctico.

En los ejemplos de FSM y de escala industrial, la principal limitación observada fue la incapacidad de los modelos para generar consistentemente reglas TGTS analizables. Los problemas comunes incluyeron la alucinación de nombres de variables, la omisión de constructos temporales y la introducción de estructuras de control no admitidas, como sentencias IF/ELSE, a pesar de las restricciones explícitas del prompt. Cabe destacar que estos errores no estuvieron presentes en diseños combinacionales o secuenciales más pequeños, lo que sugiere que la fiabilidad del modelo se degrada a medida que aumentan la complejidad del diseño y la longitud del código.

VI. Conclusiones y Trabajo Futuro

Este trabajo presenta SVA-TGTS, un flujo automatizado para la generación de aserciones SystemVerilog en el que se aprovechan los LLM específicamente para la extracción de reglas semánticas, mientras que todos los pasos posteriores para la construcción de aserciones

se realizan de manera determinista. Esta separación permite un flujo de trabajo estructurado y reproducible, demostrando la viabilidad de derivar conjuntos de aserciones significativos directamente desde el DUT sin intervención manual.

Como trabajo futuro, planeamos ampliar el *framework* con una etapa de generación automática de bancos de pruebas (*testbenches*) para permitir la evaluación sistemática de las aserciones generadas a través de métricas de cobertura, permitiendo cuantificar la efectividad y completitud de las mismas. Además, los esfuerzos futuros se centrarán en mejorar la extracción de la semántica temporal para la lógica secuencial mediante estrategias de *prompting* mejoradas, datos de entrenamiento especializados o enfoques híbridos que combinen los LLM con métodos formales. También se prestará especial atención a mejorar la robustez en diseños complejos, donde los módulos grandes han mostrado una mayor tendencia a inducir alucinaciones y a la generación de reglas incoherentes. Para abordar esto, investigaremos estrategias como el *prompting* estructurado, representaciones intermedias y la descomposición de diseños grandes en componentes más pequeños antes de la extracción de reglas. Finalmente, extender la evaluación a diseños más complejos basados en FSM y a escala industrial, así como mejorar la robustez frente a especificaciones incompletas o ambiguas, sigue siendo una dirección importante.

AGRADECIMIENTOS

Esta investigación está financiada parcialmente por el Ministerio de Ciencia e Innovación de España bajo el proyecto OASIS-GEMINI (PID2023-148285OB-C41), en colaboración con el Ministerio de Transformación Digital y de la Función Pública y la Unión Europea NextGenerationEU/PRTR a través de las ayudas PERTE Chip (Cátedra UCLM, TSI-069100-2023-0014), por la Junta de Comunidades de Castilla-La Mancha bajo el proyecto REACH (SBPLY/24/ 180225/000105) y RURALCARE SBPLY/24/180225/000158 (INNOCAM) y por el programa Horizon Europe con MAYA (GA Nº 101213323).

REFERENCES

[1] H. Foster, "2024 wilson research group fpga functional verification trend report," Siemens EDA, Tech. Rep., 2024, comprehensive industry study on functional verification trends and challenges. [Online]. Available: https://resources.sw.siemens.com/en-US/white-paper-2024-wilson-research-group-fpga-functional-verification-trend-report/

[2] S. Paul, A. Banerjee, S. Ghosh, S. Surendran, and R. K. Gajavelly, " LISA: LLM Informed Systemverilog Assertion generation with RAG and Chain-of-Thought ," in *2025 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. Los Alamitos, CA, USA: IEEE Computer Society, Jul. 2025, pp. 1–6. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ISVLSI65124.2025.11130226

[3] R. Kande, H. Pearce, B. Tan, B. Dolan-Gavitt, S. Thakur, R. Karri, and J. Rajendran, "(security) assertions by large language models," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 4374–4389, 2024.

[4] Xenoss. (2026) Fine-tuning llms at scale: Cost optimization strategies. Accessed: 3 April 2026. [Online]. Available: https://xenoss.io/blog/fine-tuning-llm-cost-optimization

[5] A. B. Rauzy, "Guarded transition systems: A new states/events formalism for reliability studies," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, vol. 222, no. 4, pp. 495–505, 2008.

[6] L. Chen and Y. C. el. al., *Science China Information Sciences*, vol. 67, no. 10, Oct. 2024, publisher Copyright: © Science China Press 2024.

[7] M. Orenes-Vera, A. Manocha, D. Wentzlaff, and M. Martonosi, "Autosva: Democratizing formal verification of rtl module interactions," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 535–540.

[8] S. Rajabi, C. Yang, and S. Patnaik, "Stellar: Structure-guided llm assertion retrieval and generation for formal verification," in *Accepted at the 63rd Design Automation Conference (DAC 2026)*, 2026.

[9] Y. Bai, G. B. Hamad, S. Suhaib, and H. Ren, "Assertionforge: Enhancing formal verification assertion generation with structured representation of specifications and rtl," 2025. [Online]. Available: https://arxiv.org/abs/2503.19174

[10] R. Guo, A. Ayalasomayajula, H. Li, J. Zhou, S. K. Saha, and F. Farahmandi, "Svagent: Ai agent for hardware security verification assertion," 2025. [Online]. Available: https://arxiv.org/abs/2507.16203

[11] A. Menon, S. Miftah, S. Kundu, S. Kundu, A. Srivastava, A. Raha, G. Sonnenschien, S. Banerjee, D. Mathaikutty, and K. Basu, "Enhancing large language models for hardware verification: A novel systemverilog assertion dataset," *ACM Trans. Des. Autom. Electron. Syst.*, Aug. 2025, just Accepted. [Online]. Available: https://doi.org/10.1145/3764934

[12] R. S. Nikhil, *Bluespec: A General-Purpose Approach to High-Level Synthesis Based on Parallel Atomic Transactions*. Dordrecht: Springer Netherlands, 2008, pp. 129–146. [Online]. Available: https://doi.org/10.1007/978-1-4020-8588-8_8

[13] J. Choi, M. Vijayaraghavan, B. Sherman, A. Chlipala, and Arvind, "Kami: a platform for high-level parametric hardware specification and its modular verification," *Proc. ACM Program. Lang.*, vol. 1, no. ICFP, Aug. 2017. [Online]. Available: https://doi.org/10.1145/3110268

[14] Siemens Digital Industries Software, *Questa One Smart Verification Solution*, Siemens Digital Industries Software, 2023.4, version 2023.4. [Online]. Available: https://eda.sw.siemens.com/en-US/ic/questa-one/

[15] C. Team *et al.*, "Codegemma: Open code models based on gemma," *arXiv preprint arXiv:2406.11409*, 2024. [Online]. Available: https://arxiv.org/abs/2406.11409

[16] D. Team *et al.*, "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model," *arXiv preprint arXiv:2405.04434*, 2024. [Online]. Available: https://arxiv.org/abs/2405.04434

[17] D. T. et. al, "Deepseek-coder: When the large language model meets programming – the rise of code intelligence," *arXiv preprint arXiv:2401.14196*, 2024. [Online]. Available: https://arxiv.org/abs/2401.14196

[18] B. Hui *et al.*, "Qwen2.5-coder technical report," *arXiv preprint arXiv:2409.12186*, 2024. [Online]. Available: https://arxiv.org/abs/2409.12186

[19] R. Cao and 19 other authors, "Qwen3-coder-next technical report," *arXiv preprint arXiv:2603.00729*, 2026. [Online]. Available: https://arxiv.org/abs/2603.00729

[20] A. Company, "Meet yi-coder: A small but mighty llm for code," September 2024. [Online]. Available: https://01-ai.github.io/blog.html?post=en/2024-09-05-A-Small-but-Mighty-LLM-for-Code.md

FRAGATA: Recuperación semántica de tickets de soporte HPC mediante RAG híbrido sobre 20 años de histórico de Request Tracker

Santiago Paramés Estévez[1], Nicolás Filloy Montesino[2], Jorge Fernández Fabeiro[1] y José Carlos Mouriño Gallego[1]



Resumen— **El soporte técnico de un centro de supercomputación acumula, a lo largo de décadas, un gran volumen de incidencias resueltas que constituyen conocimiento operacional crítico. En el Centro de Supercomputación de Galicia (CESGA) este histórico se gestiona desde hace más de veinte años con Request Tracker (RT), cuyo buscador integrado presenta limitaciones importantes que dificultan la reutilización del conocimiento por parte de los técnicos. Este trabajo presenta FRAGATA, un sistema de búsqueda semántica de tickets que combina técnicas de recuperación de información modernas con el histórico completo de RT. El sistema permite encontrar incidencias pasadas relevantes independientemente del idioma, la presencia de errores tipográficos o la formulación concreta de la consulta. La arquitectura se despliega sobre la infraestructura del CESGA, soporta actualización incremental sin interrupciones del servicio y aprovecha el supercomputador FinisTerrae III para las etapas más costosas. Los resultados preliminares muestran una mejora cualitativa sustancial frente al buscador nativo de RT.**

Palabras clave— **RAG, recuperación de información, automatización, búsqueda semántica, HPC, Request Tracker, CESGA.**

I. Introducción

LOS centros de computación de altas prestaciones (HPC) operan plataformas heterogéneas en las que conviven cientos de aplicaciones científicas, compiladores, librerías paralelas, planificadores de recursos y sistemas de almacenamiento distribuido. En este entorno, el equipo de soporte técnico actúa como interfaz crítico entre los usuarios investigadores y la infraestructura, resolviendo incidencias que abarcan desde errores de compilación de un código MPI hasta cuotas de almacenamiento, configuración de entornos Conda o fallos de red en nodos GPU. El Centro de Supercomputación de Galicia (CESGA) gestiona este flujo desde hace más de dos décadas mediante Request Tracker (RT) [1], herramienta libre de *ticketing* ampliamente utilizada en el sector.

RT ofrece dos vías principales de interacción: una interfaz web (exclusivamente para técnicos) y el correo electrónico. Los usuarios del CESGA interactúan con el sistema exclusivamente a través del correo, enviando y respondiendo a mensajes que RT convierte automáticamente en tickets y respuestas asociadas. Esta dinámica tiene implicaciones directas sobre la calidad del texto almacenado, como se detalla en la Sección III-B.

El histórico acumulado por el CESGA en RT supera los veinte años de conversaciones entre técnicos y usuarios y constituye, en la práctica, una memoria operacional de incalculable valor: un porcentaje significativo de las incidencias que se reciben hoy son variantes de problemas resueltos años atrás. Sin embargo, la versión 4.4.1 de RT presenta limitaciones severas en su buscador integrado: no indexa el contenido completo de los tickets, distingue mayúsculas y minúsculas, no tolera errores tipográficos, no normaliza variantes morfológicas y carece de cualquier noción de similitud semántica. El resultado es que tanto los técnicos veteranos como los de nueva incorporación tienen serias dificultades para localizar casos previos relevantes, lo que provoca duplicación de esfuerzos, pérdida de conocimiento institucional y aumento del tiempo medio de resolución.

Este trabajo presenta Fragata, un sistema de recuperación semántica de tickets diseñado e implementado en el CESGA para mitigar estas limitaciones. Fragata aplica el paradigma *Retrieval-Augmented Generation* (RAG) [2], un enfoque que mejora la calidad de las respuestas de los sistemas de búsqueda recuperando primero los documentos más relevantes de una base de conocimiento y presentándolos como contexto. En concreto, el sistema combina recuperación densa basada en *embeddings*, representaciones vectoriales numéricas del significado del texto, con recuperación léxica clásica BM25 [3] y *reranking* mediante *cross-encoders* [4]. El sistema integra además fuentes complementarias (documentación técnica del centro, manuales de aplicaciones científicas y repositorios) y se despliega sobre una arquitectura híbrida máquina virtual + supercomputador FinisTerrae III, con ingesta incremental y recarga en caliente del motor de búsqueda.

Las contribuciones principales son: (i) el diseño de

[1]Centro de Supercomputación de Galicia (CESGA), Santiago de Compostela, e-mail: sparames@cesga.es. Código disponible en: https://github.com/s-parames/fragata.
[2]Universidade de Vigo.

un protocolo reproducible de extracción, normalización y *chunking* del histórico SQL de RT; (ii) una arquitectura de recuperación híbrida con fusión ponderada y *reranking* sensible al tipo de consulta; (iii) un mecanismo de *hot-swap* del motor que garantiza disponibilidad continua durante reindexaciones; y (iv) la integración operacional con un planificador HPC para descargar etapas costosas de la ingesta sin penalizar la latencia del servicio.

II. Trabajo relacionado

A. Recuperación de información neuronal

La recuperación de información ha experimentado una transformación profunda en los últimos años gracias a los modelos basados en *transformers* [5]. Estas arquitecturas de redes neuronales, en las que se basan modelos como BERT [6], aprenden a representar textos como vectores numéricos densos (*embeddings*) que capturan el significado semántico de las palabras y las frases. Dos textos con significado similar obtienen vectores próximos en el espacio, lo que permite localizar documentos relevantes midiendo distancias vectoriales en lugar de depender de coincidencias exactas de palabras. En particular, Sentence-BERT [7] adaptó BERT para producir *embeddings* a nivel de oración de forma eficiente, y los sistemas de recuperación densa como DPR [8] demostraron que esta aproximación puede superar a la búsqueda léxica tradicional en tareas de pregunta-respuesta.

Sin embargo, la recuperación puramente densa presenta debilidades en consultas con terminología muy específica, nombres propios o identificadores técnicos, donde la coincidencia exacta de palabras sigue siendo determinante. Por ello, los enfoques híbridos que combinan BM25, un algoritmo clásico de recuperación basado en frecuencias de términos, con recuperación densa [9] se han consolidado como estado del arte práctico. Estos sistemas se benefician especialmente de una etapa posterior de *reranking*, en la que un *cross-encoder* [4] evalúa conjuntamente la consulta y cada candidato para reordenar los resultados con mayor precisión.

El paradigma RAG [2], [10] integra esta cadena de recuperación en un flujo completo: dado un texto de consulta, el sistema recupera los documentos más relevantes de una base de conocimiento. Este enfoque permite aprovechar grandes volúmenes de información sin necesidad de reentrenar modelos, lo que lo hace especialmente adecuado para dominios con conocimiento acumulado como el soporte técnico.

B. Automatización del soporte técnico

La aplicación de técnicas de procesamiento de lenguaje natural a sistemas de *ticketing* ha sido explorada en distintos contextos. Potharaju et al. [11] analizaron córpora de tickets de red para inferir problemas de forma automática, mientras que Zhou et al. [12] propusieron métodos de aprendizaje automático para el enrutamiento inteligente de incidencias. Estos trabajos se centran en clasificación y triaje, pero no abordan la recuperación semántica sobre el histórico acumulado.

En el ámbito HPC específicamente, los esfuerzos publicados se han centrado mayoritariamente en sistemas de monitorización y diagnóstico automático de trabajos (*jobs*), mientras que la explotación sistemática del conocimiento contenido en históricos de soporte ha recibido menor atención. Fragata se sitúa precisamente en este nicho: recuperación semántica de calidad sobre un corpus heterogéneo dominado por correo electrónico técnico real, con requisitos operativos de un servicio en producción.

III. Fuentes y preparación de datos

La Fig. 1 muestra el esquema general del *pipeline* de procesado de datos, que parte de la extracción directa del histórico de RT y culmina en la generación de representaciones vectoriales (*embeddings*) que permiten la búsqueda semántica.

A. Volcado del histórico de RT

Las limitaciones del buscador de RT obligaron a abordar el problema desde fuera de la herramienta. Se desarrollaron consultas SQL específicas contra la base de datos relacional de RT 4.4.1 que extraen, para cada ticket, los metadatos relevantes (identificador, cola, departamento, fechas) y la totalidad de los mensajes (*transactions* y *attachments* de tipo texto) que componen el hilo de conversación entre técnicos y usuarios. El volcado abarca aproximadamente veinte años de actividad y se materializa en ficheros JSONL[1], formato elegido por su simplicidad de procesado en *streaming* y su buen ajuste a *pipelines* incrementales.

B. Normalización y chunking

Como se ha indicado, la mayoría de las interacciones en RT se producen a través del correo electrónico. Cuando un usuario responde a un mensaje, el cliente de correo incluye automáticamente el texto de los mensajes anteriores del hilo de conversación. Esto provoca que los mensajes en bruto almacenados en cada ticket contengan referencias a mensajes previos, dando lugar a una redundancia acumulativa: la misma información aparece repetida parcial o totalmente en múltiples entradas del hilo. Además, los mensajes contienen ruido adicional característico del correo electrónico: cabeceras citadas, firmas personales, *banners* institucionales y formateo inconsistente.

La fase de normalización elimina esta información redundante y el ruido asociado, aplicando una secuencia de operaciones: decodificación de caracteres, eliminación de cabeceras y firmas conocidas, supresión de las citas de mensajes anteriores en el hilo y colapso de espacios. El objetivo es presentar al sistema de recuperación una única vez la información

[1] JSONL (*JSON Lines*) es un formato de texto en el que cada línea del fichero constituye un objeto JSON independiente y válido. A diferencia de un fichero JSON convencional, JSONL permite el procesado línea a línea (*streaming*) sin necesidad de cargar el fichero completo en memoria, lo que lo hace especialmente adecuado para *pipelines* de datos incrementales.

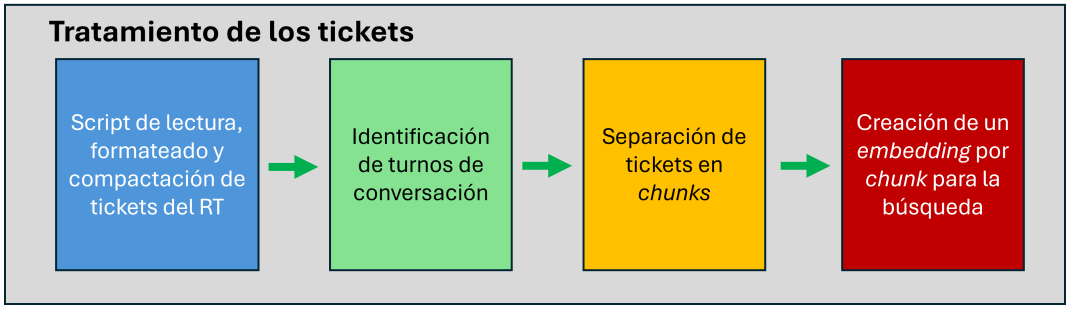


Fig. 1: Esquema del *pipeline* de procesado de tickets: desde la extracción SQL del histórico de RT hasta la generación de *embeddings* para la búsqueda semántica.

contenida en cada ticket, sin duplicaciones que puedan distorsionar los resultados.

Tras la normalización se aplica un proceso de *chunking* que segmenta cada conversación en fragmentos solapados de longitud acotada. Esta segmentación es necesaria porque los modelos de *embeddings* tienen un límite en la longitud del texto que pueden procesar, y fragmentos más cortos y focalizados producen representaciones vectoriales más precisas. Cada *chunk* preserva metadatos de identidad (`ticket_id`, `conversation_id`, `chunk_id`), de filtrado (`department`, `last_updated`, `source_type`) y de trazabilidad de ingesta.

C. Fuentes complementarias

Además del histórico de tickets, Fragata tiene la capacidad de ingerir documentación técnica externa relevante para los usuarios del CESGA: páginas web del propio centro, manuales de aplicaciones científicas en PDF y documentación de repositorios (READMEs y *wikis* de GitHub/GitLab). Cada fuente dispone de un *pipeline* dedicado que produce filas en el mismo documento JSONL, lo que permite indexarlas conjuntamente con los tickets y devolver resultados unificados.

IV. Arquitectura de recuperación híbrida

El núcleo de Fragata es un motor de recuperación implementado en Python sobre FastAPI[2], instanciado a partir de una configuración declarativa (`config/rag.yaml`) y servido mediante un gestor de ciclo de vida (`EngineManager`) que se describe en la Sección V.

A. Inicialización y artefactos

En arranque, el motor detecta el dispositivo de cómputo disponible: CPU o GPU (a través de la plataforma CUDA de NVIDIA). El uso de GPU acelera significativamente el cálculo de *embeddings* y las búsquedas vectoriales, reduciendo los tiempos de respuesta y de indexación. A continuación se carga el modelo `paraphrase-multilingual-MiniLM-L12-v2` de la familia `sentence-transformers` [7], que produce *embeddings* de 384 dimensiones con soporte para más de 50 idiomas, lo que lo hace especialmente adecuado para el corpus trilingüe del CESGA.

El motor reconstruye el índice FAISS [13][3] desde la información en disco. FAISS permite buscar, entre millones de vectores, los más próximos a un vector de consulta dado, lo que constituye la base de la recuperación semántica: encontrar los fragmentos de texto cuyo significado es más similar al de la pregunta del usuario. Para evitar duplicar el corpus en memoria, los documentos se reconstruyen desde el almacén interno de FAISS en lugar de releer el JSONL completo.

En paralelo se construye un *retriever* BM25 [3][4] sobre el mismo corpus y se carga el *reranker* `mmarco-mMiniLMv2-L12-H384-v1` [14], un *cross-encoder* entrenado sobre mMARCO, la versión multilingüe del corpus de referencia MS MARCO. Este modelo evalúa conjuntamente la consulta y cada documento candidato para producir una puntuación de relevancia más precisa que la recuperación inicial.

B. Variantes de consulta

Antes de la recuperación, el motor genera variantes ponderadas de la consulta original con el objetivo de maximizar la probabilidad de encontrar documentos relevantes. Las variantes incluyen: la forma canónica normalizada de la consulta; una corrección de un único error de edición cuando se detecta un candidato frecuente en el vocabulario del corpus; una expansión por intención que añade términos asociados al tipo de pregunta (definición, instalación, contenedores, error o *troubleshooting*); y una traducción mediante un diccionario curado de términos español/inglés.

Esta última variante aborda un aspecto característico del entorno del CESGA: el trilingüismo. Los tickets de soporte contienen texto en español, inglés y gallego, ya que los usuarios emplean indistintamente estos tres idiomas en sus comunicaciones. La traducción de términos clave al inglés, idioma predominante

[2]FastAPI (`https://fastapi.tiangolo.com`) es un *framework* moderno de Python para la construcción de APIs web de alto rendimiento, con soporte nativo para operaciones asíncronas y generación automática de documentación.

[3]FAISS (*Facebook AI Similarity Search*, `https://github.com/facebookresearch/faiss`) es una librería de código abierto desarrollada por Meta para la búsqueda eficiente de similitud entre vectores densos a gran escala.

[4]BM25 (*Best Matching 25*) es un algoritmo clásico de recuperación de información que puntúa documentos según la frecuencia de aparición de los términos de la consulta, ajustada por la longitud del documento. A diferencia de los *embeddings*, opera sobre coincidencias exactas de palabras.

en la documentación técnica, permite estandarizar la representación de las consultas y mejorar la coincidencia con los fragmentos del corpus, independientemente del idioma en que fueron redactados originalmente. El gallego, debido a su proximidad lingüística con el español, se beneficia implícitamente tanto de las variantes en español como de los *embeddings* multilingües empleados por el sistema. Esta diversificación incrementa el *recall* en consultas cortas o ambigüas y aporta robustez frente a la heterogeneidad lingüística del corpus.

C. Recuperación densa, léxica y fusión

Para cada variante se ejecutan en paralelo dos canales: una búsqueda semántica sobre FAISS (`semantic_k` candidatos) y una búsqueda BM25 (`lexical_k`). Los tamaños de cada conjunto de candidatos se ajustan dinámicamente cuando la consulta es muy corta, cuando se aplica un filtro temporal o cuando se restringe a un departamento concreto, para compensar la posterior poda. La fusión de los resultados de ambos canales emplea *Weighted Reciprocal Rank Fusion* (WRRF) [15]:

$$s(d) \;=\; \sum_{r \in R} w_r \cdot \frac{1}{k_{\mathrm{rrf}} + \mathrm{rank_r(d)}}, \qquad (1)$$

donde w_r combina el peso global del canal (semántico o léxico) y el peso de la variante de consulta. WRRF resulta robusto frente a las distintas escalas de puntuación de BM25 y FAISS y ante la heterogeneidad introducida por las variantes, ya que opera únicamente sobre posiciones relativas (*rankings*) en lugar de sobre puntuaciones absolutas.

D. Reranking *sensible a la consulta*

Sobre el conjunto fusionado se selecciona un número de candidatos máx(rerank_top_n, $4 \cdot$ final_k), ampliado opcionalmente con un rescate semántico y, en consultas de un único *token*, con un rescate de coincidencia exacta léxica. El *cross-encoder* puntúa cada candidato frente a varios *prompts* (original, variantes de capitalización, variantes derivadas) y se conserva la mejor puntuación.

A continuación se aplican ajustes de puntuación (*boosts* y penalizaciones) basados en heurísticas del dominio. Estos ajustes modifican la puntuación de cada candidato en función de señales específicas: se bonifica un documento cuando la consulta sugiere explícitamente un departamento y el documento pertenece a dicho departamento; se penalizan fuentes no procedentes de tickets en consultas cortas donde los tickets tienen mayor probabilidad de ser relevantes; se reduce la puntuación de fragmentos con baja densidad de información; se favorecen títulos que cubren todos los términos de la consulta; y se bonifica la coincidencia exacta en consultas de un único término. Estos ajustes permiten incorporar conocimiento del dominio del soporte técnico HPC que los modelos neuronales genéricos no capturan por sí solos.

E. Post-proceso a nivel de ticket

El motor opera internamente a nivel de *chunk* (fragmento), pero al técnico se le reporta la información a nivel de ticket mediante un enlace al servicio RT alojado en el CESGA. La capa de la API deduplica por `ticket_id`, aplica *overfetch* adaptativo cuando la deduplicación deja la respuesta corta, y produce fragmentos de texto limpios eliminando prefijos redundantes e incorporando el título de la página cuando aporta contexto. Esta separación entre *ranking* interno (fragmento) y presentación externa (ticket) permite optimizar la calidad de recuperación sin contaminar la información presentada al técnico.

V. Ingesta incremental y *hot-swap*

Un servicio de búsqueda en producción no puede permitirse paradas para reindexar. Fragata resuelve este requisito con una arquitectura de ingesta orquestada por trabajos Slurm, el planificador de trabajos del FinisTerrae III, y un mecanismo de recarga atómica del motor.

A. Modelo de trabajo

Toda ingesta, ya sea iniciada por API (`/ingest/web`, `/ingest/pdf`, `/ingest/repo-docs`, `/ingest/rt-weekly`) o por *scheduler*, se materializa como un trabajo con un manifiesto (`manifest.json`) y un registro de ejecución (`job.log`) persistidos atómicamente. El manifiesto mantiene estado (`queued`, `running`, `succeeded`, `failed`), etapa actual, progreso, petición validada y métricas por etapa.

B. Zona crítica serializada

Las etapas que mutan estado global, `merge` del *dataset*, actualización del catálogo de fuentes, adición al índice FAISS y recarga del motor, se ejecutan bajo un *lock* global de mutación. Sin esta serialización, dos trabajos concurrentes podrían pisar el *dataset* del otro, añadir datos sobre estados inconsistentes o recargar el motor con artefactos de generaciones distintas. El coste asumido es una menor concurrencia en la zona crítica, aceptable dado el patrón de uso del centro.

C. Adición incremental con promoción atómica

El módulo de adición incremental carga el índice FAISS activo, añade los documentos del delta, valida el crecimiento de índice y almacén de documentos, escribe en una ubicación de *staging*, vuelve a validar y solo entonces promociona atómicamente el nuevo índice y rota el anterior a una copia de seguridad con marca temporal. En el flujo *online* cualquier inconsistencia provoca un fallo explícito en lugar de disparar una reconstrucción completa silenciosa, lo que permite detectar y diagnosticar problemas de inmediato.

D. Hot-swap *del motor*

Tras una mutación exitosa del índice, el orquestador solicita una recarga del motor. El gestor utiliza dos *locks* separados: uno serializa las construcciones y el otro protege el puntero al motor activo. La construcción del nuevo motor ocurre fuera del intercambio final y solo si concluye correctamente se hace efectivo el cambio, incrementando un contador de generación. Si la construcción falla, el servicio sigue respondiendo con el motor anterior, garantizando degradación controlada sin interrupción del servicio.

E. *Ciclo de ingesta y* watermark *transaccional*

El *pipeline* por lotes semanal extrae una ventana incremental de RT determinada por una marca temporal de referencia (*watermark*) más un solapamiento de seguridad. Este solapamiento es necesario para garantizar que no se pierden tickets cuya última modificación se produjo en un instante muy próximo al límite de la ventana anterior: diferencias de reloj entre servidores, transacciones en curso en el momento del corte o retrasos en la propagación de datos podrían causar que un ticket modificado justo antes del *watermark* no quedase registrado. Al solapar ligeramente las ventanas, el sistema reprocesa un pequeño número de tickets ya conocidos, que se descartan por deduplicación, pero evita lagunas en la cobertura.

El *pipeline* prepara los datos por departamentos, consolida un delta global, actualiza el índice y recarga el motor. El *watermark* solo se confirma si todas las etapas anteriores han tenido éxito, lo que convierte el avance del puntero temporal en una operación transaccional con respecto al estado del motor.

VI. Despliegue y conexión con FinisTerrae III

Fragata se despliega en una topología máquina virtual + supercomputador, ilustrada en la Fig. 2. La máquina virtual aloja el *frontend* estático servido por Nginx[5] y el *backend* FastAPI gestionado por `systemd`. La comunicación entre el *frontend* y el *backend* se configura mediante CORS (*Cross-Origin Resource Sharing*), un mecanismo de seguridad de los navegadores web que controla qué orígenes (dominios) tienen permiso para realizar peticiones a un servidor distinto del que sirvió la página. Esta configuración permite separar el *frontend* y el *backend* en dominios independientes manteniendo la seguridad del servicio.

Las etapas costosas de ingesta y reindexado pueden descargarse al supercomputador FinisTerrae III (Fig. 2) mediante un mecanismo de *offload* controlado por variables de entorno. El ciclo de vida remoto se materializa como etapas observables (`resource_requested`, `waiting_resources`, `running_remote`, `sync_back`) cuyos metadatos se persisten para trazabilidad.

La política de liberación de recursos es configurable: en el modo automático, el éxito remoto implica liberación implícita; en modo de cancelación explícita se invoca la cancelación del trabajo también en éxito; ante fallo del *workload*, siempre se intenta cancelar la asignación. Esta integración permite reindexar volúmenes grandes del histórico RT sin saturar la máquina virtual y aprovechando los nodos del FinisTerrae III en huecos de baja ocupación.

VII. Resultados preliminares y discusión

El sistema está actualmente desplegado en producción interna del CESGA. Los resultados cualitativos sobre consultas reales del equipo de soporte muestran que Fragata recupera de forma sistemática tickets antiguos que resultaban invisibles para el buscador nativo de RT, especialmente en los siguientes escenarios:

- Consultas en lenguaje natural en español o gallego sobre incidencias originalmente reportadas en inglés, gracias a la traducción por diccionario y al modelo de *embeddings* multilingüe.
- Variantes morfológicas o errores tipográficos en nombres de aplicaciones científicas, que el buscador léxico estricto de RT no podía resolver.
- Consultas de intención (por ejemplo, "cómo instalar X" o "error al lanzar Y") donde el significado de la consulta importa más que las palabras exactas empleadas.
- Búsquedas combinadas por contenido y rango de fechas o departamento.

Las principales limitaciones detectadas son la dependencia de la calidad de la limpieza del correo en las fases previas al indexado, las heurísticas de intención basadas en diccionarios estáticos en lugar de modelos aprendidos, el vocabulario ES/EN curado en lugar de un sistema de traducción general, y un coste de *reranking* que crece con el producto del número de candidatos por el número de variantes de consulta.

VIII. Conclusiones y trabajo futuro

Este trabajo ha presentado Fragata, un sistema de recuperación semántica de tickets de soporte HPC basado en RAG híbrido y desplegado sobre la infraestructura del CESGA. La combinación de recuperación densa mediante *embeddings*, recuperación léxica BM25, fusión ponderada por WRRF y *reranking* con *cross-encoder*, junto con una arquitectura de ingesta incremental con *hot-swap* del motor y descarga selectiva al supercomputador FinisTerrae III, ha permitido convertir más de veinte años de histórico de Request Tracker en un recurso efectivamente consultable, superando las limitaciones del buscador nativo de RT 4.4.1.

Como trabajo futuro se contempla la incorporación de un clasificador de intenciones aprendido automáticamente en lugar de heurísticas estáticas, la evaluación cuantitativa con un conjunto curado de

[5]Nginx (`https://nginx.org`) es un servidor HTTP y proxy inverso de alto rendimiento, ampliamente utilizado para servir contenido estático y como punto de entrada que distribuye las peticiones entre los servicios internos de una aplicación.

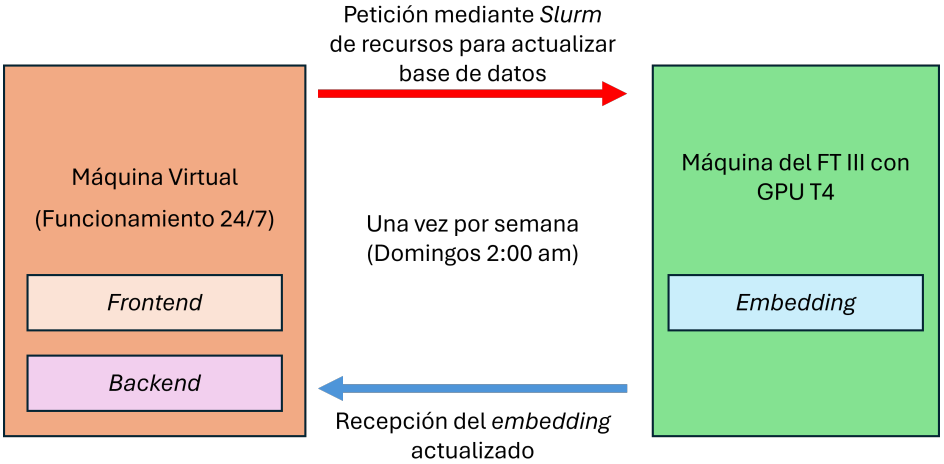


Fig. 2: Topología de despliegue: la máquina virtual aloja el servicio web y la API, mientras que las etapas costosas de indexación se descargan al supercomputador FinisTerrae III equipado con GPU NVIDIA T4.

consultas y juicios de relevancia generados por el propio equipo de soporte, la extensión del sistema a una interfaz conversacional con generación aumentada que sintetice respuestas accionables citando los tickets fuente, y la generalización del despliegue a otros centros HPC interesados en explotar sus propios históricos de soporte.

Agradecimientos

Este trabajo de investigación se desarrolló gracias al acceso concedido por el Centro de Supercomputación de Galicia (CESGA) a sus infraestructuras de supercomputación. El Superordenador FinisTerrae III y su sistema de almacenamiento permanente de datos han sido financiados por el NextGeneration EU 2021 Plan de Recuperación, Transformación y Resiliencia, ICT2021-006904, y también con el Programa Operativo Plurirregional de España 2014-2020 del FEDER (Fondo Europeo de Desenvolvemento Rexional), ICTS-2019-02-CESGA-3, y con el Programa Estatal de Fomento da Investigación Científica y Técnica de Excelencia del Plan Estatal de Investigación Científica y Técnica y de Innovación 2013-2016 Subprograma estatal de infraestructuras científicas y técnicas y equipamiento del FEDER, CESG15-DE-3114.

Adicionalmente, este trabajo ha sido realizado en el marco del Proyecto de actualización tecnológica del nodo de computación y datos del Centro de Supercomputación de Galicia (CESGA), financiado por el Plan de Recuperación, Transformación y Resiliencia a través del instrumento NextGenerationEU de la Unión Europea, en el ámbito del Proyecto Estratégico para la Recuperación y Transformación Económica de Microelectrónica y Semiconductores (PERTE Chip), de conformidad con el Real Decreto 714/2024.

Referencias

[1] Best Practical Solutions, "Request tracker (rt)," `https://bestpractical.com/request-tracker`.

[2] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[3] Stephen Robertson and Hugo Zaragoza, "The probabilistic relevance framework: BM25 and beyond," *Foundations and Trends in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.

[4] Rodrigo Nogueira and Kyunghyun Cho, "Passage re-ranking with BERT," in *arXiv preprint arXiv:1901.04085*, 2019.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, 2019.

[7] Nils Reimers and Iryna Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proceedings of EMNLP*, 2019.

[8] Vladimir Karpukhin, Barber Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih, "Dense passage retrieval for open-domain question answering," in *Proceedings of EMNLP*, 2020.

[9] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira, "Pyserini: A Python toolkit for reproducible information retrieval research with sparse and dense representations," in *Proceedings of SIGIR*, 2021.

[10] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2024.

[11] Rahul Potharaju, Navendu Jain, and Cristina Nita-Rotaru, "Juggling the jigsaw: Towards automated problem inference from network trouble tickets," in *Proceedings of NSDI*, 2013.

[12] Wenjun Zhou, Wei Xue, Ramesh Baral, Hongyuan Zha, and Robert Welch, "Smart ticket routing by multi-criteria learning," in *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, 2016.

[13] Jeff Johnson, Matthijs Douze, and Hervé Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2021.

[14] Luiz Henrique Bonifácio, Hugo Abonizio, Marzieh Fadaee, and Rodrigo Nogueira, "mMARCO: A multilingual version of the MS MARCO passage ranking dataset," in *arXiv preprint arXiv:2108.13897*, 2022.

[15] Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher, "Reciprocal rank fusion outperforms Condorcet and individual rank learning methods," in *Proceedings of SIGIR*, 2009.

Multimodal LiDAR–RGB Fusion for Object Classification on IoT Edge Devices

Blanca Poves-Arjona, Rogelio Hernandez, Gabriel Mujica[1]

Abstract— **Deploying multimodal perception at the edge requires balancing the benefits of sensor fusion with the limitations of embedded hardware. This work investigates object-level classification using Light Detection and Ranging (LiDAR) and RGB data under edge constraints, using a unified framework based on PointNet and MobileNetV2 to compare unimodal baselines with early, middle, and late fusion strategies. Experiments on simulated and real-world data show that fusion can improve performance, while real-world results are consistent with the trends observed in simulation. Moreover, late fusion provides the most robust behaviour under domain shift. Deployment on a Raspberry Pi 5 with a Hailo-8L accelerator demonstrates real-time performance with limited accuracy degradation, supporting the feasibility of multimodal perception on low-cost edge hardware.**

Keywords— **Multimodal perception, light detection and ranging (LiDAR), RGB camera, object classification, sensor fusion, edge computing.**

I. Introduction

PERCEPTION systems deployed in real environments must combine reliability with practical feasibility. This trade-off is particularly important in infrastructure-based settings such as railway level crossings, where sensors are fixed, continuously active, and expected to process data in real time on compact, low-power edge devices. In this context, the objective is not only to achieve high accuracy under controlled conditions but also to maintain consistent performance under limited computational resources and changing environmental conditions.

In practice, these systems are often structured into two stages: a perception stage, covering sensing and detection, and an interpretation stage, where higher-level understanding such as classification and tracking is performed [1]. Recent approaches often integrate these components into end-to-end frameworks. However, this coupling can increase computational cost on edge platforms and make it harder to analyse the contribution of each sensing modality and fusion strategy. This study focuses on the interpretation stage by treating classification as a separate component, while keeping the proposed module compatible with a broader perception system.

The choice of sensing modality plays a central role in this balance. RGB cameras and Light Detection and Ranging (LiDAR) sensors are among the most widely used modalities for scene understanding [2], [3]. Cameras provide rich visual information, such as colour and texture, while LiDAR captures precise geometric structure and depth. Each modality presents clear limitations when used in isolation: camera-based systems are sensitive to changes in illumination and appearance, while LiDAR lacks semantic detail. As a result, relying on a single sensor is often insufficient for robust performance in real-world scenarios. Combining both modalities helps mitigate their individual limitations. Multimodal fusion allows geometric and visual information to be used jointly, often improving downstream classification performance [4], [5]. However, these improvements increase model complexity and computational cost, which can be difficult to maintain on embedded platforms. Performance gains from multimodal fusion must, therefore, be considered alongside their computational feasibility on constrained hardware.

The evaluation setting reflects the constraints of the target deployment scenario. In particular, the data are designed to match the characteristics of edge-orientated sensing, where LiDAR observations exhibit a reduced point density, as in low-channel LiDAR sensors commonly used in edge deployments, and must be synchronised with RGB data at the object level. This contrasts with many publicly available datasets, which often rely on high-resolution sensors or lack precise multimodal alignment, and therefore do not fully represent the conditions considered in this work.

The approach uses a lightweight classification pipeline with dedicated neural backbones for each modality and three fusion strategies: early, middle, and late fusion. All configurations are evaluated within a unified framework, enabling a consistent comparison across sensing modalities and fusion strategies. Experiments are conducted on both a controlled simulated dataset and a real-world benchmark, enabling evaluation under both controlled and realistic conditions. The system is further deployed on a Raspberry Pi 5 equipped with a Hailo accelerator, so that latency, efficiency, and classification performance can be measured on the target edge platform. The main contributions of this work are summarised as follows:

- A comparative evaluation of LiDAR-only, RGB-only, and multimodal object classification in a unified experimental framework.
- An analysis of early, middle, and late fusion strategies from both performance and efficiency perspectives.
- A deployment-orientated evaluation on an edge computing setup based on Raspberry Pi 5 with Hailo acceleration, assessing the feasibility, latency, and efficiency of multimodal perception on resource-constrained edge hardware.

[1]Centro de Electrónica Industrial y Sistemas Multimodales, Universidad Politécnica de Madrid, e-mail: {b.poves, r.hlorite, gabriel.mujica}@upm.es.

The results show how accuracy and efficiency trade off in multimodal perception systems, identifying which configurations are most suitable for practical edge deployment.

II. Related Work

This section reviews previous work on object classification, LiDAR and RGB representations, multimodal fusion strategies, efficient perception on edge devices, and the use of simulated and real-world datasets for evaluation.

A. Object Classification in Image and 3D Data

Recent work in object perception has largely converged towards unified pipelines, where localisation and recognition are solved jointly within end-to-end detection models.

However, when object instances are already available, for example after image cropping or point cloud clustering, the task can be reformulated as object-level classification. In this setting, each instance is assigned a semantic label independently, supporting modular pipeline designs and enabling controlled comparisons between sensing modalities. Although classification has been widely studied in both image and 3D domains, recent work has increasingly focused on detection-orientated pipelines, often combining multiple sensing modalities [6], [5]. In such approaches, the contribution of each modality is tightly coupled within the detection framework, making it difficult to isolate and analyse their individual impact. Moreover, these methods are typically designed for high-performance computing environments and are not directly suited to resource-constrained edge platforms. As a result, object-level classification in multimodal settings under edge constraints remains comparatively underexplored.

B. LiDAR and RGB Representations for Object Classification

Building on this formulation, the design of object classification systems depends not only on the sensing modality but also on how the data are represented and processed. In this context, LiDAR and RGB inputs require different design choices, as each modality imposes its own structural constraints. LiDAR data are typically represented as 3D point clouds, which capture the geometric structure of the scene, whereas RGB images provide dense visual information in the image plane. These modalities are often considered complementary in perception systems, as they provide geometric and appearance information, respectively [3], [5]. These differences directly influence the type of models and feature extraction strategies that can be applied to each modality.

B.1 LiDAR-Based Classification and Point-Based Learning

Processing 3D point clouds for object classification is strongly influenced by the chosen data representation. Early approaches rely on structured formats such as voxel grids, allowing convolutional operations at the expense of increased memory usage and discretisation artefacts [7]. Alternative strategies project the point cloud onto 2D representations, such as bird's-eye-view maps, trading geometric fidelity for computational efficiency [3].

Point-based methods take a different approach, operating directly on raw point sets without requiring intermediate representations. PointNet [2] introduced a simple yet effective architecture capable of learning from unordered point clouds while preserving permutation invariance, making it particularly suitable for classification tasks. Its main limitation lies in the limited ability to capture local geometric structures, which motivated extensions such as PointNet++ [8], where hierarchical feature learning is used to model the local context more effectively.

B.2 RGB-Based Classification and Lightweight Models

Image-based classification relies on dense visual representations, where convolutional neural networks have become the standard approach. These models learn hierarchical features directly from pixel data and have demonstrated strong performance on large-scale benchmarks such as ImageNet [9].

To meet the constraints of embedded platforms, lightweight architectures have been developed to reduce computational and memory requirements while maintaining competitive performance. In particular, the MobileNet family [10], [11] uses separable depthwise convolutions and inverted residual connections to achieve efficient inference. Other designs, including ShuffleNet [12] and EfficientNet [13], further explore the trade-offs between accuracy, model size, and memory access patterns.

C. Multimodal Fusion for LiDAR-RGB Perception

LiDAR-RGB fusion has been widely explored as a way to combine geometric structure with visual appearance in perception systems. In most cases, these approaches are developed within end-to-end detection frameworks, where fusion is embedded in the joint localisation and recognition process. This makes it difficult to isolate the role of fusion in object-level classification when classification is treated as an independent task. Fusion strategies are commonly categorised according to the stage at which the information from each modality is integrated [14], [15].

Early fusion [3] operates at the input level, for example, by augmenting 3D points with colour information or projecting LiDAR data onto the image plane. A common approach is point-level fusion, where image features are associated with individual LiDAR points, as in PointPainting [16].

Middle fusion [5], [17] combines feature representations from each modality after separate processing branches. Each modality is first encoded in its own feature space, which makes the design suitable for heterogeneous data structures. It also allows the use

of pretrained backbones, which is useful in resource-constrained settings.

Late fusion [15] combines the output of independent models at the decision level, typically by averaging or voting. This keeps the system modular and easy to integrate, although interactions between modalities are limited.

Beyond these paradigms, other variants have been explored. Some approaches use one modality to guide the processing of the other, such as Frustum PointNets [6]. Others integrate fusion directly within the network, for example, through shared bird's-eye-view representations or attention mechanisms [18], [5]. These methods often improve performance, but increase model complexity.

From a deployment perspective, fusion is constrained by sensor synchronisation. Differences in acquisition rates, latency, or timestamp alignment can introduce inconsistencies in the combined representation, especially in real-world scenarios where perfect synchronisation is not guaranteed. For this reason, early, middle, and late fusion remain well-defined and practical design options, as they provide a controlled design space with predictable computational requirements [14].

D. Efficient Perception on Edge Devices

Running perception systems directly on edge devices enables local data processing and reduces reliance on external communication, which is particularly important in real-time applications [19], [20]. However, this setting introduces strict constraints on computational resources, memory footprint, and energy consumption, which directly impact both model design and inference pipelines.

These limitations become more pronounced at the extreme edge, where devices must meet tight latency requirements and often rely on reduced numerical precision. In this context, specialised hardware such as Neural Processing Units (NPUs) enables the deployment of deep learning models through quantised inference, although not all architectures are equally suited to these conditions [21], [22]. As a result, model design must balance accuracy with hardware efficiency and deployability.

Lightweight architectures have therefore become a common choice for edge-based perception. As discussed in Section B.2, lightweight architectures are essential to balance accuracy and computational cost in these environments.

For LiDAR data, point-based approaches operate directly on raw point sets, avoiding the need for intermediate representations such as voxelisation.

Several works have explored the deployment of perception pipelines on edge hardware, particularly for LiDAR-based detection and tracking tasks [20], [23], [1]. Although these approaches focus on end-to-end perception, they highlight that achieving real-time performance depends not only on lightweight models but also on data movement, memory access, and system integration.

When multiple sensing modalities are considered, these challenges become more pronounced. Multimodal fusion increases computational cost and introduces additional constraints related to synchronisation and data alignment, making tightly coupled architectures difficult to deploy on low-power devices.

E. Simulation and Real-World Datasets

Deep learning-based perception systems require large amounts of annotated data, which are difficult to obtain in real-world settings, particularly when multiple sensors must be synchronised and accurately calibrated.

Simulation offers an effective alternative, providing full control over scene configuration and sensor placement while generating precise ground-truth labels without manual effort [24], [25]. Platforms such as CARLA [26] are widely used in autonomous driving research, as they support multimodal sensing and allow consistent sensor configurations, including synchronised LiDAR and RGB data. This flexibility also makes it possible to simulate specific sensor setups, including low-cost or application-specific LiDAR configurations that are poorly represented or unavailable in real-world datasets.

Simulation alone cannot fully characterise system performance under real-world conditions. Synthetic environments struggle to replicate effects such as sensor noise, calibration errors, and timing inconsistencies.

Datasets that provide synchronised LiDAR and camera data from infrastructure-based viewpoints remain relatively limited. Recent efforts such as RCooper [27], DAIR-V2X [28], and TUMTraf [29] have begun to address this gap, although they are often restricted to specific scenarios or configurations. In these datasets, point clouds also exhibit sparsity and non-uniform density. Fewer points available for distant objects can affect the stability of lightweight models, especially under few-channel LiDAR configurations typical of edge deployments.

Combining simulated and real-world datasets provides a complementary evaluation setup. Simulation enables controlled experimentation and scalability, while real data capture the complexity of practical deployments. This combination is particularly relevant for multimodal perception systems operating under edge constraints.

III. System Architecture

A. Overview

The system performs object-level classification from multimodal sensor data, combining LiDAR point clouds and RGB imagery in a pipeline designed for edge deployment. The target platform is a Raspberry Pi 5 coupled with a Hailo-8L accelerator.

Objects are assumed to be available from an upstream perception stage and are represented using annotated 3D bounding boxes, which associate LiDAR points with the corresponding image crops. The result is an aligned object-level representation across

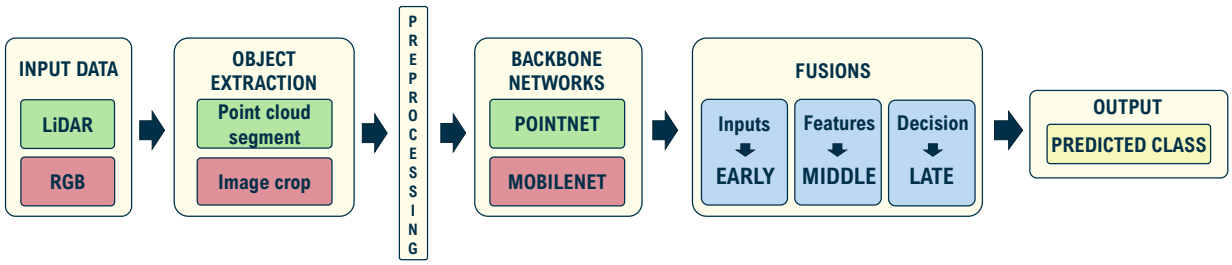


Fig. 1: Overview of the multimodal object classification pipeline.

both sensing modalities. LiDAR and image data are processed independently using lightweight backbone networks. A PointNet-based model handles the point cloud representation, while MobileNetV2 encodes visual appearance from the RGB crops. The resulting embeddings are then combined according to the selected strategy. The architecture supports three fusion schemes: early fusion at the input level, middle fusion after feature extraction, and late fusion at the decision level. This enables a direct comparison between unimodal baselines and multimodal configurations under identical conditions.

The pipeline runs on a heterogeneous edge platform. Data preparation and fusion are handled on the CPU, while inference is delegated to the NPU using quantised models compiled with the Hailo toolchain. This division supports a deployment-orientated evaluation in which latency, efficiency, and classification performance are assessed under realistic resource constraints.

Figure 1 shows the overall pipeline, including both the data flow and the deployment setup.

B. Input Data

The input to the system consists of synchronised LiDAR point clouds and RGB images, aligned at the object level through annotated 3D bounding boxes. Two complementary data sources are used for evaluation: a synthetic dataset generated in the CARLA simulator, providing controlled sensing configurations, and real-world recordings from the RCooper dataset. Both datasets provide synchronised LiDAR and RGB observations, where synchronisation refers to temporally aligned sensor data corresponding to the same acquisition frame. This allows objects to be consistently represented across modalities. Although the original annotations differ, all labels are mapped into a shared set of object categories: car, pedestrian, bicycle, truck, and van, to ensure consistency across datasets.

B.1 CARLA Dataset

A synthetic dataset is generated using the CARLA simulator, allowing a controlled configuration of both the scene and the sensing setup. This baseline configuration is known as the ideal scenario.

The LiDAR sensor is configured to resemble lower-cost devices such as the Velodyne Puck VLP-16, according to the sensing constraints of edge-orientated deployments. Each frame includes synchronised RGB images, LiDAR point clouds, annotated 3D bounding boxes, and metadata describing the actors in the scene. This setup supports the extraction of object-level samples with consistent alignment between both sensing sources.

Two additional variants are generated to analyse performance under degraded conditions. The night version modifies the scene lighting, mainly affecting the visual input. The second variant perturbs the LiDAR measurements through point jitter, small rotations, and local noise.

B.2 RCooper Dataset

The experiments also incorporate real-world infrastructure-based recordings from the RCooper dataset. The dataset provides synchronised LiDAR and camera data together with annotated bounding boxes, enabling object-level sample extraction consistent with the synthetic data. This allows the classification pipeline to be evaluated under real sensing conditions, including sensor noise and calibration imperfections.

Two types of scenarios are considered: corridor and intersection sequences. Corridor scenes involve traffic moving in a common direction, while intersection scenes present more complex dynamics, with objects approaching from multiple directions and more frequent occlusions. The intersection group (117–118–120–119) is excluded due to calibration inconsistencies reported in the official repository, which affect the alignment between sensor data and annotations. The remaining groups are used without modification.

C. Data Preprocessing

The data from both datasets are transformed into object-level representations before being used in the classification pipeline. Each object is defined by its 3D bounding box, which provides a common spatial reference for extracting aligned LiDAR segments and RGB image crops. This ensures consistency between both modalities before any model-specific processing is applied. Each modality is then adapted to match the input requirements of its corresponding network. Point clouds are converted into a fixed-size representation, while image crops are resized to a consistent format for visual encoding. In addition, a fused representation is constructed for the early fusion configuration by combining information from both sources at the point level.

Table I: Number of object instances per dataset and scenario.

Dataset	Scenario	Objects
CARLA	Ideal	4592
CARLA	Night	4592
CARLA	LiDAR corrupted	4592
RCooper	Corridor	56173
RCooper	Intersection	13793

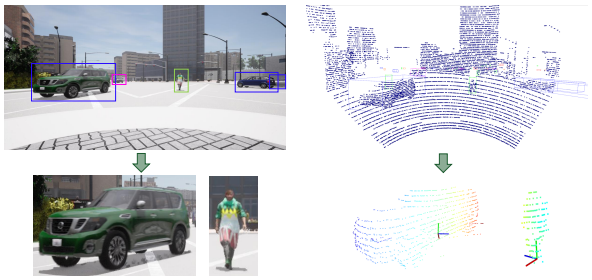


Fig. 2: CARLA: Object-level LiDAR-RGB extraction



Fig. 3: RCooper: Object-level LiDAR-RGB extraction

C.1 Object Extraction

Object instances are obtained from annotated 3D bounding boxes available in each scene. These annotations define the spatial extent of each object and provide a common reference across both sensing modalities. Since the objects are already defined, the task can be formulated as a classification problem, allowing a controlled evaluation of multimodal representations.

For LiDAR data, the point cloud of the scene is transformed into a local coordinate frame aligned with the object's orientation and centred on its geometry, while preserving its original scale. Points falling within the limits of the box are then selected, producing a segment associated with that instance.

For visual input, the same 3D geometry is projected onto the image plane to determine the corresponding region in the RGB image. The projected corners define the axis-aligned 2D bounding box from which the image crop is obtained.

This process ensures that both modalities provide geometrically consistent observations of the same object, as illustrated in Fig. 2 for CARLA and Fig. 3 for RCooper. Only instances with valid observations in both modalities are retained. Objects without LiDAR points or without a corresponding image region are discarded to ensure consistent multimodal inputs.

Table I summarises the number of object instances extracted from each dataset and scenario after the object extraction stage. While RCooper provides a significantly larger number of instances, reflecting its real-world variability, CARLA relies on a smaller but more controlled set of scenes with limited actor diversity. The three CARLA variants share the same set of objects, as they are generated from the same scenes under different sensing conditions.

C.2 Modality-specific preprocessing

Once the object instances have been extracted, each modality is adapted to match the input requirements of its corresponding model. Although both branches describe the same object, their representations are processed independently before being fed into the network.

C.2.a LiDAR Processing. The number of LiDAR points per object varies depending on the distance, the object size, and the partial occlusions. To ensure a consistent input, every object is represented by a fixed number of points. Each point cloud is adjusted to a fixed set of $N = 1024$ points. When more points are available, a random subset is selected. When fewer points are present, existing ones are oversampled until reaching the desired size. This preserves the geometric distribution while providing a uniform input.

The original geometric scale of each object is preserved, as absolute dimensions are highly discriminative for traffic-related categories.

C.2.b Image Processing. Each object is encoded by its corresponding RGB crop. As crop size depends on distance and perspective, images are resized to a fixed resolution of 224×224 pixels. After resizing, the images are preprocessed following the standard MobileNetV2 procedure, applying the same input normalisation used during ImageNet pretraining, with pixel values scaled to the range $[-1, 1]$.

C.3 Early Fusion Preparation

In the early fusion configuration, visual information is integrated at the point level by associating each LiDAR point with its corresponding RGB observation. This formulation preserves the spatial correspondence between geometry and appearance at the lowest level of representation.

LiDAR points are projected onto the image plane using the intrinsic and extrinsic calibration parameters, allowing each point to be associated with its corresponding RGB triplet. Points outside the image boundaries are discarded, ensuring that only valid multimodal observations are retained. The resulting point set is then adjusted to a fixed size of $N = 1024$ points following the sampling procedure described in Section C.2.a. The final representation defines a six-dimensional feature vector for each point of the form (x, y, z, r, g, b). The RGB values are scaled to the range $[0, 1]$ before being concatenated with the geometric coordinates, defining the fused input representation for the PointNet-based model.

IV. System Implementation

A. Backbone Architectures

The extracted object instances are processed through two parallel branches, corresponding to LiDAR and RGB inputs. Each modality is handled by a dedicated neural backbone adapted to its data representation. Lightweight architectures are selected to match the constraints of the target platform, consisting of a Raspberry Pi equipped with a Hailo accelerator. In this context, a PointNet-based model is used for LiDAR data, while MobileNetV2 is adopted for RGB images. Both networks produce class predictions for their corresponding modality. Intermediate feature embeddings are extracted and later used in the fusion strategies described in the following sections.

A.1 Geometric Backbone

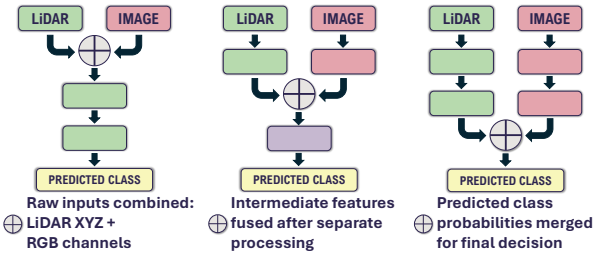
LiDAR point clouds associated with each object are processed using a network based on the PointNet architecture [2], which operates directly on unordered point sets. This avoids the need for structured representations, which can be inefficient in low-density LiDAR scenarios.

Each instance consists of a fixed set of $N = 1024$ points obtained during preprocessing. Depending on the configuration, the input consists of the spatial coordinates (x, y, z) for LiDAR-only experiments or the augmented representation (x, y, z, r, g, b) when early fusion is applied. The model extracts point-wise features and aggregates them through a symmetric max-pooling operation, producing a global descriptor of the object. This descriptor is then processed by fully connected layers to generate the final class probabilities.

The implementation follows the basic classification model provided in the official PointNet repository [30]. In this version, transformation networks (T-Net) are omitted to reduce computational cost and simplify the architecture, making it more suitable for deployment on resource-constrained hardware.

A.2 Visual Backbone

RGB image crops are processed using MobileNetV2, selected for its efficiency and suitability for resource-constrained environments. The network is initialised with ImageNet-pretrained weights and used without its original classification head. The convolutional backbone is followed by a global average pooling layer and a lightweight classifier adapted to the object categories considered in this work. During training, the model is fine-tuned using the cropped images obtained during preprocessing. Using pretrained weights provides a strong visual representation from the start, which helps when training on a limited number of object instances. In addition, a 128-dimensional feature vector is extracted from the penultimate layer and later used in the middle fusion strategy to combine information from both modalities.



(a) Early fusion (b) Middle fusion (c) Late fusion

Fig. 4: Fusion strategies.

B. Fusion Strategies

Beyond single-modality baselines, three strategies are used to combine LiDAR and RGB information. These approaches integrate both modalities at different stages of the pipeline: at the input level (Fig. 4a), after feature extraction (Fig. 4b), or at the final decision stage (Fig. 4c).

All configurations rely on the same backbone networks described previously. The models trained for the LiDAR-only and RGB-only settings are reused for middle and late fusion, ensuring that differences between approaches are driven by the fusion mechanism itself.

B.1 Early Fusion

In the early fusion configuration (Fig. 4a), the augmented point representation obtained during preprocessing (see Section C.3) is used as input to the classification network. Each point is described by the six-dimensional vector (x, y, z, r, g, b), combining spatial coordinates with the corresponding colour information.

The network structure follows the same design as the geometric backbone (Section A.1), with the input layer adjusted to accommodate the six-dimensional feature vector. Apart from the extended input representation, the network structure and the training procedure remain unchanged. This isolates the effect of the additional colour information without modifying the underlying model.

B.2 Middle Fusion

In the middle fusion structure (Fig. 4b), both inputs are first processed independently by their respective backbones. PointNet produces a 256-dimensional embedding from the point cloud, while MobileNetV2 generates a 128-dimensional representation from the image crop.

These representations are concatenated to form a joint feature vector that captures information from both sources. The combined vector is then passed to a lightweight, fully connected classification head with a 128-unit dense layer, batch normalisation, ReLU activation, dropout, and a final softmax layer.

B.3 Late Fusion

In the late fusion configuration (Fig. 4c), the LiDAR and image branches operate independently and

produce their respective class probability distributions. The final prediction is obtained by aggregating these outputs at the decision level.

In this work, the class probabilities of PointNet and MobileNetV2 are averaged. This aggregation avoids the introduction of an additional fusion module while still allowing both branches to contribute to the final decision.

C. Edge Deployment

The system is deployed on the Raspberry Pi platform equipped with a Hailo accelerator. The execution pipeline is divided between the CPU and the NPU, assigning each stage according to its computational requirements. Neural network inference is offloaded to the accelerator, where both the PointNet and MobileNetV2 models are executed using quantised versions compiled with the Hailo toolchain. The multimodal pipeline is executed sequentially, with the LiDAR and RGB branches independently evaluated on the accelerator prior to fusion. Each branch produces predictions and feature embeddings used by the fusion stage.

In the early fusion configuration, the projection and colour association are performed on the CPU before inference, producing the augmented point representation used as input to the accelerator.

In the middle fusion approach, feature embeddings from both branches are extracted on the accelerator and transferred to the CPU, where they are concatenated and passed to a lightweight classifier for final prediction.

In the late fusion configuration, the final decision is computed on the CPU by combining the class probabilities produced by both models. This separation keeps neural inference on the accelerator, while the fusion logic runs on the host processor and remains lightweight.

V. Experimental Setup

A. Training Protocol

All configurations are trained and evaluated on a shared set of object instances across modalities, ensuring a consistent comparison between LiDAR-only, RGB-only, and multimodal approaches.

A.1 Data split

For the CARLA dataset, samples are divided into training, validation, and test sets using a temporal split, preventing observations of the same object from appearing in both training and evaluation subsets. The same indices are shared across all sensing configurations and fusion strategies, ensuring that each object instance is assigned consistently across modalities and avoiding cross-modal data leakage.

For the RCooper dataset, the sequence-based partition defined in the original dataset is used.

A.2 Training strategy

The two modality-specific backbones are trained independently. The PointNet model is trained from scratch using LiDAR data, while MobileNetV2 is initialised with ImageNet-pretrained weights and fine-tuned on the extracted image crops.

For multimodal configurations, the unimodal backbones are frozen after training, so that all approaches rely on the same underlying models. Under this setting, middle fusion is implemented by training a lightweight classifier on concatenated feature embeddings, while late fusion combines the predictions from each branch without additional training. This ensures that performance differences are driven by the fusion strategy, rather than changes in the backbones.

In contrast, the early fusion model is trained independently, as it requires an adapted input representation combining geometric and colour information.

Table II: Main training hyperparameters.

Model	Learning Rate	Batch Size	Max. Epochs
PointNet	1×10^{-3}	32	150
PointNet (EF)	5×10^{-4}	16	150
MobileNetV2	5×10^{-3}	16	200
Middle Fusion	5×10^{-3}	16	30

A.3 Optimisation

All trainable models are optimised using categorical cross-entropy loss. Early stopping and learning-rate reduction on plateau are applied during training. Early stopping monitors validation loss with patience of 10 epochs, while the learning rate is reduced by a factor of 0.1 after 5 epochs without improvement.

The main training hyperparameters are summarised in Table II.

B. Evaluation Protocol

Model performance is evaluated using standard classification metrics including precision, recall, and F1-score, reported using weighted averaging, and accuracy. Confusion matrices are also included to provide a detailed view of the classification behaviour.

In addition to these metrics, prediction confidence and entropy are analysed to assess the reliability of the different configurations. Confidence is defined as the maximum predicted class probability, while entropy is computed over the softmax output distribution. These measures provide additional insight into how each approach handles uncertainty, particularly under challenging sensing conditions.

Beyond classification performance, practical metrics are considered to evaluate the suitability of the system for embedded deployment. These include inference time per sample, model size, and number of parameters, which help characterise the computational cost.

To evaluate real execution conditions, inference is performed on the target edge platform using the same test sets employed in the CPU-based experiments. This allows both classification performance and runtime behaviour to be assessed under realistic conditions. The predictions obtained on the embedded device are compared with those produced on

Table III: Weighted F1-score for the CARLA dataset. I, N and C denote Ideal, Night and LiDAR corrupted scenarios, respectively. The arrow indicates train→test configuration.

	Homogeneous			Cross-domain	
Model	I→I	N→N	C→C	I→N	I→C
LiDAR	0.926	0.956	0.847	0.943	0.830
RGB	0.991	0.974	0.991	0.133	0.991
Early	0.869	0.868	0.855	0.880	0.835
Middle	0.991	0.999	0.991	0.586	0.991
Late	0.991	0.993	0.991	0.929	0.991

the CPU to verify that quantisation and hardware execution do not introduce significant performance degradation.

VI. Results

This section presents the results obtained for the different model configurations, including unimodal approaches (LiDAR and RGB) and multimodal fusion strategies (early, middle, and late fusion). The analysis focuses on performance under controlled conditions, robustness to domain shift, and suitability for edge deployment. This section compares the unimodal baselines and fusion strategies across four settings: homogeneous CARLA experiments, CARLA domain shift, real-world RCooper evaluation, and edge deployment on the Hailo-accelerated platform.

The analysis first focuses on the CARLA dataset, which allows controlled evaluation under different environmental conditions. Three scenarios are considered (ideal, night, and LiDAR corrupted), enabling the analysis of both performance under matched conditions and robustness under domain shift, where models trained on ideal data are evaluated on degraded environments. Table III reports the Weighted F1-scores for all configurations.

A. Homogeneous Conditions (CARLA)

When training and evaluation domains match, all models achieve high performance. RGB obtains the best results in ideal and corrupted conditions, while LiDAR remains stable across all scenarios, including the corrupted one. Among fusion strategies, middle and late fusion reach the highest scores, whereas early fusion consistently performs below them.

B. Domain Shift Robustness

A different behaviour is observed under domain shift. Figure 5 provides a visual comparison of model performance across domains. The impact of domain shift is particularly evident for the RGB model, which shows a sharp performance drop when evaluated under night conditions.

This degradation also affects middle fusion, whose performance drops to 0.586 despite its strong results under homogeneous conditions. Since this approach relies on joint feature representations, errors in the visual branch directly distort the fused embedding.

In contrast, LiDAR remains relatively stable across domains. Late fusion achieves the most robust

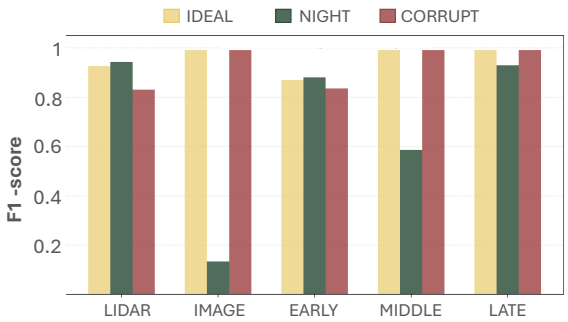


Fig. 5: CARLA domain-shift performance for models trained on the ideal scenario

Table IV: Confidence-based analysis under domain shift (CARLA ideal→night).

Model	**F1**	**Conf.**	**Margin**	**Entropy**
Middle fusion	0.59	0.82	0.70	0.51
Late fusion	0.93	0.65	0.38	0.74

behaviour (F1 = 0.929), as decision-level combination allows the model to rely on the modality that remains informative. Early fusion (F1 = 0.880) also shows reasonable robustness, outperforming middle fusion in this setting.

To further analyse these differences, we examine confidence-related metrics in the most challenging scenario (ideal→night). Confidence is defined as the maximum predicted probability, margin as the difference between the two highest class probabilities, and entropy as a measure of uncertainty.

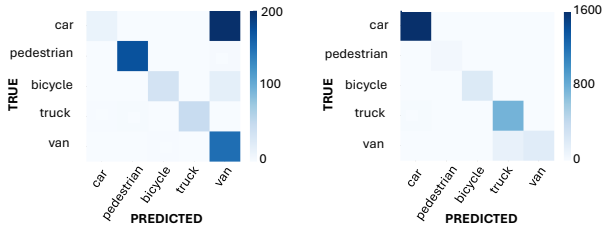
We focus on middle and late fusion, as they represent the most competitive multimodal strategies observed in the previous analysis.

Table IV compares both approaches in terms of performance and confidence-related measures. Despite its lower F1 score, middle fusion produces highly confident predictions (high confidence and margin, low entropy), indicating that the model remains certain even when predictions are incorrect. This suggests that errors introduced by degraded inputs are embedded in the learned representation, resulting in misleadingly confident predictions.

In contrast, late fusion maintains high performance while exhibiting lower confidence and higher entropy. This reflects a more cautious behaviour: when one modality becomes unreliable, its predictions tend to be less confident, reducing its influence in the final decision.

These observations help explain the behaviour reported in Table III. Middle fusion is strongly affected by degraded inputs, as errors propagate through the shared feature representation, whereas late fusion better handles disagreement between modalities.

In this scenario, most errors occur between visually similar classes such as cars and vans, particularly under degraded illumination conditions. This behaviour is illustrated in Fig. 6a, where confusions between both classes become more frequent.



(a) CARLA (I → N) (b) RCooper (Intersection)

Fig. 6: Confusion matrices highlighting dominant error modes across domains in middle fusion models.

C. Real-World Evaluation (RCooper)

Table V reports the results on two real-world scenarios with different levels of complexity: *corridor* and *intersection*. In the *corridor* scenario, all models achieve similarly high performance ($F1 > 0.980$), with only minor differences between unimodal and multimodal approaches. In contrast, the *intersection* scenario introduces greater variability, more complex spatial layouts, and frequent occlusions. Under these conditions, clearer differences between models emerge. Early fusion (0.915) does not improve over the unimodal baselines and slightly underperforms compared to LiDAR-only (0.929). This suggests that point-level colour association is sensitive to projection errors and does not always provide a more discriminative representation than the separate modality branches.

Middle and late fusion achieve higher scores, confirming the benefit of combining modalities in more challenging environments. Late fusion provides the best performance ($F1 = 0.962$), consistent with the trends observed in CARLA.

Most classification errors in this scenario occur between vans and trucks, which present similar geometric and visual characteristics, as illustrated in Fig. 6b.

Table V: Results on the RCooper dataset for the *corridor* and *intersection* scenarios. Weighted F1-score is reported.

Model	Corridor	Intersection
LiDAR only	0.983	0.929
RGB only	0.980	0.919
Early fusion	0.985	0.915
Middle fusion	**0.995**	0.946
Late fusion	0.994	**0.962**

D. Edge Deployment and Efficiency Evaluation

Table VI summarises the performance and efficiency metrics for all models deployed on a Raspberry Pi 5 equipped with the Hailo-8L accelerator. The results show that the impact of INT8 quantisation on predictive performance is limited. All models were quantised using the Hailo Dataflow Compiler via Post-Training Quantisation (PTQ). For instance, the Late Fusion model achieves an $F1$-score of 0.949 on the NPU, remaining close to its CPU baseline (0.962), while Middle Fusion even shows a slight improvement after deployment, likely due to quantisation-induced changes in the decision boundaries for this test subset rather than a systematic improvement.

Agreement between CPU and NPU predictions is high across all configurations, particularly for Early Fusion (0.996). In contrast, the RGB branch exhibits the lowest agreement (0.881), suggesting that the convolutional visual backbone is more sensitive to quantisation than the PointNet-based model.

In terms of performance, inference latency varies across modalities. The RGB model achieves the lowest latency (3.0 ms), while LiDAR-based models require approximately 11.8 ms per object. Multimodal configurations (Middle and Late fusion) increase this cost to around 14.9 ms, as both backbones are executed sequentially on the NPU and combined through lightweight fusion logic on the host CPU.

Despite this additional cost, the multimodal configurations achieve approximately 67 object-level classifications per second. This throughput remains compatible with real-time processing requirements in infrastructure-based perception systems, in line with previous work demonstrating real-time LiDAR-based perception on edge platforms [31].

Model sizes are significantly reduced after deployment, particularly for the RGB branch. Despite the constrained memory of the embedded platform, all models can be executed efficiently, as inference is performed in a layer-wise streaming manner with buffer reuse, rather than loading the entire network into memory at once.

VII. Conclusions and Future Work

Multimodal object classification with LiDAR and RGB can improve performance under edge constraints, but its effectiveness depends strongly on how the information from each modality is combined. Late fusion provides the most robust behaviour under degraded conditions, maintaining stable performance when individual modalities become unreliable. This suggests that combining class predictions is a reliable strategy when the two modalities are affected differently by the environment.

These observations remain consistent across both simulated and real-world data. While unimodal approaches perform well in simple scenarios, multimodal fusion becomes increasingly important as environmental complexity grows. The successful deployment on a Raspberry Pi 5 with Hailo-8L acceleration, achieving real-time throughput after INT8 quantisation, further demonstrates that robust multimodal perception is feasible on low-cost and resource-constrained platforms.

Future work will focus on integrating the proposed classification module into a complete perception pipeline to evaluate its impact at the system level. In addition, the role of simulation as a digital twin for edge perception will be further explored, particularly in terms of transferability to real-world conditions. Finally, hardware-aware optimisations, including parallelisation and efficient data movement, remain a key direction to improve scalability and efficiency in

Table VI: Performance and efficiency on the RCooper intersection subset. Latency per object includes on-device inference; fusion results account for both backbones and the fusion stage.

Model	F1 (CPU)	F1 (Hailo)	Latency (ms)	Throughput (obj/s)	Model size CPU (MB)	Deployed size (MB)	Agreement
LiDAR	0.929	0.927	11.8	84.6	9.43	5.41	0.963
RGB	0.919	0.908	3.0	328.9	19.16	3.60	0.881
Early fusion	0.915	0.913	11.8	84.6	9.44	5.41	0.996
Middle fusion	0.946	0.966	14.9	67.0	25.83	9.02	0.977
Late fusion	0.962	0.949	14.9	67.3	28.60	9.01	0.951

edge deployment.

Acknowledgment

The publication is part of the INARTRANS 4.0 project PLEC2023-010343, funded by MICIU/AEI/10.13039/501100011033.

References

[1] Rogelio Hernandez, Gabriel Mujica, and Jorge Portilla, "Situational awareness i2x pipeline methodology for lidar-based sensor networks at the edge of internet of things," in *2024 IEEE International Conference on Evolving and Adaptive Intelligent Systems (EAIS)*, 2024.

[2] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[3] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[4] Ming Liang, Bin Yang, Yun Chen, Rui Hu, and Raquel Urtasun, "Multi-task multi-sensor fusion for 3d object detection," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[5] Zhijian Liu, Haotian Tang, Alexander Amini, Xinyu Yang, Huizi Mao, Daniela Rus, and Song Han, "Bevfusion: Multi-task multi-sensor fusion with unified bird's-eye view representation," 2022.

[6] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data," 2018.

[7] Daniel Maturana and Sebastian Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[8] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 01 2012.

[10] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[11] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[12] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," 2017.

[13] Mingxing Tan and Quoc Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[14] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency, "Multimodal machine learning: A survey and taxonomy," 2017.

[15] Cees GM Snoek, Marcel Worring, and Arnold WM Smeulders, "Early versus late fusion in semantic video analysis," in *Proceedings of the 13th annual ACM international conference on Multimedia*, 2005, pp. 399–402.

[16] Sourabh Vora, Alex H. Lang, Bassam Helou, and Oscar Beijbom, "Pointpainting: Sequential fusion for 3d object detection," 2020.

[17] Natalia Neverova, Christian Wolf, Graham W. Taylor, and Florian Nebout, "Moddrop: adaptive multi-modal gesture recognition," 2015.

[18] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven Waslander, "Joint 3d proposal generation and object detection from view aggregation," 2018.

[19] Taiyuan Gong, Li Zhu, F. Yu, and Tao Tang, "Edge intelligence in intelligent transportation systems: A survey," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[20] Hong Zhang et al., "Object tracking for a smart city using iot and edge computing," *Sensors*, 2019.

[21] Jorge Portilla, Gabriel Mujica, Jin-Shyan Lee, and Teresa Riesgo, "The extreme edge at the bottom of the internet of things: A review," *IEEE Sensors Journal*, 2019.

[22] Itay Krispin-Avraham, Roy Orfaig, and Ben-Zion Bobrovsky, "Real-time 3d object detection using innovizone lidar and low-power hailo-8 ai accelerator," 2024.

[23] Cristian Wisultschew, Gabriel Mujica, Jose Manuel Lanza-Gutierrez, and Jorge Portilla, "3d-lidar based object detection and tracking on the edge of iot for railway level crossing," *IEEE Access*, 2021.

[24] Jin Fang, Dingfu Zhou, Feilong Yan, Tongtong Zhao, Feihu Zhang, et al., "Augmented lidar simulator for autonomous driving," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1931–1938, Apr. 2020.

[25] Cristian Wisultschew, Rogelio Hernandez, Carlos Pastor, and Jorge Portilla, "Synthetic lidar-labeled data set generation to train deep neural networks for object classification in iot at the edge," *IEEE Internet of Things Journal*, vol. PP, pp. 1–1, 12 2022.

[26] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun, "Carla: An open urban driving simulator," in *Proceedings of the Conference on Robot Learning (CoRL)*, 2017.

[27] Ruiyang Hao, Siqi Fan, Yingru Dai, Zhenlin Zhang, Chenxi Li, et al., "Rcooper: A real-world large-scale dataset for roadside cooperative perception," 2024.

[28] Haibao Yu, Yizhen Luo, Mao Shu, Yiyi Huo, Zebang Yang, Yifeng Shi, et al., "Dair-v2x: A large-scale dataset for vehicle-infrastructure cooperative 3d object detection," 2022.

[29] Walter Zimmer, Gerhard Arya Wardana, Suren Sritharan, Xingcheng Zhou, Rui Song, and Alois C. Knoll, "Tumtraf v2x cooperative perception dataset," 2024.

[30] Charles R. Qi et al., "Pointnet implementation (pointnet_cls_basic)," `https://github.com/charlesq34/pointnet`, 2017, Accessed: 2026-04.

[31] Rogelio Hernández, Gabriel Mujica, and Jorge Portilla, "Multi-object tracking at the edge of iot using lidar data for railway applications," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–15, 2026.

Sistemas ciberfísicos e IoT

Triple Redundancia Modular en Rust para Sistemas Multicore bajo Radiación

Darío González-Montesoro [1], A. Serrano-Cases[1], S. Cuenca-Asensi[1], R. Possamai-Bastos, [2] E. Atukpor [3] y A. Martínez-Álvarez[1],

Resumen— **Este trabajo propone un votador distribuido (DV-TMR) basado en Rust para mejorar la técnica clásica de triple redundancia modular. La propuesta está diseñada para lograr una mayor tolerancia a fallos en sistemas críticos para la seguridad que operan en entornos propensos a la radiación, eliminando el único punto de fallo del votador tradicional. Aprovecha las capacidades multihilo nativas de Rust para implementar un votador distribuido utilizando las primitivas de comunicación de `mpsc`. Esta propuesta hereda las características intrínsecas de seguridad de memoria de Rust y reduce la probabilidad de fallos inducidos dentro de la función de votador. Con el fin de evaluar la eficacia del DV-TMR, la propuesta se implementó en Rust y se comparó con una versión en C, ya que este lenguaje sigue siendo un estándar establecido en estos entornos. Además, esta propuesta se probó en una campaña de irradiación de neutrones en el ILL. Los datos experimentales obtenidos muestran que la implementación protegida presenta mejor fiabilidad. Además, las métricas de cross-section y Mean Work To Failure (MWTF) sugieren que Rust puede ser una opción viable para el desarrollo de software en aplicaciones críticas.**

Palabras clave— **Fiabilidad, Tolerancia a fallos, neutrones, ARM, Rust, TMR.**

I. Introducción

LAS plataformas de computación modernas dependen cada vez más de dispositivos Commercial Off-The-Shelf (COTS) debido a su bajo coste y su disponibilidad. Sin embargo, estos componentes no están diseñados para entornos adversos, donde la radiación o las perturbaciones eléctricas pueden provocar fallos transitorios o permanentes. Los fallos transitorios, también conocidos como *soft-errors*, son los efectos de un Single Event Error (SEE) y pueden modificar un bit (Single Event Upset, SEU) en los componentes de memoria o alterar el estado de un transistor. Garantizar la fiabilidad y la tolerancia a fallos en este tipo de sistemas es, por tanto, esencial [1, 2]. Los sistemas basados en COTS utilizados en aplicaciones aeroespaciales, industriales o críticas para la seguridad, requieren mecanismos de protección adicionales para mantener un funcionamiento fiable bajo condiciones adversas. Existen dos métodos para cumplimentar el requisito de fiabilidad. La primera opción implica modificar el hardware, mientras que la segunda implica modificar el software. Las técnicas de software introducen redundancia espacial o temporal para aportar fiabilidad. Dos de las técnicas de software más consolidadas son N-version y N-modular. La técnica N-version utiliza diferentes copias del mismo programa para aprovechar distintos recursos de hardware y mitigar fallos [3]. La técnica N-modular replica el programa usando diferentes recursos de hardware (redundancia espacial), o lo replica temporalmente si no hay suficientes recursos. En este contexto, dos de las propuestas más conocidas de N-modular son la Doble y Triple Redundancia Modular [4]. La primera utiliza dos copias y comprueba si existen errores, activando un mecanismo de recuperación si se detecta alguno. La segunda utiliza tres copias y calcula el resultado mediante un votador mayoritario. Debido a la baja probabilidad de que se produzca más de un error en cada ejecución, la técnica puede aislar la versión errónea y proporcionar el resultado correcto [5–7].

Históricamente, los lenguajes preferidos para implementar estas técnicas de fiabilidad de software, en dispositivos COTS y otros sistemas embebidos, han sido ensamblador y C. Esta decisión está justificada por el gran control a bajo nivel que proporcionan sobre los recursos del dispositivo, así como su capacidad para generar binarios compactos. Sin embargo, la gestión manual de memoria (*malloc/free*, punteros) se ha identificado como una de las principales causas de errores en memoria, como *buffer overflow*, *use-after-free* o fugas de memoria [8, 9]. Esta situación ha motivado a algunos autores a buscar alternativas que garanticen la seguridad de la memoria, sin sacrificar el alto rendimiento. En este sentido, Rust surge como un candidato prometedor para sistemas críticos, ya que su diseño garantiza la seguridad de la memoria. Rust combina rendimiento a nivel de sistema con garantías frente a la corrupción de la memoria y *data races* [10, 11]. Algunos estudios ya han comparado Rust con otros lenguajes en contextos multihilo y críticos para la seguridad, destacando sus ventajas en el control de la concurrencia y la seguridad del código [12, 13]. Estas propiedades convierten a Rust en una alternativa interesante a C para implementar técnicas de tolerancia a fallos. La combinación de la redundancia modular y las garantías de seguridad de Rust abre una nueva vía para el desarrollo de software fiable sobre hardware COTS. Este trabajo explora dicha intersección, aprovechando el modelo de concurrencia de Rust para diseñar una técnica

[1]Dept. de Tecnología Informática y Computación, Universidad de Alicante, Alicante, España, e-mail: {aserrano,sergio,amartinez}@dtic.ua.es
[2]CNRS, Grenoble INP, TIMA, Universidad Grenoble Alpes, Grenoble, Francia, e-mail: rodrigo.bastos@univ-grenoble-alpes.fr
[3]Institut Laue-Langevin, Responsable TENIS, Grenoble, Francia, e-mail: atukpor@ill.fr

TMR con votador distribuido capaz de mejorar la tolerancia a fallos y la fiabilidad del votador. Hasta donde alcanza nuestro conocimiento, este trabajo presenta la primera implementación de un votador completamente distribuido que aprovecha el multihilo nativo de Rust y sus garantías de seguridad. Al integrar el proceso de votación dentro de cada hilo redundante, nuestra propuesta elimina el votador tradicional como punto único de fallo y mejora la tolerancia frente a *soft errors*.

II. Estado del Arte

Las técnicas de software para mejorar la tolerancia a fallos proporcionan una alternativa más rentable a la redundancia de hardware, permitiendo la detección y la recuperación, tanto de fallos transitorios como permanentes, mediante mecanismos a nivel de software. Solouki et al. [14] ofrecen una visión general de estas técnicas de software, destacando enfoques algorítmicos, estructurales a nivel de hilo para mejorar la fiabilidad en sistemas embebidos. Sin embargo, estas presentan sobrecostes en el rendimiento y el uso de los recursos. En este sentido, Arslan y Unsal [15] introdujeron la replicación selectiva de regiones críticas de código para mitigar de forma eficiente corrupciones silenciosas de datos, mientras que So et al. [16] muestran un modelo de multihilo redundante a nivel de software (RMT), capaz tanto de detectar como de recuperar los errores. Dichev y Nikolopoulos [17] propusieron TwinPCG, un esquema de redundancia de doble hilo que combina recuperación hacia adelante y conciencia del rendimiento en cálculos científicos. De manera similar, Serrano-Cases et al. [18] abordaron los *soft errors* inducidos por radiación en sistemas *bare-metal* mediante mitigación multihilo, reforzando la aplicabilidad de la redundancia software en entornos embebidos. Siguiendo este desarrollo, el marco proporcionado por OpenMP ha desempeñado un papel fundamental para mejorar la fiabilidad en tareas con grandes especificaciones. En este contexto, Tahan y Shawky [19] introdujeron redundancia dinámica a nivel de tarea en OpenMP para mejorar la adaptabilidad de las tareas, mientras que Munera et al. [20] ampliaron este concepto en GuardianOMP, un marco que mejora la productividad y la resiliencia mediante replicación de tareas en OpenMP. Estos ejemplos previos de TMR, aunque eficaces, se basan en mecanismos de votación centralizados que pueden convertirse en puntos sensibles en los que pueda introducirse un fallo. Investigaciones recientes han explorado mecanismos de votación distribuida como medio para eliminar estos puntos sensibles en arquitecturas redundantes. Fara et al. [21] investigaron la planificación de la votación de réplicas en sistemas en tiempo real, introduciendo métodos para coordinar eficientemente tareas redundantes bajo restricciones temporales. Duarte y Hametner [22] propusieron una programación asíncrona con *futures* en contextos críticos, sugiriendo una distribución parcial de la toma de decisiones entre tareas concurrentes. Sin embargo, los enfoques existentes todavía dependen de la suma de votos centralizada o de supuestos específicos de planificación.

Todas las técnicas de software anteriores fueron desarrolladas en lenguaje C. Este lenguaje ha sido tradicionalmente la opción más popular para el desarrollo de código fiable en entornos críticos. La principal ventaja de utilizar C es su control de bajo nivel, que permite una gestión muy específica del hardware a bajo nivel. Sin embargo, esta cualidad presenta ciertas desventajas, ya que dificulta la generación de código seguro y la garantía de que la memoria se gestione correctamente. En [8] van Oorschot analiza los problemas de seguridad de memoria en C, destacando que entre el 65–70 % o más de las vulnerabilidades de software reportadas en los últimos años han involucrado errores de memoria. La gestión manual de memoria y las operaciones con punteros sin comprobación conducen a vulnerabilidades comunes como *buffer overflows* y errores de *use-after-free*. Los autores en [9] profundizan en esta última problemática, examinando vulnerabilidades de *use-after-free* en programas C/C++ y proponiendo un mecanismo de *heap* protegido para evitar la explotación de punteros colgantes. Pereira et al. [23] presentan técnicas que combinan análisis estático y comprobaciones en tiempo de ejecución para detectar *buffer overflows* y desreferenciaciones inválidas de punteros. En [24] los autores emplean un análisis dinámico para detectar *integer overflow to buffer overflow* mediante el seguimiento de operaciones enteras y su propagación en los accesos a memoria. El uso de valores no inicializados es otro error común en C, en [25] se presenta un sistema de protección que refuerza las aplicaciones frente a este comportamiento indefinido, garantizando la inicialización de todos los valores en el *heap* y la pila.

Esta situación ha motivado a algunos autores a buscar alternativas que garanticen la seguridad de la memoria y un alto rendimiento. La aparición de Rust, como una opción alternativa para el desarrollo de software crítico, representa un cambio de paradigma en la fiabilidad del software [26], ofreciendo seguridad de memoria y de hilos en tiempo de compilación mediante semánticas de propiedad y un tipado fuerte. En comparación con C y C++, donde la seguridad de la memoria debe garantizarse mediante la disciplina del programador, Rust impone esta seguridad en tiempo de compilación, reduciendo significativamente las vulnerabilidades a errores comunes de concurrencia y de memoria. Jung et al. [27] formalizan el marco RustBelt, proporcionando una base semántica para verificar la solidez de las garantías de seguridad de Rust. Los estudios de Qin et al. [28] analizaron aplicaciones reales en Rust, demostrando cómo su modelo de concurrencia previene eficazmente las *data races* al tiempo que destacan los desafíos de seguridad aún

existentes. Otros trabajos han mejorado además el aislamiento en tiempo de ejecución de Rust [29] y sus capacidades de comunicación *zero-copy* [30], reforzando su idoneidad para sistemas de alta integridad. Sini et al. [31] mostraron recientemente el uso de Rust para implementar una comprobación del flujo de control en aplicaciones tolerantes a fallos, validando el potencial de Rust para la fiabilidad en entornos críticos. Además, el modelo de hilos de Rust, construido en torno a std::thread y la sincronización mediante paso de mensajes, difiere fundamentalmente del modelo de memoria compartida de OpenMP: mientras que OpenMP permite paralelismo implícito mediante directivas del compilador, Rust impone propiedad y sincronización explícitas, garantizando una mayor seguridad a costa de una gestión manual de la concurrencia.

Hasta donde alcanza nuestro conocimiento, este estudio presenta el primer diseño de un mecanismo de votación completamente distribuido construido utilizando las características de concurrencia integradas de Rust y sus garantías de seguridad de memoria. Al integrar el proceso de decisión directamente en cada hilo replicado, el método propuesto elimina el votador centralizado convencional como posible punto de fallo y refuerza la resiliencia frente a *soft errors*.

III. Metodología

El objetivo principal de esta estrategia es proteger el programa frente a *soft-errors* mediante la triplicación por software. Sin embargo, no todas las secciones del código son candidatas adecuadas para su endurecimiento. Esta estrategia utiliza un endurecimiento selectivo de las partes relevantes del código. Una función *wrapper* introduce redundancia y devuelve el resultado votado del código seleccionado. De este modo, la función se encarga de proteger, ejecutar el código en diferentes hilos y validar el resultado correcto.

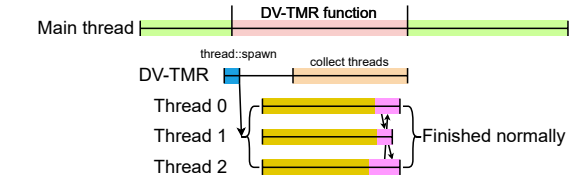
El Algoritmo 1 muestra el pseudocódigo del flujo de trabajo de la técnica. En él se pueden identificar cuatro partes: configuración de los canales de comunicación, creación de hilos (*spawn*), votación y selección del resultado. En primer lugar, se crean tres canales de comunicación, cada uno con un par receptor-transmisor. Cada hilo tendrá uno de estos canales para intercambiar datos y votar de forma independiente. En segundo lugar, se lanzan tres hilos con una copia del contexto necesario para realizar la computación. Por lo tanto, cada hilo ejecuta en paralelo una copia del mismo código y genera un *checksum* del resultado. Una vez finalizada la computación, cada hilo envía su *checksum* al siguiente y recibe el del hilo anterior. En tercer lugar, cada hilo contrasta su propio *checksum* con el recibido para validar su resultado. Finalmente, los tres votos y *checksums* son recopilados por el gestor de hilos y enviados al hilo principal, que recibe un resultado libre de errores. Esta técnica permite detectar un error en el código replicado mediante

Algorithm 1 Flujo de control DV-TMR

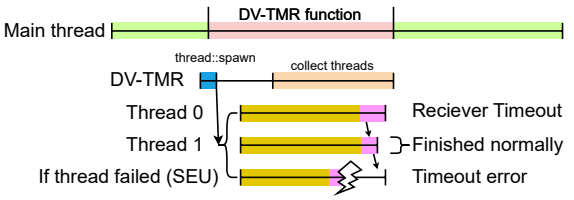
```
1:  Crea tres canales de comunicación
2:  Asigna las rutas de comunicación
3:  Lanza tres hilos (triplicación del flujo)
4:
5:  Triple computación
6:  - [Th_0] : F_0(Data_0) = R_0
7:  - [Th_1] : F_1(Data_1) = R_1
8:  - [Th_2] : F_2(Data_2) = R_2
9:
10: Comunicación de resultados y votación
11: - [Th_0] : R_0 → Thread_1
12: - [Th_0] : Thread_0 ← R_2
13: - [Th_0] : R_0 == R_2
14:
15: - [Th_1] : R_1 → Thread_2
16: - [Th_1] : ∧Thread_1 ← R_0
17: - [Th_1] : R_1 == R_0
18:
19: - [Th_2] : R_2 → Thread_0
20: - [Th_2] : Thread_2 ← R_1
21: - [Th_2] : R_2 == R_1
22:
23: El hilo principal recupera las 3 soluciones y los 3
    votos
24: Si el voto es afirmativo la solución es válida
```

dos mecanismos: votación o *timeouts*. El primero se encarga de los fallos en los datos, que se mitigan al recopilar todos los *checksums* y votos. El sistema de votación puede desembocar en tres situaciones. Cuando todas las copias son correctas, no hay fallo; si una copia es incorrecta, las otras dos pueden validar la solución (mitigado); y finalmente, si los tres resultados son diferentes, no se puede mitigar el error, pero sí reportarlo (detectado). Los *timeouts* advierten si un hilo termina prematuramente. Si un fallo afecta al flujo del programa de un hilo, el sistema puede detectar *timeouts* en el envío o en la recepción del mensaje. En esta situación, si un hilo termina prematuramente, el sistema aún puede producir una solución verificada (error corregido). Sin embargo, si dos hilos terminan prematuramente, el sistema no podrá producir una solución verificada y lo reportará (error detectado).

Para aclarar este caso, la Figura 1 muestra dos escenarios diferentes: uno de ejecución libre de fallos y otro que ilustra cómo funciona la corrección en una situación en la que un hilo falla. El hilo principal se representa como una línea continua; cuando llega a la sección del código protegida, se generan tres hilos desde la función DV-TMR. El hilo principal ahora espera recoger los resultados de los hilos secundarios cuando estos finalicen. Mientras tanto, cada hilo secundario realiza su propio trabajo de forma independiente. Cuando un hilo secundario termina, envía su resultado al siguiente hilo (por ejemplo, el hilo 0 lo envía al hilo 1 y así sucesivamente). Una vez que un hilo recibe el resultado de su vecino, compara su propio resultado y emite un voto: si coinciden,



(a) Ejecución correcta



(b) Escenario de fallo de hilo

Fig. 1: Implementación de DV-TMR en Rust

la solución es correcta; si no coinciden, es porque algún evento ha afectado a la operación en algún punto. Sin embargo, suponiendo como máximo un error por ejecución, siempre existirá una solución verificada. En un escenario que involucra múltiples eventos por ejecución, la estrategia es capaz de detectarlos. No obstante, sería incapaz de aportar una solución verificada. La Figura 1a muestra una ejecución normal, en la que todos los hilos envían su resultado al siguiente hilo y emiten un voto. En esta circunstancia, todos los hilos son correctos y cualquier solución es válida. La Figura 1b muestra una ejecución en la que un *soft error* provoca que un hilo alcance un *timeout*. En este caso, el Thread 0 advertirá sobre el *timeout* en el Thread 2 y enviará su propio valor al Thread 1, que mantendrá la solución correcta y validada.

IV. Configuración del Experimento

A. Hardware

El dispositivo empleado para la experimentación (DUT) fue una Raspberry Pi 4B, un ordenador monoplaca ejecutando una distribución basada en Debian. Esta placa incorpora un System-on-Chip Broadcom BCM2711, que integra un procesador ARM Cortex-A72 de cuatro núcleos y 64 bits con un sistema jerárquico de caché de dos niveles (32KB de L1D y 48KB de L1I por núcleo, y una caché L2 compartida de 1MB [32]).

B. Software

Se utilizó el benchmark de multiplicación de matrices (GeMM) como caso de estudio, debido a su amplio uso como operación básica en muchos sistemas, por ejemplo, en transformaciones espaciales, procesamiento de señales e inteligencia artificial [33].

Nuestro enfoque se basa en las bibliotecas estándar de Rust `thread` y `mpsc`. La biblioteca `thread` incluye soporte para hilos nativos del sistema operativo, cada uno con su propia pila y estado local, y proporciona primitivas para esperar la finalización de un hilo (por ejemplo, `join`) [34]. Los errores lógicos fatales provocarán un *panic* en el hilo, durante el cual este desenrolla la pila, ejecuta destructores y libera los recursos que posee. Si el *panic* no es gestionado por el programador, el hilo termina. El modelo "Multi-Producer, Single-Consumer" utiliza primitivas de cola FIFO (biblioteca `mpsc`) y proporciona comunicación entre hilos mediante canales. Los canales ofrecen tres modos de operación, pero en esta implementación solo se utilizan dos: `sender` y `receiver`.

La implementación de esta metodología requiere cuatro versiones desarrolladas en Rust. Además, se implementó una versión del benchmark GeMM en C como referencia para comparar con Rust. Las versiones de Rust se nombran en función de si el código está protegido (`TMR`) o no protegido (`RAW`), así como del uso de optimizaciones del compilador, denominadas `DLF` cuando no se utilizan optimizaciones y `RLS` cuando las optimizaciones están activas. Por ejemplo, la versión `RLS-RAW` se refiere al código no protegido compilado con opciones de optimización habilitadas.

V. Caracterización

Antes de las campañas de irradiación, todas las versiones fueron caracterizadas tanto estáticamente como dinámicamente, utilizando la implementación en C como referencia para Rust.

A. Caracterización estática

La Tabla I muestra el análisis del tamaño del binario para cada versión, centrándose en las secciones `.text`, `.data` y `.bss`.

Tabla I: Ocupación de las secciones en MB

Version name	.text	.data	.bss
DFL-RAW (C version)	0.004	3.754	0.500
DFL-RAW	0.507	5.719	0.000
DFL-TMR	0.585	4.503	0.001
RLS-RAW	0.314	5.634	0.000
RLS-TMR	0.351	4.402	0.001

La versión en C presenta una sección `.text` muy compacta, mientras que la mayor parte del tamaño del binario proviene de la sección `.data`, donde se almacenan los distintos datos de entrada (10 de los operandos A, B y Golden), y finalmente la sección `.bss`, donde las matrices para el GeMM se copian y almacenan (A, B, Golden y Result).

Las versiones en Rust muestran secciones `.text` más grandes, entre ×78 y ×117 mayores que la versión en C. Este incremento puede atribuirse a las rutinas adicionales introducidas para reforzar la seguridad de memoria y otras características heredadas del compilador LLVM. La técnica de protección incrementa esta huella de código hasta un 10 % en comparación con las versiones no protegidas. Sin embargo, tras aplicar optimizaciones

del compilador (verde), esta huella disminuye aproximadamente un 40 %.

La segunda diferencia radica en la gestión de la memoria en las secciones `.bss` y `.data`. En el caso de Rust, la sección `.data` almacena todas las matrices y otras variables empleadas en el benchmark. Se observa una reducción de casi 30 % en las versiones protegidas (amarillo); la razón es la ausencia de necesidad de una matriz golden y el hecho de que las matrices de resultado pertenecen a la pila de cada hilo, reduciendo efectivamente el espacio necesario en la sección `.data`.

En cuanto a la sección de variables no inicializadas (`.bss`), Rust inicializa todas las variables por defecto, dejando esta sección casi vacía. Solo en el caso de las versiones protegidas (azul) se observa su uso. Una inspección más profunda muestra que los punteros a las pilas de cada hilo se declaran aquí.

B. Caracterización dinámica

Para evaluar cómo pequeñas diferencias estáticas afectan a la ejecución, se utiliza la herramienta `linux-perf` para recopilar información de las Performance Monitoring Units (PMUs) durante la ejecución [35].

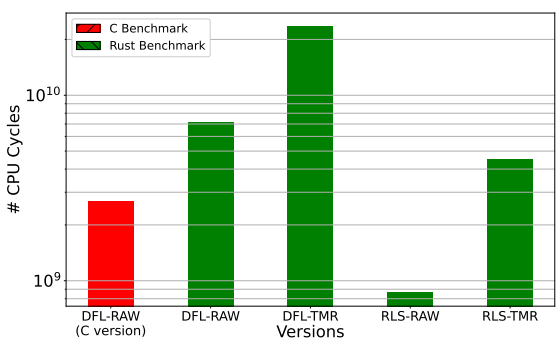


Fig. 2: Coste del benchmark en ciclos de CPU

La Figura 2 muestra el coste en ciclos de CPU de las distintas versiones para realizar una única multiplicación. La figura revela una tendencia predecible: Rust por defecto requiere un mayor número de ciclos que C para completar la ejecución, y las versiones protegidas muestran un incremento aún mayor. Sin embargo, las versiones optimizadas muestran una reducción notable en comparación con sus versiones análogas. Específicamente, antes de la implementación de optimizaciones en las versiones `DFL`, Rust requiere entre 2,7× y 3,0× más ciclos de CPU para ejecutar el benchmark. Este comportamiento se correlaciona con la mayor sección `.text` observada en el análisis estático (procedimientos de validación de la integridad de la memoria). Sin embargo, en las versiones `RLS`, el coste del benchmark en Rust se reduce a 0,33× respecto del benchmark en C y aumenta hasta 1,70× en el caso de `TMR`, mostrando la capacidad del compilador para optimizar Rust.

La actividad de caché y memoria (hits y misses) se presenta en la Figura 3, de izquierda a derecha: número de referencias a la caché L1D, Misses en L1D, Referencias a L2 y Misses en L2 para cada una de las versiones. También se proporciona el cálculo teórico de los valores esperados de uso de la caché, calculado en función del tamaño de las matrices, el tamaño de las cachés y las operaciones de bajo nivel realizadas por el GeMM.

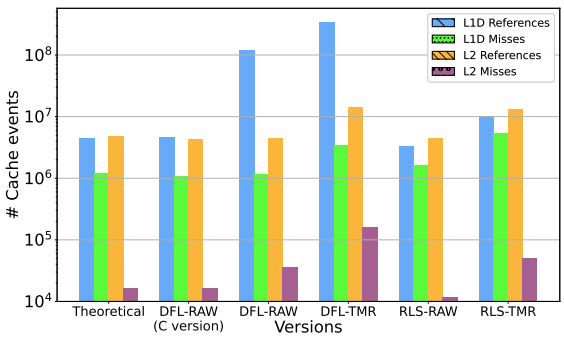


Fig. 3: Eventos de caché L1 y L2 medidos para cada versión

La figura 3 muestra que, por un lado, la versión en C mantiene un perfil de uso de caché que se aproxima estrechamente al cálculo teórico. Por otro lado, Rust presenta un incremento de 25× en el número de referencias a L1D en las versiones `RAW` respecto a los cálculos teóricos. Este comportamiento puede atribuirse a las múltiples comprobaciones de memoria introducidas en tiempo de compilación, que permiten a Rust garantizar la seguridad de la memoria. A pesar del aumento observado en las referencias, el número de fallos generados en la caché L1D permanece similar a los cálculos teóricos y es comparable al de C. Las referencias a L2 tampoco se ven afectadas por la política de seguridad de memoria de Rust; sin embargo, el número de fallos aumenta entre 2,2 ∼ 3,3× en las versiones `RAW`. En el caso de Rust, la inclusión de optimizaciones del compilador representa una reducción notable en las referencias a L1D, reduciendo el número de 25× a 0,73× en comparación con los cálculos teóricos. Esto representa solo el 3 % de las referencias utilizadas por la versión `DFL-RAW`. Los fallos en L1D también experimentan un aumento de 1,4× debido a las optimizaciones. Mientras que las referencias a L2 permanecen aparentemente inalteradas, el número de fallos en L2 se reduce al 30 ∼ 33 % de las versiones `DFL`.

VI. Experimentación bajo Radiación

Los experimentos se llevaron a cabo en la instalación del Institute Laue-Langevin (ILL), utilizando la estación de irradiación de neutrones térmicos y epitérmicos (TENIS), véase la Figura 4. En esta instalación, el DUT fue expuesto a neutrones con un amplio espectro de energías, incluyendo neutrones de 0.1 a 10 MeV, con los mayores flujos en la región de neutrones térmicos (∼25 meV) [36]. La experimentación siguió el estándar European Space Components Coordination (ESCC) 25100 [37], que

define las metodologías de ensayo para evaluar la resiliencia bajo entornos de radiación controlados.

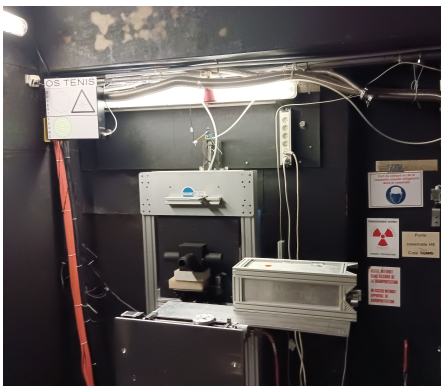


Fig. 4: Montaje experimental en el reactor nuclear del ILL, Grenoble

El montaje constaba de dos dispositivos: el DUT (una Raspberry Pi 4B) y un Control Computer (CC) que monitorizaba la ejecución y gestionaba los reinicios en caso de cuelgue. El DUT arrancaba mediante el Pre-eXecution Environment (PXE), mientras que el CC utilizaba NFS para almacenar los registros en tiempo real.

VII. Discusión de Resultados

Esta sección presenta las métricas de fiabilidad empleadas, incluyendo la cross-section (*cross-section*) y MWTF. La cross-section (σ) expresa la probabilidad de que una partícula produzca un evento en el sistema al atravesarlo. Se determina dividiendo el número de eventos entre la fluencia (Φ) (ecuación 1 izquierda). El Mean Work to Failure (MWTF) captura el compromiso entre la tasa de error y el tiempo de ejecución, y representa la cantidad media de trabajo que puede completarse antes de que ocurra un error. Se estima como el inverso de la cross-section multiplicado por el tiempo medio de ejecución (ecuación 1 derecha) [39].

$$\sigma = \frac{\#SDC}{\Phi} \qquad MWTF = \frac{1}{\sigma \cdot \overline{T_{exec}}} \tag{1}$$

Cada experimento alcanzó una fluencia superior a 10^9 n/cm^2 y registró una media de más de 50 eventos. En cuanto al tiempo medio de ejecución, se observa que las mejoras se deben principalmente a las optimizaciones del compilador, pasando de $1{,}3 \sim 1{,}4s$ en `DFL` a $0{,}1 \sim 0{,}2s$ en `RLS`. Además, es destacable que la pérdida de rendimiento se reduce entre la versión protegida y la no protegida debido al uso de más hilos (`DFL` de 1,3 a 1,4s y `RLS` de 0,1 a 0,2). El número de eventos demuestra que la técnica puede mitigar algunos errores (44 y 16) y detectar todos los eventos no corregibles (37 y 16). Estos efectos permiten confirmar que las técnicas pueden manejar múltiples errores en una misma ejecución.

Pasando a la serie de cross-section de SDC, las versiones protegidas funcionan como se esperaba, dando como resultado una reducción de la cross-section de SDC en más de un orden de magnitud, específicamente $\times 80{,}5$ para la versión `DFL` y $\times 26{,}8$ para la versión `RLS`. Sin embargo, al observar las secciones eficaces de todos los eventos, los resultados muestran incrementos moderados ($1{,}13 \sim 1{,}27$) en el valor de la cross-section en comparación con la versión de referencia, principalmente debido a las optimizaciones del compilador y al código adicional para la gestión de hilos.

En el caso del MWTF, la baja cross-section de `SDC` de `TMR` y el rendimiento de `RLS` conducen a mejores valores de MWTF. Observando las versiones `RLS`, incluso la versión no protegida `RLS-RAW` muestra un incremento de $\times 68{,}3$, mientras que la versión protegida `RLS-TMR` alcanza la puntuación más alta con un incremento de $\times 121$.

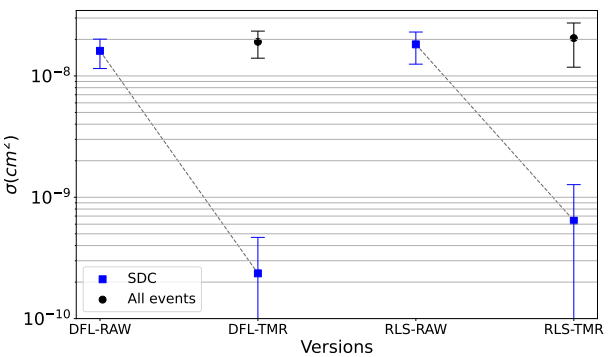


Fig. 5: Cross-section con intervalos de confianza del 95 %

La Figura 5 muestra claramente las tendencias de la incorporación de protección en el software. De acuerdo con la serie de eventos detectados, añadir protección conduce a un ligero incremento de la cross-section. Esto se explica por la sobrecarga de añadir código de protección al *benchmark*, lo que lo vuelve más susceptible a fallos inducidos por la radiación. Sin embargo, como se muestra en la serie de SDC, la técnica detecta o mitiga con éxito estos fallos, mejorando la cross-section hasta $80{,}5\times$.

VIII. Conclusiones

Este trabajo presentó un enfoque de redundancia modular triple distribuida (DV-TMR) implementado en Rust para mejorar la tolerancia a fallos del software en plataformas COTS multinúcleo expuestas a radiación ionizante. El diseño propuesto distribuye la lógica de votación entre subprocesos redundantes mediante el paso de mensajes, eliminando el votador centralizado como punto de fallo asociado.

La técnica se evaluó utilizando un benchmark de multiplicación de matrices. La caracterización estática y dinámica reveló que Rust produce binarios más grandes que C debido a sus mecanismos de seguridad, aunque las optimizaciones del compilador reducen significativamente la sobrecarga de ejecución, lo que da un rendimiento comparable al de las implementaciones en C.

Los experimentos de radiación realizados en las instalaciones del ILL TENIS indican que la técnica

Tabla II: Resultado experimentales de la campaña de radiación

Versión	Tiempo Ejecución (s)	$\Phi \cdot 10^9$ (n/cm^2)	#Events SDC	Detected	Mitigated	$\sigma \cdot 10^{-8}(cm^2)$ SDC	Events	$MWTF_{SDC}$ $(\times 10^8)$
DFL-RAW	1,3471	4,17	67	–	–	1,61 –	1,61 –	0,458
DFL-TMR	1,4435	4,23	0	37	44	0,02* ⇓ ×80,5	1,91 ↑ ×1,19	31,267*
RLS-RAW	0,1327	3,07	56	–	–	1,82 ↑ ×1,13	1,82 ↑ ×1,13	2,606
RLS-TMR	0,2560	1,56	0	16	16	0,06* ⇓ ×26,8	2,05 ↑ ×1,27	55,774*

* No se observaron errores, a efectos comparativos, se calcula suponiendo un error (peor caso) [38].

DV-TMR detectó o mitigó con éxito todos los eventos de corrupción silenciosa de datos. Aunque las versiones protegidas mostraron un ligero aumento en la sección transversal total de eventos debido a la sobrecarga adicional, las variantes optimizadas alcanzaron valores de trabajo medio hasta el fallo (MWTF) sustancialmente más altos.

Estos resultados respaldan la idoneidad de Rust para implementar soluciones TMR de software en entornos propensos a la radiación. El enfoque DV-TMR propuesto representa un paso práctico hacia la mejora de la tolerancia a fallos del software tolerante a fallos en plataformas COTS. En el futuro se ampliará la evaluación a otras aplicaciones y estrategias de redundancia.

Agradecimientos

La investigación descrita en este artículo ha sido financiada en parte por los siguientes proyectos: MultiRad (financiado por la Región Auvernia-Ródano-Alpes, Francia); IRT Nanoelec (proyecto ANR-10-AIRT-05 de la Agencia Nacional de Investigación francesa, financiado a través del Programa de inversión para el futuro); plataforma UGA/LPSC/GENESIS y PID2022-138696OB-C22 (financiado por el Ministerio de Ciencia e Innovación de España).

[32]

Referencias

[1] Kun Ma, Ningfang Song, Jing Jin, Enrico Zio, Jiliang He, Xiaowei Wang, and Linghai Kong, "Radiation-induced degradation analysis and reliability modeling of cots adcs for space-borne miniature fiber-optic gyroscopes," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–8, 2021.

[2] Charles J. Corley, Heather M. Quinn, and Earl E. Swartzlander, "Accelerated nuclear radiation effects on the raspberry pi 3b+," in *2022 IEEE Radiation Effects Data Workshop (REDW) (in conjunction with 2022 NSREC)*, 2022, pp. 1–8.

[3] Pankaj Jalote, *Software Fault Tolerance: Techniques and Implementation*, Prentice Hall, Englewood Cliffs, NJ, USA, 1994.

[4] Flaviu Cristian, Christof Fetzer, et al., "Software fault tolerance: State of the art," NASA Technical Memorandum NASA/TM-2000-210616, NASA Langley Research Center, 2000, Online, accessed on October 2025.

[5] Sharon Hudson, R. S. Shyama Sundar, and Srinivas Koppu, "Fault control using triple modular redundancy (tmr)," in *Progress in Computing, Analytics and Networking*, Prasant Kumar Pattnaik, Siddharth Swarup Rautaray, Himansu Das, and Janmenjoy Nayak, Eds., Singapore, 2018, pp. 471–480, Springer Singapore.

[6] Yuanqing Li, Anselm Breitenreiter, Marko Andjelkovic, Junchao Chen, Milan Babic, and Milos Krstic, "Double cell upsets mitigation through triple modular redundancy," *Microelectronics Journal*, vol. 96, pp. 104683, 2020.

[7] Michael Rogenmoser, Yvan Tortorella, Davide Rossi, Francesco Conti, and Luca Benini, "Hybrid modular redundancy: Exploring modular redundancy approaches in risc-v multi-core computing clusters for reliable processing in space," *ACM Trans. Cyber-Phys. Syst.*, vol. 9, no. 1, 2025.

[8] Paul C. van Oorschot, "Memory errors and memory safety: C as a case study," *IEEE Security & Privacy*, vol. 21, no. 2, pp. 70–76, Mar. 2023.

[9] Mingbo Zhang and Saman Zonouz, "Use-after-free mitigation via protected heap allocation," in *2018 IEEE Conference on Dependable and Secure Computing (DSC)*. Dec. 2018, p. 1–8, IEEE.

[10] Steve Klabnik and Carol Nichols, *The Rust programming language*, No Starch Press, 2023.

[11] Abhiram Balasubramanian, Marek S. Baranowski, Anton Burtsev, Aurojit Panda, Zvonimir Rakamari, and Leonid Ryzhyk, "System programming in rust: Beyond safety," *SIGOPS Oper. Syst. Rev.*, vol. 51, no. 1, pp. 94–99, 2017.

[12] H. Heyman and L. Brandefelt, "A comparison of performance & implementation complexity of multithreaded applications in rust, java and c++," Bachelor's thesis, KTH School of Electrical Engineering and Computer Science (EECS), 2020.

[13] Jaemin Hong and Sukyoung Ryu, "Automatically translating c to rust," *Commun. ACM*, vol. 68, no. 11, pp. 58–65, 2025.

[14] Mohammadreza Amel Solouki, Shaahin Angizi, and Massimo Violante, "Dependability in embedded systems: A survey of fault tolerance methods and software-based mitigation techniques," *IEEE Access*, vol. 12, pp. 180939–180967, 2024.

[15] Sanem Arslan and Osman Unsal, "Efficient selective replication of critical code regions for sdc mitigation leveraging redundant multithreading," *The Journal of Supercomputing*, vol. 77, no. 12, pp. 14130–14160, 2021.

[16] Hwisoo So, Moslem Didehban, Aviral Shrivastava, and Kyoungwoo Lee, "A software-level redundant multithreading for soft/hard error detection and recovery," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 1559–1562.

[17] Kiril Dichev and Dimitrios S. Nikolopoulos, "Twinpcg: Dual thread redundancy with forward recovery for preconditioned conjugate gradient methods," in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, 2016, pp. 162–163.

[18] Alejandro Serrano-Cases, Felipe Restrepo-Calle, Sergio Cuenca-Asensi, and Antonio Martínez-Álvarez, "Multi-threaded mitigation of radiation-induced soft errors in bare-metal embedded systems," *Journal of Electronic Testing*, vol. 36, no. 1, pp. 47–57, 2019.

[19] Oussama Tahan and Mohamed Shawky, "Using dynamic task level redundancy for openmp fault tolerance," in *Architecture of Computing Systems – ARCS 2012*, Andreas Herkersdorf, Kay Römer, and Uwe Brinkschulte, Eds., Berlin, Heidelberg, 2012, pp. 25–36, Springer Berlin Heidelberg.

[20] Adrian Munera, Eduardo Quiñones, and Sara Royuela, "Guardianomp: A framework for highly productive fault tolerance via openmp task-level replication," in *2025*

IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2025, pp. 1–12.

[21] Pietro Fara, Gabriele Serra, Alessandro Biondi, and Ciro Donnarumma, "Scheduling replica voting in fixed-priority real-time systems," in *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*. 2021, Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[22] Oscar Medina Duarte and Reinhard Hametner, "Asynchronous programming with futures in c on a safety-critical platform in the railway-control domain," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, pp. 1–8.

[23] José D'Abruzzo Pereira, Naghmeh Ivaki, and Marco Vieira, "Characterizing buffer overflow vulnerabilities in large c/c++ projects," *IEEE Access*, vol. 9, pp. 142879–142892, 2021.

[24] Hao Sun, Xiangyu Zhang, Chao Su, and Qingkai Zeng, "Efficient dynamic tracking technique for detecting integer-overflow-to-buffer-overflow vulnerability," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, New York, NY, USA, 2015, ASIA CCS '15, p. 483–494, Association for Computing Machinery.

[25] Alyssa Milburn, Herbert Bos, and Cristiano Giuffrida, "Safeinit: Comprehensive and practical mitigation of uninitialized read vulnerabilities," in *Proceedings 2017 Network and Distributed System Security Symposium*. 2017, NDSS 2017, Internet Society.

[26] Robin Müller, Paul Nehlich, and Sabine Klinkner, "Leveraging the rust programming language for space applications," in *2024 IEEE Space Computing Conference (SCC)*, 2024, pp. 40–50.

[27] Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer, "Rustbelt: securing the foundations of the rust programming language," *Proc. ACM Program. Lang.*, vol. 2, no. POPL, 2017.

[28] Boqin Qin, Yilun Chen, Zeming Yu, Linhai Song, and Yiying Zhang, "Understanding memory and thread safety practices and issues in real-world rust programs," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 763–779.

[29] Anton Burtsev, Dan Appel, David Detweiler, Tianjiao Huang, Zhaofeng Li, Vikram Narayanan, and Gerd Zellweger, "Isolation in rust: What is missing?," in *Proceedings of the 11th Workshop on Programming Languages and Operating Systems*, New York, NY, USA, 2021, p. 76–83, Association for Computing Machinery.

[30] Arthur Lafrance, David Detweiler, Zhaofeng Li, Xiangdong Chen, Vikram Narayanan, and Anton Burtsev, "Extending rust with support for zero copy communication," in *Proceedings of the 12th Workshop on Programming Languages and Operating Systems*, New York, NY, USA, 2023, p. 74–82, Association for Computing Machinery.

[31] Jacopo Sini, Mohammadreza Amel Solouki, Massimo Violante, and Giorgio Di Natale, "Improving software reliability with rust: Implementation for enhanced control flow checking methods," in *2025 Design, Automation & Test in Europe Conference (DATE)*, 2025, pp. 1–7.

[32] ARM Community, "Arm cortex-a72 mpcore processor technical reference manual," 2025, Online, accessed on November 2025.

[33] Shixun Wu, Yujia Zhai, Jinyang Liu, Jiajun Huang, Zizhe Jian, Bryan Wong, and Zizhong Chen, "Anatomy of high-performance gemm with online fault tolerance on gpus," in *Proceedings of the 37th ACM International Conference on Supercomputing*. 2023, ICS '23, p. 360–372, Association for Computing Machinery.

[34] Rust docs, "std::thread - rust," 2025, Online, accessed on November 2025.

[35] Claudio Scheer, Renato B. Hoffmann, Dalvan Griebler, Isabel H. Manssour, and Luiz G. Fernandes, "Performance data visualization of linux events on multicores," in *Anais do XXII Simposio em Sistemas Computacionais de Alto Desempenho (SSCAD 2021)*. 2021, SSCAD 2021, p. 108–119, Sociedade Brasileira de Computação.

[36] ILL, "Tenis: Thermal and epi-thermal neutron irradiation station," 2025, Online, accessed on October 2025.

[37] ESCC, "Single event effects test method and guidelines, escc basic specification no. 25100," Specification, Issue 2, October 2014, 2014, European Space Agency / ESCC, Paris, France.

[38] Heather Quinn and George Tompkins, "Measuring zero: Neutron testing of modern digital electronics," *IEEE Transactions on Nuclear Science*, vol. 71, no. 4, pp. 670–679, 2024.

[39] G.A. Reis, J. Chang, N. Vachharajani, S.S. Mukherjee, R. Rangan, and D.I. August, "Design and evaluation of hybrid fault-detection systems," in *32nd International Symposium on Computer Architecture (ISCA'05)*, 2005, pp. 148–159.

Análisis Experimental de la Sincronización de LoRaWAN Clase B: Hacia Redes IoT Integradas con NTN

María Ángeles Amador[1], Celia Garrido-Hidalgo[2], Luis Roda-Sanchez[2] y Teresa Olivares[3]

Resumen— **La expansión de las redes no terrestres (NTN) para la conectividad global del Internet de las Cosas (IoT) plantea importantes desafíos en términos de sincronización y eficiencia energética. La Clase B de LoRaWAN permite la comunicación descendente programada mediante ventanas de recepción periódicamente sincronizadas (ping slots), aunque su rendimiento práctico ha sido poco evaluado experimentalmente.**

Este trabajo presenta una evaluación experimental de la precisión de sincronización y del consumo energético de LoRaWAN Clase B mediante un banco de pruebas compuesto por cuatro dispositivos STM32 B-L072Z-LRWAN1, un gateway Lorank8 con referencia GPS y un servidor de red ChirpStack. Se analizaron cuatro configuraciones de ping slots (PS64, PS32, PS16 y PS8) utilizando mediciones temporales obtenidas con un analizador lógico.

Los resultados muestran que el error absoluto de sincronización aumenta de 1.5 ms a 11.5 ms al incrementarse el período del ping slot, mientras que el error relativo se mantiene por debajo del 0.02 %, dominado por la deriva del oscilador (∼190 ppm). La deriva acumulada intra-beacon alcanza 23.6 ms, dentro de la ventana de recepción de 30 ms definida por la especificación LoRaWAN. Asimismo, el uso del modo de bajo consumo (sleep) reduce la corriente media de 16.80 mA a 6.40 mA (62 %). Estos resultados proporcionan una caracterización empírica de la operación de Clase B y aportan información relevante para el diseño de redes IoT y futuras arquitecturas híbridas terrestre–NTN.

Palabras clave— **LoRaWAN Clase B, Precisión de Sincronización, Consumo Energético, Ping Slots, Redes IoT, NTN.**

I. Introducción

LORAWAN es una tecnología LPWAN clave para IoT, ofreciendo comunicación de largo alcance y bajo consumo [1]. Define tres clases de dispositivo: A (bajo consumo, downlink solo tras uplink), C (recepción continua, alto consumo) y B (equilibra consumo energético y capacidad de downlink). La Clase B introduce *ping slots* o ventanas de recepción periódicas y sincronizadas mediante beacons enviados por el Gateway (GW). Esto permite un downlink determinista y eficiente, crucial para aplicaciones como agricultura inteligente o control industrial [2].

Sin embargo, el despliegue terrestre tiene limitaciones de cobertura. La integración con redes no-terrestres (NTN), como satélites de órbita baja (LEO), es una alternativa prometedora [3]. En este contexto, la sincronización precisa de la Clase B es aún más crítica debido al entorno satelital dinámico.

A pesar de su potencial, la Clase B es la menos explorada experimentalmente. La mayoría de estudios son simulaciones [4], [5], [6], existiendo una brecha en la validación con hardware real. Este trabajo llena ese vacío mediante el diseño y evaluación de un banco de pruebas con dispositivos reales. Se mide la precisión de sincronización para diferentes configuraciones de ping slots, se analiza la acumulación de errores y se caracteriza el consumo energético, estableciendo una base para futuras integraciones con NTN y redes 6G.

II. Fundamentos de LoRaWAN Clase B

En la Clase B, los dispositivos finales (EDs) se sincronizan periódicamente mediante beacons temporizados con referencia GPS, lo que les permite abrir ventanas de recepción en instantes predefinidos para la recepción de downlinks. A diferencia de la Clase A, donde los downlinks solo siguen a un uplink, la Clase B permite ventanas de recepción programadas dentro de cada período de beacon.

El ED se une inicialmente en modo Clase A (OTAA o ABP). Tras el `JoinAccept`, solicita la referencia temporal mediante `DeviceTimeReq`/`DeviceTimeAns`. Posteriormente, el Servidor de Red (NS) configura la periodicidad y canal de ping slots mediante `PingSlotInfoReq` y `PingSlotChannelReq`. La periodicidad (0–7) define el intervalo entre aperturas de ping slots según la especificación LoRaWAN:

$$T_{\text{ping}} = 0{,}96 \cdot 2^{\text{Periodicidad}} \text{ s.} \tag{1}$$

Los GWs transmiten beacons cada 128 s, sincronizados con GPS. El instante de transmisión del beacon puede expresarse como:

$$B_T = k \cdot 128 + T_{\text{RetardoBeacon}}, \tag{2}$$

donde $T_{\text{RetardoBeacon}} \approx 1{,}5$ ms representa el retardo de conmutación RX–TX [1]. Los EDs utilizan el inicio del beacon como referencia temporal para compensar deriva de reloj y latencias de procesamiento.

A partir de esta referencia temporal, el dispositivo calcula los instantes de apertura de los ping slots dentro de cada período de beacon. Cada ventana de recepción tiene una duración de 30 ms y su posición temporal viene dada por:

[1]Instituto de Investigación en Informática de Albacete (I3A), Universidad de Castilla-La Mancha, e-mail: `MariaAngeles.Amador@uclm.es`

[2]Dpto. de Ingeniería Eléctrica, Electrónica, Automática y Comunicaciones, Universidad de Castilla-La Mancha, e-mail: `{Celia.Garrido,Luis.Roda}@uclm.es`

[3]Dpto. de Sistemas Informáticos, Universidad de Castilla-La Mancha, e-mail: `Teresa.Olivares@uclm.es`

$$T_{\text{slot}} = T_{\text{reservado}} + \left[d_{\text{Ping}} + N \cdot \frac{4096}{\text{numPings}} \right] \cdot 30\,\text{ms} \tag{3}$$

donde N representa el índice del ping slot dentro del período de beacon y d_{Ping} es un desplazamiento pseudoaleatorio que distribuye los slots entre dispositivos para reducir colisiones. Este desplazamiento se calcula como:

$$d_{\text{Ping}} = \text{AES}_{0^{16}}(\text{tiempoBeacon} \,\|\, \text{DireccionED}) \;\text{mód}\; \left(\frac{4096}{\text{numPings}} \right) \tag{4}$$

El período fijo $T_{\text{beacon}} = 128$ s se divide en intervalo reservado (2.120 s), guarda (3.000 s) y ventana utilizable:

$$T_{\text{utilizable}} = 122{,}880 \text{ s}$$

Estos intervalos son deterministas y no contribuyen al error de sincronización; las desviaciones analizadas en este trabajo provienen exclusivamente de deriva del oscilador y latencias de procesamiento.

Si se pierde un beacon, el ED amplía progresivamente su ventana de escucha, manteniendo la operación hasta 120 minutos sin resincronización. Superado este tiempo, vuelve a Clase A.

Cada beacon incluye marca de tiempo GPS (4 bytes) y CRC-16, además de campos opcionales `GwSpecific` para soporte de movilidad. En caso de cambio de GW, el ED actualiza la ruta de downlink mediante uplink de Clase A, aplicando un retardo aleatorio para evitar congestión [1].

III. Trabajo Relacionado

La literatura muestra avances relevantes en la optimización de LoRaWAN Clase B, aunque persiste una clara falta de validación experimental, especialmente en sincronización y consumo energético.

En el ámbito de la optimización del downlink, Ron et al. [4] propusieron un modelo analítico M/G/1 para ajustar el número de ping slots, logrando reducir el retardo hasta un 64 % a costa de mayor consumo energético. Sin embargo, asumieron sincronización y condiciones de GW ideales. De forma similar, Elbsir et al. [5] compararon Clase A y B mediante NS-3, mostrando que Clase B reduce significativamente la latencia (de 923 s a 52 s) y la pérdida de paquetes, aunque aumenta colisiones bajo congestión. Posteriormente, ampliaron este análisis identificando que las limitaciones de duty cycle en el GW explican el 55 % de las pérdidas en downlink, y demostraron que deshabilitar RX2 puede mejorar el Packet Delivery Ratio (PDR) del 38 % al 82 % [5]. No obstante, ambos trabajos se basan exclusivamente en simulación.

En entornos industriales, Todoli-Ferrandis et al. [7] incorporaron un mecanismo ADR bidireccional para Clase B, reduciendo la pérdida de paquetes en un 22 %, aunque bajo modelos de canal idealizados. Jagatha et al. [8] también emplearon NS-3 para analizar Clase B y C, destacando mejoras en tráfico confirmado mediante retransmisiones y diversidad de recepción.

Respecto al consumo energético, Cheong et al. [9] realizaron una comparación experimental entre clases, concluyendo que Clase A puede alcanzar hasta 10 años de vida útil, mientras que Clase C resulta ineficiente. Sin embargo, excluyeron explícitamente Clase B, dejando abierto el análisis experimental de su consumo.

En el contexto DtS-IoT, Qu et al. [10] definieron una arquitectura LoRaWAN sobre satélites LEO, y Ullah et al. [11] evaluaron mediante simulaciones Monte Carlo la viabilidad uplink directa e indirecta, destacando la influencia del Spreading Factor (SF), la densidad de nodos y la movilidad satelital. Más recientemente, Rolland et al. [12] analizaron la sincronización de beacons en escenarios satelitales proponiendo una estrategia "listen-first" con ahorros energéticos del 33 %, aunque sin validación hardware.

En el ámbito experimental, Tang et al. [13] proporcionaron un extenso dataset urbano de mediciones RSSI y SNR con GWs aéreas y terrestres, demostrando mejoras significativas en cobertura uplink, pero sin abordar sincronización ni downlink de Clase B.

Finalmente, en un trabajo previo de los autores [14], se validó experimentalmente un esquema TDMA sobre Clase A, logrando una precisión de sincronización de 10 ms y mejoras de PDR del 29 % frente a ALOHA. Las limitaciones identificadas — especialmente en sincronización basada en beacons y deriva de reloj— motivan directamente el presente estudio.

En conjunto, aunque existen aportaciones analíticas y de simulación relevantes, la validación experimental integral de sincronización, consumo energético y rendimiento de downlink en Clase B sigue siendo limitada, particularmente en escenarios realistas y satelitales.

IV. Configuración Experimental y Metodología

Este estudio emplea un banco de pruebas práctico diseñado para evaluar el rendimiento y la precisión temporal de las operaciones de Clase B de LoRaWAN en condiciones realistas. El banco de pruebas implementa la arquitectura LoRaWAN estándar, compuesta por 4 EDs, un GW y un NS. La métrica principal de evaluación es la precisión de sincronización entre los beacons del GW y los ping slots de los ED, que es crítica para el rendimiento de la Clase B.

Se utiliza la placa STM32 B-L072Z-LRWAN1 como dispositivo principal en este estudio, debido a su integración de un microcontrolador de bajo consumo STM32L072CZ y el transceptor Semtech SX1276, que proporciona un rendimiento de radio estable y bien documentado en el rango de 860–930 MHz y soporte de Clase B. Además, esta plataforma es ampliamente utilizada en investigación académica e industrial [14], lo que facilita la reproducibilidad y la

comparación con trabajos previos.

El GW empleado en este estudio es el Lorank8 [15], cuya arquitectura se basa en una SeeedStudio BeagleBone Green [16] ejecutando el software *packet forwarder*. Su módulo de Radiofrecuencia (RF) consiste en un concentrador iC880A equipado con un procesador de señal Semtech SX1301, lo que permite el manejo de múltiples canales simultáneos [17]. El módulo GPS Neo-6M proporciona la referencia de temporización precisa requerida para la Clase B, generando una señal de Pulso Por Segundo (PPS) con precisión inferior a un microsegundo para la sincronización de beacons [18].

En el lado de la red, desplegamos una instancia del NS ChirpStack. La arquitectura modular de ChirpStack fue fundamental para la gestión de dispositivos, el manejo de comandos MAC y, crucialmente, la planificación y monitorización de los downlinks de Clase B [19]. La configuración física completa del banco de pruebas se muestra en la Figura 1. Integra los EDs seleccionados, el GW Lorank8 con su módulo GPS, el analizador lógico para las mediciones de temporización y la unidad de monitorización de consumo. Se incluye un botón de inicio físico para desencadenar la activación simultánea de la comunicación LoRaWAN en todos los EDs, asegurando una inicialización sincronizada del experimento.

Los EDs utilizados en este trabajo se programaron usando STM32CubeIDE y el paquete de expansión de software I-CUBE-LRWAN para LoRaWAN. El GW ejecuta el *packet forwarder* de código abierto [20]. Este software reenvía paquetes RF bidireccionalmente entre el concentrador y el NS a través de IP/UDP. La arquitectura de software del GW incluye el repositorio `lora_gateway` [21] (HAL/controlador para SX1301), actualizado a las versiones 4.0.1 y 5.0.1 para soporte de Clase B.

La evaluación de la precisión temporal requirió instrumentación precisa. Monitorizamos los estados de Transmisión (TX) y Recepción (RX) del ED activando los pines de Entrada/Salida de Propósito General (GPIO) PA8 y PA10, respectivamente. Simultáneamente, en el GW, configuramos GPIO50 para señalizar el inicio y fin de cada transmisión de beacon. Estas señales digitales fueron capturadas usando un analizador lógico de 8 canales AZ-Delivery [22], muestreando a 24 Msps, para medir la alineación temporal entre las transmisiones de beacon y las aperturas de ping slots con alta precisión. La Tabla I resume las conexiones GPIO utilizadas con fines de medición, incluyendo la detección de beacons, la actividad de uplink/downlink y los ping slots.

Se probaron cuatro configuraciones de ping slots: PS64, PS32, PS16 y PS8, con SF10 y ancho de banda de 125 kHz. La temperatura ambiente se mantuvo estable ($\approx 31°C$) para aislar el efecto de la deriva térmica [14].

Tabla I: Conexiones físicas entre los componentes del banco de pruebas.

Dispositivo Origen	Pin	Destino	Pin Destino
Conexiones del Concentrador iC880A			
iC880A	21	BBG	9_5 (5V)
iC880A	22	BBG	9_1 (GND)
iC880A	14	BBG	9_22 (CLK)
iC880A	15	BBG	9_21 (D0)
iC880A	16	BBG	9_18 (D1)
iC880A	17	BBG	9_17 (CS0)
iC880A	13	BBG	9_26 (RESET)
Conexiones del Módulo GPS			
GPS	VCC	BBG	9_6 (5V)
GPS	GND	BBG	9_2 (GND)
GPS	TX	BBG	9_11 (RXD)
GPS	RX	BBG	9_13 (TXD)
GPS	PPS	iC880A	Pin 19
Conexiones del Analizador Lógico			
Analizador Lógico	CH1	BBG	9_14 (GPIO50)
Analizador Lógico	CH2	ED	PA8 (TX)
Analizador Lógico	CH3	ED	PA10 (RX)

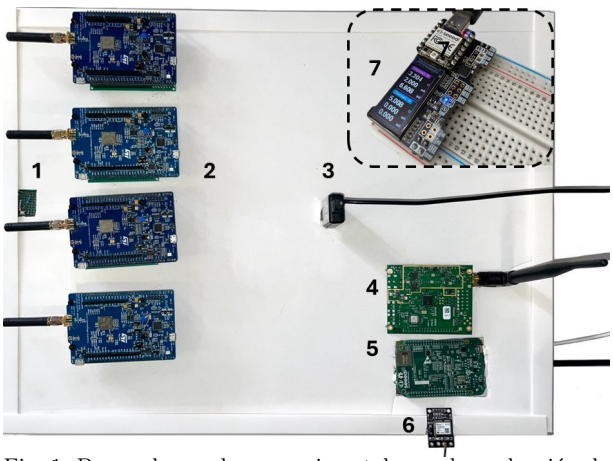


Fig. 1: Banco de pruebas experimental para la evaluación de Clase B: (1) botón de inicio, (2) 4 EDs STM32 B-L072Z-LRWAN1, (3) analizador lógico de 8 canales monitorizando señales GPIO de temporización, (4) concentrador iC880A, (5) BeagleBone Green ejecutando el packet forwarder, (6) GPS Neo-6M proporcionando sincronización PPS, y (7) XIAO PowerBread para la medición de consumo.

V. Resultados y Discusión

Esta sección presenta los resultados de tres experimentos diseñados para evaluar los aspectos temporales y energéticos de la operación de Clase B. Los experimentos evalúan (i) la precisión de sincronización bajo diferentes densidades de ping slots, (ii) el consumo medio de corriente en modos siempre activo y reposo (sleep), y (iii) la acumulación del error de temporización intra-beacon. En conjunto, proporcionan una evaluación completa de cómo la configuración de los ping slots y las características hardware influyen en la precisión de la sincronización y la eficiencia energética.

Todas las pruebas se realizaron bajo condiciones controladas de temperatura (30–31 °C) y humedad relativa (41–42 %) para minimizar la deriva del High-Speed Internal oscillator (oscilador HSI) debido a factores ambientales, asegurando que los errores observados dependieran principalmente de la densidad de ping slots.

A. Análisis del Error de Sincronización de Ping Slots

Se evaluó el error de sincronización para distintas configuraciones de ping slots mediante diez ejecuciones experimentales, recopilando marcas de tiempo durante diez intervalos de beacon consecutivos. PS64

genera 2 ping slots por beacon, PS32 produce 4, PS16 genera 8 y PS8 produce 16 slots.

La Figura 2 muestra la distribución de errores mediante diagramas de violín. Se distinguen dos componentes del error: el sesgo sistemático, derivado de la deriva del reloj, y el jitter aleatorio, que se mantiene en el orden de unos pocos microsegundos. Debido a esto, configuraciones con mayor densidad de ping slots (PS8) presentan una mayor variabilidad relativa del error, mientras que configuraciones con menor densidad de ping slots (PS64) muestran un coeficiente de variación más reducido.

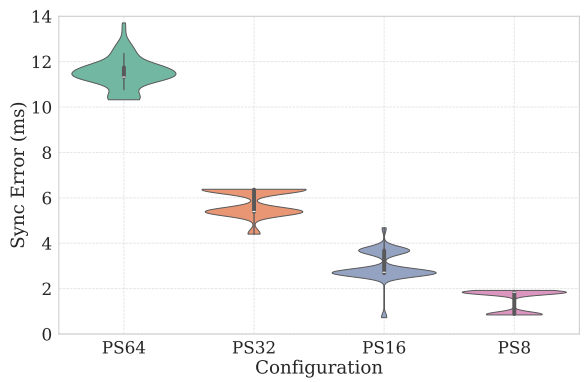


Fig. 2: Distribución del error de sincronización para cada configuración de ping slots. Las mayores densidades muestran más variabilidad temporal, mientras que densidades más bajas producen sincronización más consistente.

Los resultados muestran que el error absoluto aumenta con el período del ping slot, mientras que el error relativo permanece por debajo del 0.02 %. La deriva lineal del oscilador HSI (∼190 ppm) domina el error total, mientras que el jitter aleatorio es menor a 10 μs. Por tanto, el error total puede modelarse como:

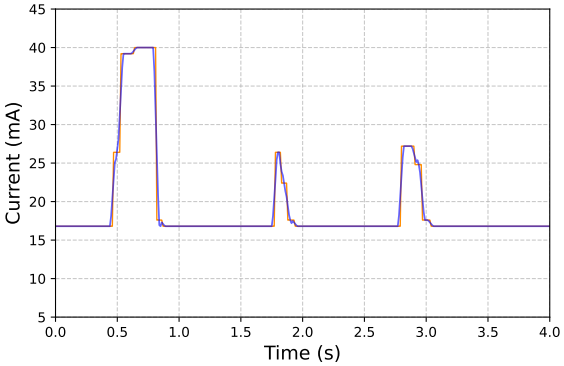
$$E_{\text{total}}(t) = R_{\text{deriva}} \cdot t + \epsilon_{\text{jitter}} \tag{5}$$

El comportamiento observado confirma que la densidad de ping slots influye en la variabilidad absoluta del error, mientras que la deriva sistemática del oscilador determina la precisión global.

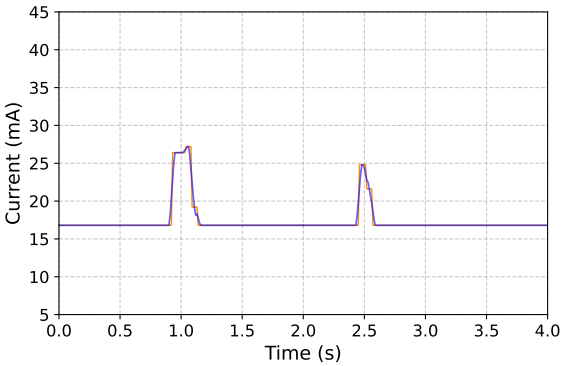
B. Estimación del Consumo Medio de Corriente

Se midió el consumo con XIAO PowerBread [23], tomando muestras cada 10 ms. Las Figuras 3 y 4 muestran los perfiles típicos para modos siempre activo y reposo. El consumo total incluye MCU, módulo LoRa, RTC, reguladores y SPI.

La corriente media total por configuración de ping slots se resume en la Tabla III. La actividad de los ping slots domina el consumo variable; configuraciones con mayor número de slots (PS8) consumen más, mientras que PS64 ofrece un consumo más bajo (16.92 mA siempre activo, 6.45 mA en reposo) manteniendo errores temporales aceptables. La estimación de duración de batería para 1000 mAh indica 2.5 días en Always-on y 6.5 días en modo reposo, mostrando el claro trade-off entre precisión de sincronización y eficiencia energética.

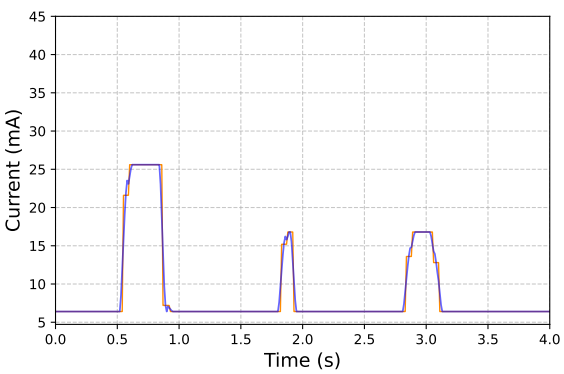


(a) Uplink, RX1 y RX2

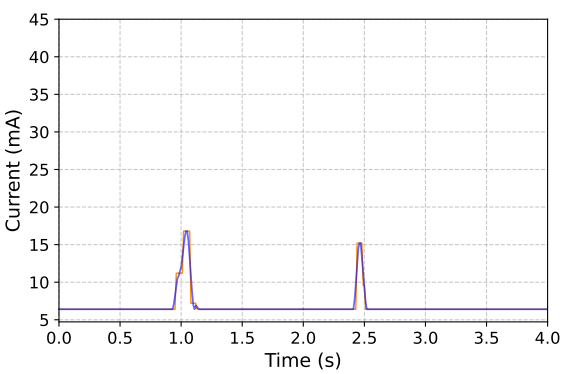


(b) Beacon y ping slot

Fig. 3: Perfiles de consumo Clase B modo siempre activo.



(a) Uplink, RX1 y RX2



(b) Beacon y ping slot

Fig. 4: Perfiles de consumo Clase B modo reposo (sleep).

Tabla III: Corriente media total por configuración de ping slots.

Conf.	Siempre activo (mA)	Modo reposo (mA)
PS8	17.20	6.54
PS16	17.04	6.49
PS32	16.96	6.46
PS64	16.92	6.45

Tabla II: Resumen estadístico del error de sincronización por configuración de ping slots.

Conf.	Muestras (n)	Error Medio (ms)	Desv. Est. (ms)	Coef. Var. (%)	Deriva estimada (ppm)
PS8	1500	1.50	0.0061	31.10	194.8
PS16	700	2.99	0.0043	21.88	194.4
PS32	300	5.81	0.0019	10.09	189.0
PS64	100	11.49	0.0011	5.87	186.9

C. Experimento de Agregación Temporal

Se evaluó cómo se acumulan los errores dentro de un período de beacon usando cuatro EDs B-L072Z-LRWAN1. Los dispositivos pertenecen a dos lotes de fabricación: ED1 y ED2 (2018, inspection SS8516005) y ED3 y ED4 (2017, inspection SS7N25009). Caracterizaciones previas mostraron diferencias significativas en la deriva de los relojes entre lotes [14], lo que proporciona contexto para interpretar los patrones de drift temporal observados.

Para cuantificar la acumulación de errores, se definió el error temporal acumulado intra-beacon $\Delta_{\text{agg}}^{(k)}$ en el k-ésimo ping slot como:

$$\Delta_{\text{agg}}^{(k)} = \sum_{i=1}^{k} (T_{\text{slot}} - (t_{i+1} - t_i)) \tag{6}$$

donde $T_{\text{slot}} = 122{,}880/N$ s es la duración ideal de un ping slot, N el número de ping slots por beacon y $(t_{i+1} - t_i)$ el intervalo medido entre slots consecutivos. Esta métrica permite evaluar cómo se propaga la deriva de sincronización a lo largo de cada período de beacon.

La Figura 5 muestra los errores acumulados intra-beacon (Δ_{agg}) para dos configuraciones representativas: PS8 (alta densidad) y PS32 (configuración intermedia). Se observa que los dispositivos del mismo lote (ED1 y ED2) presentan patrones de deriva casi idénticos, mientras que los del lote más antiguo (ED3 y ED4) muestran una deriva ligeramente menor, sugiriendo que factores específicos del lote — principalmente la calibración del oscilador— influyen más que la edad del dispositivo.

En PS8, la deriva acumulada máxima alcanzó 23.6 ms, lo que representa el 78.7 % de la ventana de 30 ms, operando cerca del límite temporal del ping slot. En contraste, PS32 mostró un crecimiento de error más moderado y mayor margen de seguridad dentro de la ventana de recepción. Las configuraciones PS16 y PS64 siguieron el mismo comportamiento lineal esperado, con márgenes progresivamente mayores a medida que aumenta el período entre ping slots. Esto confirma que los errores se reinician en cada beacon y que la deriva sistemática del oscilador domina la acumulación intra-beacon.

Aunque el comportamiento cíclico evita una acumulación indefinida, la magnitud de la deriva dentro de un único período es crítica para la fiabilidad de los downlinks. Comparando los desfases acumulados observados con la ventana nominal de recepción definida por la especificación LoRaWAN [1] (30 ms por ping slot), se confirma que los márgenes son suficientes para asegurar la operación dentro de los límites de tolerancia.

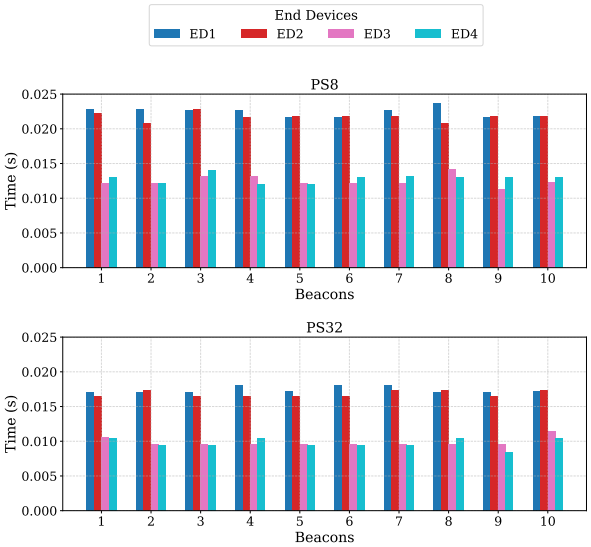


Fig. 5: Errores temporales acumulados intra-beacon para las configuraciones PS8 (arriba) y PS32 (abajo), mostrando los cuatro EDs.

Estos hallazgos subrayan la necesidad de hardware con estabilidad superior o mejoras a nivel de protocolo para despliegues críticos o integración con NTN, donde los retardos dinámicos y desplazamientos Doppler reducen los márgenes de sincronización.

VI. Conclusiones y Trabajos Futuros

Este estudio presenta una caracterización experimental integral de la operación de Clase B en LoRaWAN, evaluando la precisión de sincronización y el consumo energético de dispositivos STM32 B-L072Z-LRWAN1. Los hallazgos principales son los siguientes:

- **Dominio de la deriva de reloj:** El error de sincronización está principalmente determinado por la deriva lineal del oscilador HSI (∼190 ppm), mientras que el jitter aleatorio contribuye menos de 10 μs. El error relativo se mantiene por debajo del 0.02 % en todas las configuraciones de ping slots.
- **Impacto del período de ping slot:** El error absoluto crece con el periodo del ping slot, alcanzando hasta 23,64 ms en PS8, pero siempre dentro de la ventana de recepción nominal de 30 ms, lo que confirma la operatividad segura de la Clase B incluso para configuraciones con mayor número de ventanas de recepción adicionales.
- **Eficiencia energética:** La utilización del modo sleep reduce la corriente de reposo aproximadamente un 62 %, constituyendo el factor más determinante para la duración de la batería. Configuraciones con mayor número de ventanas de

recepción adicionales (PS8) incrementan significativamente el consumo energético.

- **Compromiso precisión-energía:** Existe un equilibrio claro entre densidad de ping slots y eficiencia energética. PS64 ofrece la mejor relación para despliegues de larga duración, mientras que PS8 proporciona mayor capacidad de respuesta temporal, adecuado para aplicaciones críticas en tiempo real si se dispone de osciladores de mayor estabilidad.
- **Variabilidad por lote de fabricación:** La calibración del oscilador y las diferencias entre lotes afectan la precisión de sincronización, subrayando la importancia de la selección de hardware para aplicaciones sensibles a la temporalidad.

Estos resultados proporcionan una línea base sólida para el diseño y optimización de redes IoT híbridas terrestre-NTN, donde la sincronización precisa y la eficiencia energética son cruciales para afrontar los desafíos asociados a la comunicación satelital, como las ventanas de visibilidad limitadas y los desplazamientos Doppler. En este contexto, un paso natural es extender el análisis a escenarios más representativos de entornos satelitales.

Como continuación de este trabajo, se plantea el desarrollo de un emulador de satélite que integre LR-FHSS (Long Range Frequency Hopping Spread Spectrum), una modulación que fragmenta los paquetes y los transmite a través de múltiples canales de frecuencia para mejorar la robustez frente a interferencias, junto con LoRaWAN Clase B. Este entorno permitirá evaluar de forma controlada los mecanismos de sincronización en escenarios NTN bajo condiciones más cercanas a despliegues reales.

Agradecimientos

Este trabajo ha sido financiado por el proyecto PID2024-158682OB-C32, subvencionado por MICIU/AEI/10.13039/501100011033 y por "FEDER Una manera de hacer Europa", mediante la ayuda 2025-INVGO-13370, financiada por el "Programa Investigo" y cofinanciada por el Fondo Social Europeo Plus (FSE+). Asimismo, se reconocen los proyectos 2025-GRIN-38312, 2025-GRIN-38532 y SBPLY/24/180225/000260, financiados por la Universidad de Castilla-La Mancha, la Junta de Comunidades de Castilla-La Mancha y cofinanciados por "FEDER Una manera de hacer Europa".

Referencias

[1] LoRa Alliance, *LoRaWAN Specification v1.0.3*, 2018, Available online: LoRaWAN Specification.

[2] Fred F. Z. Cai, Ash J. S. Shi, Jeff R. S. Zou, C. Q. Jiang, Ray C. C. Cheung, and Alan H. F. Lam, "A LoRaWAN-BLE Based AIoT Smart Farm Management and Control System," in *Proceedings of the 2024 IEEE 10th World Forum on Internet of Things (WF-IoT)*, 2024, pp. 523–529.

[3] Muhammad Asad Ullah, Konstantin Mikhaylov, and Hirley Alves, "An Overview of Direct-to-Satellite IoT: Opportunities and Open Challenges," in *Proceedings of the 2023 IEEE 9th World Forum on Internet of Things (WF-IoT)*, 2023, pp. 1–8.

[4] Dara Ron, Chan-Jae Lee, Kisong Lee, Hyun-Ho Choi, and Jung-Ryun Lee, "Performance Analysis and Optimization of Downlink Transmission in LoRaWAN Class B Mode," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7836–7847, 2020.

[5] Houssem Eddin Elbsir, Mohamed Kassab, Sami Bhiri, and Mohamed Hedi Bedoui, "Evaluation of LoRaWAN Class B Performances and Its Optimization for Better Support of Actuators," *Computer Communications*, vol. 198, pp. 128–139, 2023.

[6] Joseph Finnegan, Stephen Brown, and Ronan Farrell, "Evaluating the Scalability of LoRaWAN Gateways for Class B Communication in ns-3," in *2018 IEEE Conference on Standards for Communications and Networking (CSCN)*, 2018, pp. 1–6.

[7] David Todolí-Ferrandis, Javier Silvestre-Blanes, Víctor Sempere-Payá, and Ana Planes-Martínez, "Analysis of Bidirectional ADR-Enabled Class B LoRaWAN Networks in Industrial Scenarios," *Applied Sciences*, vol. 10, no. 22, pp. 7964, 2020.

[8] Aditya Jagatha, Bhaskar P. Rimal, Varghese Vaidyan, and Yong Wang, "Exploring LoRaWAN Class B and Class C Devices: Performance Analysis and Parameter Optimization Strategies," in *Proceedings of the 2024 IEEE International Conference on Electro Information Technology (eIT)*, 2024, pp. 194–201.

[9] Phui San Cheong, Johan Bergs, Chris Hawinkel, and Jeroen Famaey, "Comparison of LoRaWAN classes and their power consumption," in *2017 IEEE Symposium on Communications and Vehicular Technology (SCVT)*, 2017, pp. 1–6.

[10] Zhicheng Qu, Haotong Cao, Yifan Cheng, Shengchen Wu, and Gengxin Zhang, "A LoRaWAN-Based Network Architecture for LEO Satellite Internet of Things," in *2019 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, 2019, pp. 1–2.

[11] Muhammad Asad Ullah, Konstantin Mikhaylov, and Hirley Alves, "Massive Machine-Type Communication and Satellite Integration for Remote Areas," *IEEE Wireless Communications*, vol. 28, no. 4, pp. 74–80, 2021.

[12] Florian Rolland, Zheng Zhou, Nicola Accettura, and Pascal Berthou, "Achieving reduced latency and energy efficiency in Direct-to-Satellite LoRaWAN communications," in *2025 IEEE 21st International Conference on Factory Communication Systems (WFCS)*, 2025, pp. 1–4.

[13] Wenxiao Tang, Hu Zhao, Shuo Chen, et al., "Measurement of LoRa signal propagation in urban areas utilizing aerial gateway and ground gateway ," *Scientific Data*, vol. 12, pp. 1464, 2025.

[14] Celia Garrido-Hidalgo, Jetmir Haxhibeqiri, Bart Moons, Jeroen Hoebeke, Teresa Olivares, F. Javier Ramirez, and Antonio Fernández-Caballero, "LoRaWAN Scheduling: From Concept to Implementation," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 12919–12933, 2021.

[15] Ideetron, *LORANK8 Gateway*, 2025, Available online: LORANK8 Gateway.

[16] BeagleBoard, *SeeedStudio BeagleBone Green*, 2025, Available online: BeagleBone Green.

[17] WiMOD, *iC880A Datasheet, Version 1.1*, 2019, Available online: iC880A Datasheet.

[18] U-blox, *NEO-6 Series GPS Modules Data Sheet*, 2025, Available online: GPS NEO-6 Datasheet.

[19] ChirpStack, *ChirpStack, open-source LoRaWAN Network Server*, 2025, Available online: ChirpStack website.

[20] Semtech, "Packet Forwarder," 2025, Available online: Packet Forwarder GitHub repository.

[21] Semtech, "LoRa Gateway," 2025, Available online: LoRa Gateway GitHub repository.

[22] AZ-Delivery, *Logic Analyzer Datasheet*, 2025, Available online: AZ-Delivery Datasheet.

[23] Seeed Studio, *XIAO PowerBread - A Breadboard Power Supply with Real-Time Monitoring*, 2025, Available online: GitHub repository for XIAO PowerBread.

[24] STMicroelectronics, *RM0376 Reference Manual*, 2022, Available online: ST RM0376 manual.

[25] Juan A Fraire, Sandra Céspedes, and Nicola Accettura, "Direct-to-satellite iot-a survey of the state of the art and future research perspectives: Backhauling the iot through leo satellites," in *International Conference on Ad-Hoc Networks and Wireless*. Springer, 2019, pp. 241–258.

[26] Murata Manufacturing Co., Ltd., *Sub-G Module Datasheet — CMWX1ZZABZ-091*, 2018, Available online: CMWX1ZZABZ-091 Datasheet.

Calibración Resiliente de Sensores Ambientales de Bajo Coste

Abdellatif Bendjeddou[1] y Ahmed Boubrima[1]

Resumen— **La monitorización de la calidad del aire mediante redes de sensores de bajo coste desempeña un papel fundamental en la regulación medioambiental y en la toma de decisiones públicas. Debido a la disyuntiva inherente entre coste y fiabilidad, estos sensores requieren métodos de calibración adecuados para satisfacer los requisitos de precisión. Sin embargo, muchos métodos de calibración existentes adoptan hipótesis simplificadoras y asumen que las condiciones de funcionamiento son estables y nominales. Como resultado, su precisión puede degradarse significativamente en condiciones adversas, como desastres naturales, cambios meteorológicos bruscos, envejecimiento de los sensores o interferencias deliberadas. Para hacer frente a esta limitación, en este trabajo proponemos un método de calibración resiliente orientado a preservar la calidad de la calibración de los sensores en condiciones anormales. El método aborda conjuntamente las no linealidades inducidas por la temperatura y las perturbaciones transitorias mediante la combinación de un modelo dinámico de calibración con un filtrado adaptativo de señales. Los resultados preliminares muestran mejoras consistentes en el rendimiento de la calibración, tanto en condiciones nominales como anormales, en comparación con las soluciones existentes.**

Palabras clave— **Redes de sensores, sensores de contaminación del aire, calibración.**

I. Introducción

La contaminación atmosférica ha alcanzado niveles peligrosos en los últimos años debido a la urbanización masiva y a la industrialización. Esto supone una grave amenaza para el medio ambiente y la salud humana, y por lo tanto requiere una supervisión rigurosa de la calidad del aire. Ciertamente, supervisar la calidad del aire ayuda a los gobiernos a adoptar medidas correctivas, como la regulación de las emisiones industriales y la limitación del tráfico urbano durante los periodos de alta contaminación.

La supervisión de la calidad del aire se ha basado tradicionalmente en estaciones de monitorización reguladas, que proporcionan datos de referencia y sustentan la toma de decisiones críticas en materia ambiental. Sin embargo, sus elevados costes de instalación, operación y mantenimiento, junto con su gran tamaño, limitan su despliegue a gran escala y reducen la resolución espacial de las observaciones. Esta limitación, unida a la rápida innovación en el Internet de las cosas, ha favorecido el desarrollo de sensores compactos y de bajo coste, considerablemente más pequeños y económicos que las estaciones tradicionales. Aunque estos dispositivos presentan una menor precisión que las estaciones de referencia, su bajo coste, reducido tamaño y facilidad de instalación permiten ampliar sustancialmente la cobertura espacial de la monitorización. Como resultado, las redes de sensores inalámbricos (WSN) han surgido como una solución prometedora para la monitorización distribuida de la calidad del aire, al permitir que sensores desplegados en distintos puntos se comuniquen y colaboren en la recopilación de datos medioambientales.

Calibración de sensores: Los sensores ambientales, y más concretamente los sensores de contaminación atmosférica de bajo coste, presentan varias limitaciones intrínsecas. Entre ellas se incluyen la sensibilidad a las condiciones meteorológicas (por ejemplo, la temperatura y la humedad), la sensibilidad cruzada a otros contaminantes y la deriva a largo plazo debida al envejecimiento del sensor. Estos sensores son en efecto muy sensibles y pueden verse afectados por factores ambientales como la temperatura, la humedad, la luz solar, el viento y la presión atmosférica. Para solucionar esto, se utilizan métodos de calibración que convierten la señal eléctrica del sensor (mV) en niveles de concentración significativos (ppb o μg/m^3), teniendo en cuenta las variaciones ambientales.

Limitaciones de los métodos actuales: Los métodos de calibración suelen partir de la base de que las condiciones de funcionamiento son nominales y de que las variaciones ambientales son graduales. En realidad, los sensores suelen estar expuestos a cambios bruscos y a fenómenos extremos, lo que puede provocar desviaciones en los sensores y una menor precisión de la calibración [2]. Factores como el envejecimiento de los sensores, las interferencias deliberadas y los desastres naturales pueden afectar aún más a la fiabilidad de las mediciones. En estas condiciones, el rendimiento de la calibración se deteriora significativamente, ya que rara vez se tienen en cuenta estos escenarios no ideales. Para garantizar que los sensores de bajo coste puedan complementar o sustituir de forma fiable a las estaciones de monitorización tradicionales, la calibración debe ser resistente a estas condiciones adversas.

Contribuciones: En este trabajo se presenta un método de calibración resiliente para mejorar la fiabilidad de las mediciones realizadas por sensores de bajo coste en condiciones adversas. En concreto:

- Se propone un método de calibración resiliente que aborda conjuntamente la deriva de la línea de base, es decir, la referencia del sensor, las no linealidades inducidas por la temperatura y las perturbaciones transitorias mediante la combinación de un modelo global de calibración con un filtrado adaptativo de señales.
- Se diseña una estrategia de calibración dinámica

[1]Inria, INSA Lyon, CITI, UR3720, France.

que se adapta a condiciones de funcionamiento anormales mediante un control de consistencia combinado con mecanismos de respaldo, lo que permite mantener un rendimiento robusto tanto en escenarios nominales como adversos.

Estructura: Este artículo se estructura de la siguiente manera. En primer lugar, se presentan los trabajos relacionados. A continuación, se proporciona el contexto necesario sobre el funcionamiento de los sensores electroquímicos. Posteriormente, se introduce el método de calibración resiliente propuesto y se presentan los primeros resultados preliminares. Por último, se exponen las conclusiones del artículo.

II. Estado del arte

La fuerte influencia de las condiciones meteorológicas en los sensores de contaminación atmosférica hace que la calibración sea un proceso crucial, por lo que ha sido ampliamente estudiada en la literatura.

En los últimos años, tras la democratización de la detección ambiental mediante sensores de bajo coste, se han presentado numerosos modelos correctivos y de calibración. Los métodos incluyen modelos basados en la física, modelos estadísticos (o empíricos) y métodos de aprendizaje automático. En implementaciones a gran escala —como las redes de sensores inalámbricos ambientales (WSN)—, los métodos de calibración *in situ* se han convertido en una alternativa práctica a la calibración en laboratorio. Eso significa calibrar los sensores sin retirarlos del campo. Delaine et al. [2] clasifican los métodos de calibración en función de múltiples criterios, entre los que se incluyen, por ejemplo, la dependencia de instrumentos de referencia (calibración basada en referencias, parcialmente ciega o ciega), la estructura de la red (calibración por pares, basada en grupos o macrocalibración) y la naturaleza de la relación entre las variables. En el caso de la mayoría de los sensores de bajo coste, a menudo se emplea la calibración de la relación entre variables de múltiples tipos para incluir el efecto de factores influyentes como la temperatura y la humedad, y en ocasiones el factor de sensibilidad cruzada de los gases. Este es el caso de trabajos como [3] [4], en los que se presentaron métodos de calibración estadística (*regresión multilineal* y *splines B cúbicos*).

En trabajos recientes, los métodos de calibración basados en el aprendizaje automático han ganado popularidad debido a su capacidad para modelar relaciones no lineales complejas entre las lecturas de los sensores y las mediciones de referencia. Modelos como los árboles de decisión, los bosques aleatorios, la regresión con refuerzo de gradientes, los k vecinos más cercanos y las máquinas de vectores de soporte han demostrado mejoras significativas en la precisión de la calibración, logrando coeficientes de determinación más altos y reducciones sustanciales del error en comparación con los métodos de regresión tradicionales [6] [5]. También se han investigado las redes neuronales profundas (DNN) para la calibración [7] [8]. Su ventaja radica en su capacidad para captar relaciones más complejas. Sin embargo, los modelos de aprendizaje automático requieren grandes conjuntos de datos etiquetados, especialmente en el caso de las DNN, y pueden tener dificultades para generalizarse en diferentes condiciones ambientales.

Los trabajos existentes parten de la base de que los sensores funcionan en condiciones normales y de que las variaciones ambientales son naturales. Sin embargo, en la práctica, los sensores suelen verse expuestos a fenómenos extremos o a cambios ambientales bruscos, lo que provoca tanto la deriva del sensor como la del calibrador.

Trabajos recientes han comenzado a abordar este reto; [8] consideró condiciones extremas (es decir, desastres naturales), en las que presentó un algoritmo inteligente de selección de calibración para detectar lo que denominan la desviación del calibrador. Estos trabajos alternan dinámicamente entre una calibración de caja blanca fiable (regresión lineal) y una calibración de caja negra más precisa que utiliza redes neuronales profundas en condiciones normales. Sin embargo, en algunos casos podría producirse una interferencia deliberada. Los ataques a la capa física mediante inyección remota de señales contra los sensores constituyen una clase emergente de amenazas; por ejemplo, [9] demostró que los sensores de gas electroquímicos de bajo coste son susceptibles a interferencias dirigidas.

Por lo tanto, existe la necesidad de un método de calibración resiliente para sensores de contaminación atmosférica de bajo coste que sea adecuado para todas las condiciones de funcionamiento, ya sean naturales o deliberadas.

III. Contexto sobre los sensores de calidad del aire

En este trabajo nos centramos principalmente en los sensores electroquímicos de gases. Estos sensores se utilizan para el control de la calidad del aire y funcionan midiendo la concentración de los gases contaminantes de interés, como el NO_2 y el CO_2. No obstante, los métodos presentados en este trabajo siguen siendo aplicables a otros sensores basados en tecnologías diferentes, como la dispersión de la luz en el caso de los sensores de partículas en suspensión (PM) o la detección por fotoionización en el caso de los sensores de compuestos orgánicos volátiles.

A. Sensores electroquímicos

Los sensores electroquímicos de gases funcionan mediante reacciones *redox* específicas que generan una corriente proporcional a la concentración del gas de interés. Cuando el gas de interés llega al electrodo de trabajo (WE o *Working Electrode*), se reduce (o se oxida, dependiendo del tipo de sensor específico para cada gas), mientras que en el contraelectrodo (CE) se produce una reacción complementaria de oxidación (o reducción), lo que forma un circuito cerrado y genera una corriente medible.

La figura 1 muestra el diseño típico de un sensor electroquímico. Utiliza un sistema de cuatro electro-

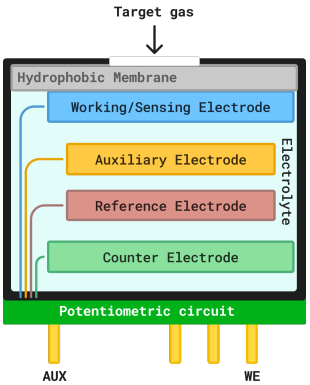


Fig. 1: Gráfico de un sensor electroquímico

dos en el que el potenciostato mantiene un potencial fijo entre el electrodo de trabajo (WE) y el electrodo de referencia (RE) para garantizar reacciones selectivas en el WE, mientras que el RE proporciona una referencia estable y sin corriente y el contraelectrodo (CE) equilibra el sistema suministrando la corriente necesaria para la reacción. La difusión de gas se controla mediante una membrana y los iones se desplazan a través del electrolito para completar el circuito.

Estos sensores usan también un electrodo auxiliar (AE o *Auxiliary Electrode*) de forma similar al WE, pero protegido del gas objetivo, de modo que no reacciona; en su lugar, captura las corrientes de fondo causadas por factores ambientales y la deriva, que pueden restarse más adelante de la señal del WE para mejorar la precisión global del sensor.

B. Influencia meteorológica

Los sensores electroquímicos de gases se ven muy influidos por las condiciones meteorológicas ambientales, principalmente la temperatura T y la humedad relativa RH, que afectan tanto a su respuesta en estado estacionario como a su comportamiento transitorio. La corriente medida $i(t)$ puede expresarse como

$$i(t) = s(T, RH)\, c(t) + i_0(T, RH) + \varepsilon(t), \qquad (1)$$

donde $c(t)$ es la concentración del gas objetivo, $s(\cdot)$ denota la sensibilidad, $i_0(\cdot)$ es la corriente de referencia y $\varepsilon(t)$ representa el ruido.

B.1 Deriva de la línea de base

Las variaciones lentas en T y Rh provocan cambios graduales tanto en s como en i_0, principalmente a través de su influencia en la cinética de reacción y los procesos de difusión. Esto da lugar a una línea de base (i.e. referencia o *baseline*) que varía lentamente, lo que afecta a la precisión de las mediciones a largo plazo y requiere una compensación mediante calibración. La relación entre la temperatura y la sensibilidad es intrínsecamente no lineal, especialmente a temperaturas elevadas, donde las desviaciones se hacen más pronunciadas [10]. Aunque algunos métodos aproximan esta dependencia utilizando modelos lineales, las estrategias de calibración más precisas y robustas se basan en modelos lineales por tramos o no lineales para captar estos efectos con mayor fidelidad.

B.2 Efectos transitorios

Los cambios rápidos en las condiciones ambientales (por ejemplo, fluctuaciones de temperatura, humedad o presión debidas al viento) producen perturbaciones transitorias en la señal del sensor, que suelen observarse como picos breves [10]. Estos transitorios pueden aproximarse mediante una respuesta exponencial por tramos:

$$i_{\mathrm{tr}}(t) = \begin{cases} A\left(1 - e^{-(t-t_0)/\tau_r}\right), & t_0 \le t < t_p, \\ A\, e^{-(t-t_p)/\tau_d}, & t \ge t_p, \end{cases} \qquad (2)$$

donde A es la amplitud máxima, τ_r y τ_d son las constantes de tiempo de subida y caída, t_0 es el tiempo de inicio y t_p denota el tiempo en el que se alcanza el pico. La figura 2 ilustra un pico transitorio positivo representativo observado en la salida de un sensor Alphasense NO_2-B43F [11] en mV (proporcional a la corriente de salida), causado por una rápida variación de temperatura en un entorno no contaminado. El evento tiene una duración inferior a 5 min y presenta una amplitud máxima de aproximadamente +5 mV con respecto a la línea de base, lo que corresponde a unos 15 ppb dependiendo de la sensibilidad del sensor. Tal y como describe el modelo en (2), el transitorio se caracteriza por un rápido aumento exponencial seguido de un decaimiento exponencial más lento. La magnitud de la respuesta está directamente relacionada con la intensidad de la perturbación ambiental, de modo que los cambios más grandes y rápidos dan lugar a amplitudes máximas más elevadas.

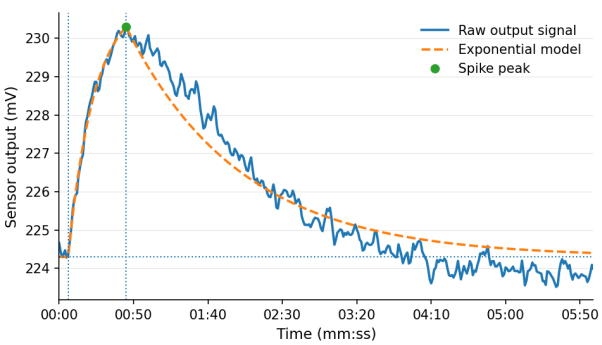


Fig. 2: Ejemplo de un pico transitorio provocado por un cambio brusco de temperatura (Alphasense NO2-B43F)

B.3 Implicaciones para la calibración

Es necesario distinguir entre estos dos efectos: la deriva de la línea de base afecta a la precisión a largo plazo, mientras que las reacciones transitorias introducen desviaciones a corto plazo que pueden dar lugar a lecturas o alertas erróneas.

Los métodos de calibración de la literatura abordan los efectos meteorológicos graduales, pero los transitorios a corto plazo se pasan por alto en gran medida. Además, cuando se mencionan, suelen descartarse como ruido en lugar de modelarse y corregirse de forma activa. Al contrario, en este trabajo,

proponemos una calibración resiliente que tiene en cuenta todos los efectos, incluidos los resultantes de cambios transitorios o incluso de interferencias deliberadas.

IV. Método de calibración resiliente

Proponemos un método de calibración resiliente que aborda de forma conjunta la deriva de la línea de base a largo plazo, las no linealidades inducidas por la temperatura y las perturbaciones transitorias en los sensores electroquímicos de gas. El método combina un modelo de calibración global que aprovecha tanto las señales del sensor (WE, AUX) como una entrada de temperatura, con un método de filtrado diseñado para corregir los efectos transitorios. En condiciones normales de funcionamiento, la calibración mediante un sensor de temperatura externo proporciona mayor precisión y estabilidad en comparación con basarse únicamente en la información derivada de AUX, lo que permite una compensación más precisa de las influencias ambientales. Sin embargo, en presencia de interferencias o transitorios rápidos, la señal AUX se convierte en un indicador más fiable, ya que refleja directamente las perturbaciones que afectan a la célula electroquímica.

Para garantizar la robustez, el método incorpora un mecanismo de supervisión de la consistencia que adapta dinámicamente la estrategia de calibración. En condiciones anómalas, recurre a una calibración basada en AUX e infiere la temperatura a partir de la señal AUX utilizando una relación aprendida (con un R^2 observado de 0,91 en los datos de implementación), lo que permite continuar el funcionamiento incluso cuando el sensor de temperatura externo no es fiable. El flujo de trabajo general se ilustra en el diagrama esquemático de la Fig. 3

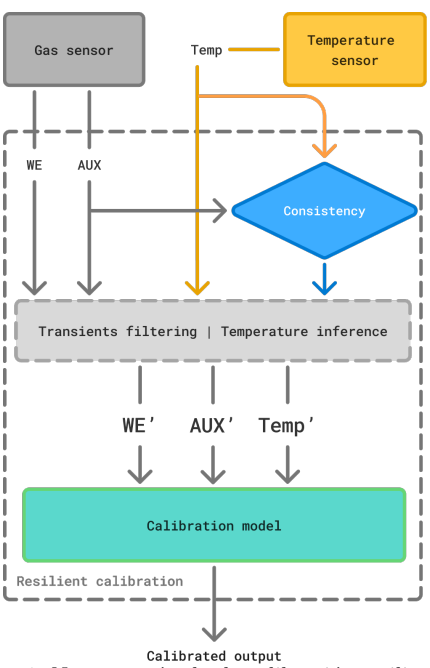


Fig. 3: Nuestro método de calibración resiliente

A. Modelo de calibración global

La temperatura, que está estrechamente relacionada con la humedad relativa, ejerce una influencia significativa en la señal de salida del sensor (a través de los dos efectos descritos en la sección anterior). Para corregir el desplazamiento de la línea de base, fabricantes como Alphasense incorporan un electrodo auxiliar (AUX) en sensores como el NO_2-B043F. Este electrodo tiene por objeto tener en cuenta las señales de fondo no relacionadas con la contaminación.

Sin embargo, en la práctica, la mayoría de las instalaciones se basan en un sensor de temperatura externo para la compensación, en lugar de depender únicamente del electrodo AUX. Esta preferencia se debe a varios factores, entre ellos la menor precisión del AUX en condiciones reales [12], así como la respuesta no lineal del sensor a las variaciones de temperatura [10]. El AUX, sin embargo, puede reflejar mejor las interferencias localizadas y la deriva a largo plazo del sensor.

Para aprovechar las ventajas de ambos métodos, los combinamos: una corrección de la línea de base sencilla basada en el AUX (escalada por un factor para tener en cuenta las diferencias geométricas entre el electrodo de trabajo (WE) y el AUX), seguida de una regresión por tramos para modelar los efectos no lineales de la temperatura:

$$WE_{\text{calib}} = f(WE - \alpha \cdot AUX, T) \tag{3}$$

Donde:

- f es una función de regresión por tramos,
- WE es la señal del electrodo de trabajo,
- α es un factor de escala,
- AUX es la señal del electrodo auxiliar,
- T es la temperatura ambiente exterior.

Este método proporciona una corrección dinámica de la línea de base basada en el electrodo AUX, en lugar de basarse en el valor de línea de base del electrodo de trabajo (WE) facilitado por el fabricante, que puede ser sensible a la deriva a largo plazo y a los efectos del envejecimiento. Además, el uso de una calibración externa basada en la temperatura, implementada mediante un modelo de regresión entrenado con señales corregidas por AUX, permite compensar con precisión las influencias de la temperatura y captar la respuesta no lineal del sensor. La humedad relativa está fuertemente correlacionada con la temperatura, y su inclusión como variable explicativa adicional en la regresión proporciona una mejora insignificante. Por lo tanto, se excluye del modelo.

B. Filtrado de efectos transitorios

Los efectos transitorios se manifiestan como picos bruscos en la señal de salida y no pueden compensarse eficazmente utilizando únicamente un sensor de temperatura externo. Estos picos pueden deberse no solo a cambios de temperatura, sino también a variaciones en la presión atmosférica o la humedad, que el sensor de temperatura no detecta. Además, el uso

de un sensor de compensación externo supone que las condiciones ambientales son uniformes en todo el espacio entre el sensor y la célula electroquímica. En consecuencia, las perturbaciones localizadas y los efectos transitorios quedan sin tener en cuenta.

En este contexto, el electrodo AUX resulta especialmente valioso, ya que capta estos efectos transitorios. Un método de corrección sencillo consiste en restar la señal del AUX de la señal del WE, aprovechando sus respuestas diferenciales. Sin embargo, según nuestros experimentos, esta simple resta solo permite recuperar alrededor del 60 % de la señal original, debido a las inconsistencias entre los electrodos y sus respuestas, especialmente durante los eventos transitorios.

Para resolver esta limitación, proponemos un método en el dominio de la frecuencia que aísla los componentes inducidos por transitorios en la señal AUX y los elimina en la salida WE.

Para mostrar las diferentes respuestas de las señales WE y AUX durante los eventos transitorios, la figura 4 presenta un segmento de señal representativo bajo una perturbación térmica localizada, junto con su descomposición espectral. La señal AUX se mantiene relativamente estable, ya que es sensible principalmente a las variaciones de temperatura, mientras que la señal WE contiene además el componente relacionado con la contaminación, incluso durante la perturbación. El análisis espectral muestra que la señal WE presenta, en general, componentes de mayor magnitud, excepto en el rango de altas frecuencias, donde su espectro se vuelve comparable al de la señal AUX. Este comportamiento puede atribuirse a perturbaciones inducidas por transitorios térmicos, que generan cambios abruptos e introducen componentes dominantes de alta frecuencia. Estos componentes son claramente visibles en la señal AUX y su espectro, y están igualmente presentes en la señal WE. Al aprovechar esta similitud espectral en las altas frecuencias, es posible aislar los componentes inducidos por transitorios que comparten ambas señales.

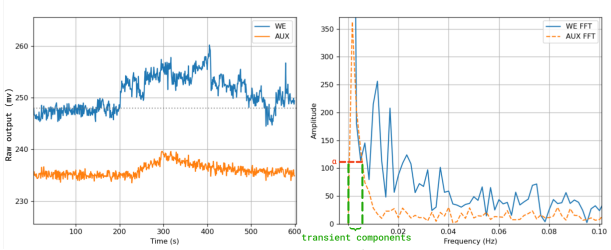


Fig. 4: a: Segmentos de las señales AUX y WE, incluidos los efectos transitorios; b: Espectro de frecuencias de las señales

El algoritmo 1 describe el método propuesto de *filtrado adaptativo basado en la frecuencia (AFBF)*, que corrige la señal WE utilizando la señal AUX como referencia. El algoritmo comienza calculando la FFT de los segmentos de señal tanto de WE como de AUX.

El parámetro $\alpha \in [0, 1]$ define la fracción de la amplitud máxima del espectro de AUX que se va a conservar. Los componentes de frecuencia que superan $\alpha \cdot \text{máx}(|\text{AUX}_f|)$ se identifican como relacionados con transitorios y se seleccionan para su supresión. Este criterio se basa en el hecho de que las perturbaciones inducidas por transitorios se manifiestan como componentes espectrales abruptos y de gran magnitud en la señal AUX, que refleja principalmente efectos ambientales (por ejemplo, la temperatura) y es en gran medida insensible a la contaminación. Por el contrario, las variaciones ambientales normales tienden a ser más suaves y a concentrarse en frecuencias más bajas.

A continuación, los componentes de frecuencia identificados se eliminan del espectro WE, y la señal WE corregida se reconstruye utilizando la FFT inversa. De esta manera, el algoritmo atenúa selectivamente las frecuencias dominadas por transitorios, al tiempo que conserva la señal subyacente relacionada con la contaminación.

Algorithm 1 Adaptive frequency-based filtering

Require: WE and AUX signals $WE[n], AUX[n]$, threshold fraction $\alpha \in [0, 1]$

Ensure: Corrected WE and AUX segments $WE_{\text{corr}}[n], AUX_{\text{corr}}[n]$

1: Compute FFT of WE and AUX signals

$$WE_f[k] \leftarrow \text{FFT}(\text{WE[n]})$$

$$AUX_f[k] \leftarrow \text{FFT}(\text{AUX[n]})$$

2: Identify attack-dominated frequencies in AUX

$$K \leftarrow \{\, k \mid |AUX_f[k]| > \alpha \cdot \text{máx}(|AUX_f|) \,\}$$

3: Suppress attack components in WE and AUX

$$WE_f[k] \leftarrow 0,\ WE_f[-k] \leftarrow 0 \quad \forall k \in K$$

$$AUX_f[k] \leftarrow 0,\ AUX_f[-k] \leftarrow 0 \quad \forall k \in K$$

4: Reconstruct WE and AUX using inverse FFT

$$WE_{\text{corr}}[n] = \text{IFFT}(\text{WE}_\text{f})[\text{n}],$$

$$AUX_{\text{corr}}[n] = \text{IFFT}(\text{AUX}_\text{f})[\text{n}].$$

C. *Calibración resiliente*

Para lograr robustez frente tanto a la deriva de la línea de base como a las perturbaciones transitorias, utilizamos un modelo de calibración global que emplea la señal AUX para la compensación de la línea de base y los valores del sensor de temperatura para la calibración de los efectos no lineales (el mismo que se describe en 3), con filtrado transitorio adaptativo.

Supervisamos constantemente el comportamiento de AUX y del sensor de temperatura para detectar condiciones anormales y, en tales casos, recurrimos dinámicamente a la calibración solo con AUX. A continuación se detallan las diferentes condiciones.

La calibración solo con AUX funciona deduciendo la temperatura a partir de la señal AUX y utilizando

AFBF en caso de que se detecten transitorios. La calibración del electrodo AUX, aunque menos precisa, es sin embargo más robusta en condiciones anormales.

Supervisión de la coherencia: Para garantizar la solidez en condiciones adversas o anómalas, supervisamos continuamente la coherencia entre las señales auxiliares (AUX) y las de temperatura. Esta coherencia se evalúa mediante un modelo polinómico de segundo orden que relaciona AUX con la temperatura:

$$T_{\text{pred}} = a \cdot AUX^2 + b \cdot AUX + c, \tag{4}$$

donde T_{pred} representa la temperatura prevista a partir de la señal AUX. La temperatura prevista se compara con la temperatura medida T, y si la desviación $|T_{\text{pred}} - T|$ supera un umbral predefinido durante una secuencia de muestras consecutivas, se detecta una condición anómala y las señales se consideran inconsistentes.

Además de este indicador de consistencia, supervisamos la varianza de la señal AUX para caracterizar la dinámica de la perturbación. Esto permite identificar tres regímenes de funcionamiento:

- **Señales coherentes con baja varianza AUX:** Esto corresponde a condiciones benignas o a perturbaciones lentas y correlacionadas que afectan a ambos sensores. En este caso, el sistema funciona en condiciones nominales y se aplica directamente el modelo de calibración global:

$$\hat{C} = f(WE - \alpha \cdot AUX, T). \tag{5}$$

- **Señales consistentes con alta varianza en AUX:** Este escenario indica perturbaciones rápidas y correlacionadas que afectan tanto a la señal AUX como a la de temperatura. Aunque se mantiene la consistencia, las señales presentan variaciones de alta frecuencia. En este caso, se aplica el método de filtrado adaptativo basado en la frecuencia (AFBF) para eliminar los componentes transitorios. A continuación, se deduce la temperatura a partir de la señal AUX filtrada utilizando el modelo polinómico:

$$T_{\text{inferida}} = a \cdot AUX_{\text{corr}}^2 + b \cdot AUX_{\text{corr}} + c, \tag{6}$$

  Se prefiere este método porque, en presencia de perturbaciones correlacionadas rápidas, la señal AUX filtrada proporciona una estimación de la temperatura más estable y dinámicamente coherente que la medición sin procesar del sensor. Por último, la salida calibrada se calcula de la siguiente manera:

$$\hat{C} = f(WE_{\text{corr}} - \alpha \cdot AUX_{\text{corr}}, T_{\text{inferred}}). \tag{7}$$

- **Señales incoherentes:** Este caso indica que el sensor de gas o el sensor de temperatura están afectados por interferencias. Dado que no es posible determinar con fiabilidad qué sensor está afectado, el sistema recurre a una estrategia basada en la señal AUX. Es importante destacar que, en este régimen, la temperatura utilizada para la calibración siempre se deduce de la señal AUX en lugar de obtenerse del sensor, ya que refleja mejor las condiciones en las proximidades de la célula sensora.
  - *Alta varianza de AUX:* Corresponde a perturbaciones transitorias. Se aplica el AFBF tanto a WE como a AUX, y la temperatura se deduce a partir de la señal AUX filtrada antes de aplicar el modelo de calibración global.
  - *Baja varianza de AUX:* Corresponde a perturbaciones lentas o persistentes (por ejemplo, deriva o sesgo del sensor de temperatura). En este caso, la temperatura se deduce directamente de la señal AUX sin filtrado, y se aplica el modelo de calibración global. La compensación de la línea de base se mantiene mediante la respuesta del electrodo diferencial.

El procedimiento completo se resume en el algoritmo 2.

Algorithm 2 Resilient Calibration

Require: Temperature T, Working Electrode WE, Auxiliary AUX, τ (consistency threshold), γ (variance threshold)
Ensure: Resilient calibrated output $\hat{C}$

$S_{AT} \leftarrow \text{Consistency}(AUX, T)$
$\sigma^2 \leftarrow \text{Var}(AUX)$
if $S_{AT} \geq \tau$ and $\sigma^2 \leq \gamma$ **then**
  $\hat{C} \leftarrow f(WE - \alpha \cdot AUX,\ T)$
else
  if $\sigma^2 > \gamma$ **then**
    $(WE_{corr}, AUX_{corr}) \leftarrow \text{AFBF}(WE, AUX)$
  else
    $WE_{corr} \leftarrow WE$
    $AUX_{corr} \leftarrow AUX$
  end if
  $T_{inferred} \leftarrow a \cdot AUX_{corr}^2 + b \cdot AUX_{corr} + c$
  $\hat{C} \leftarrow f(WE_{corr} - \alpha \cdot AUX_{corr},\ T_{inferred})$
end if

V. Resultados preliminares

A. Conjunto de datos

Los datos se recopilaron durante aproximadamente dos semanas utilizando una caja de sensores instalada en el exterior, equipada con un sensor electroquímico Alphasense NO2-B43F (salida en mV) y un sensor DHT22 para medir la temperatura ambiente. Se colocó junto a ella un monitor comercial Cairsens de NO_2 (ppb) para proporcionar una referencia de los niveles de contaminación en el área de los experimentos. Se utilizó una placa Arduino Yun como sistema de control y adquisición de datos.

Durante la primera semana, las condiciones de funcionamiento fueron normales. Sin embargo, en la segunda semana, inyectamos múltiples picos transitorios aleatorios. Esto nos permitirá evaluar la eficiencia en condiciones normales y la resiliencia cuando el

sensor se ve sometido a interferencias o perturbaciones localizadas.

B. Procedimiento de validación

Para validar el método de calibración propuesto, llevamos a cabo dos experimentos:

- **Calibración combinada AUX-temperatura frente a calibración solo por temperatura:** Utilizando los datos de la primera semana (condiciones normales), entrenamos un modelo de calibración lineal por tramos (intervalos de 5 grados C) durante los tres primeros días. A continuación, el modelo se evaluó en los días restantes de la misma semana. Comparamos el rendimiento del modelo al incluir información adicional del AUX frente a la calibración basada únicamente en la temperatura. Consideramos el error absoluto medio (MAE) como métrica.
- **Calibración resiliente con modelización AUX-temperatura:** La primera semana se utilizó para entrenar el modelo de calibración por tramos y para ajustar un polinomio de segundo grado que describiera la relación AUX-temperatura ($R^2 = 91$). A continuación, utilizamos la semana siguiente, que contenía picos transitorios inyectados, para evaluar la mejora que supone el uso de la calibración resiliente frente al uso del modelo de calibración por tramos AUX (sin filtrado transitorio ni inferencia de temperatura). De nuevo, utilizamos el MAE como métrica de evaluación.

Destacamos que el objetivo de este análisis es evaluar la mejora relativa, más que el rendimiento absoluto de la calibración, dado el tamaño limitado del conjunto de datos. Además, el uso del dispositivo Cairsens como referencia (a pesar de sus propias limitaciones de precisión) introduce una mayor incertidumbre en la evaluación.

C. Resultados

La figura 5 presenta los resultados del primer experimento. El panel inferior corresponde a la calibración convencional basada únicamente en la temperatura, mientras que el panel superior muestra el modelo propuesto basado en la temperatura-auxiliar (media móvil de 3 minutos).

La calibración realizada con los datos restantes de la primera semana arrojó un error absoluto medio (MAE) de 7 ppb para el método basado únicamente en la temperatura y de 6,3 ppb tras incluir información adicional del AUX, lo que corresponde a una mejora relativa de aproximadamente el 11 %. Aunque esta mejora pueda parecer modesta, se espera que aumente a medida que se incorporen al modelo datos adicionales y variabilidad de medición.

Los recuadros rojos del gráfico inferior resaltan las regiones en las que la inclusión de variables auxiliares mejora la precisión de la corrección de la línea de base, al captar efectos localizados que no se tienen en cuenta en la calibración basada únicamente en la temperatura.

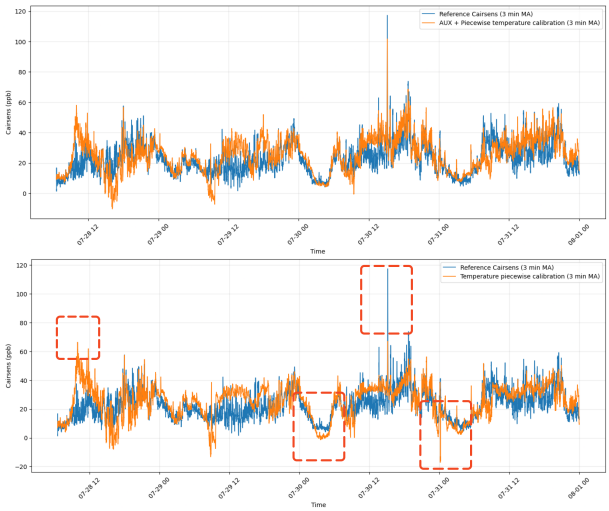


Fig. 5: Comparación de métodos de calibración. **Arriba:** Calibración con temperatura-auxiliar (modelo propuesto). **Abajo:** Calibración solo con temperatura (referencia).

La figura 6 presenta los resultados del segundo experimento. El panel superior muestra la salida calibrada obtenida mediante el método de calibración resiliente propuesto, mientras que el panel inferior corresponde a la calibración de temperatura-auxiliar aplicada sin filtrado de transitorios ni inferencia de temperatura (media móvil de 3 minutos).

Cuando se aplicó a los datos de la segunda semana, que incluyen eventos transitorios inyectados, el método de referencia (sin tratamiento de transitorios) dio como resultado un error medio absoluto (MAE) de 7,1 ppb. La incorporación de la calibración resiliente redujo el error a 6,4 ppb (de forma similar al experimento anterior, en el que no había transitorios presentes en la señal). Observamos de nuevo una mejora relativa, en este caso de alrededor del 10 %.

El panel inferior muestra un patrón irregular causado por picos transitorios, que distorsionan la señal y sesgan las estadísticas agregadas, como la media o la media móvil (algunos ejemplos están indicados con recuadros rojos). Estos artefactos pueden potencialmente desencadenar falsas alertas al imitar eventos de contaminación a corto plazo. Esta es la limitación de los métodos de calibración típicos, ya que son incapaces de calibrar correctamente la señal bajo picos transitorios. Por el contrario, el panel superior muestra una respuesta más suave que sigue más de cerca la señal de referencia. La discrepancia restante se atribuye principalmente a las limitaciones inherentes a los datos, en consonancia con el nivel de error observado anteriormente (6,4 ppb).

VI. Discusión

Este trabajo presenta una validación inicial del método propuesto, y aún existen varias limitaciones. La evaluación se ha llevado a cabo con un conjunto de datos relativamente breve, recopilado durante un periodo de tiempo limitado y en un rango restringido de condiciones ambientales. Aunque se han introducido

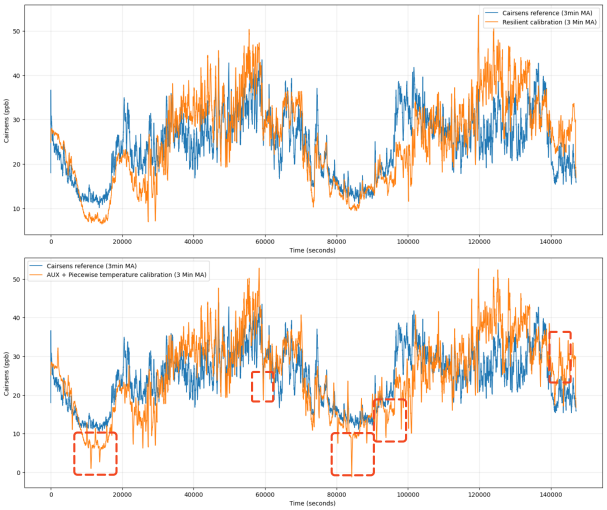


Fig. 6: Comparación de métodos de calibración. **Arriba:** Calibración resiliente con filtrado de transitorios e inferencia de temperatura. **Abajo:** Calibración con temperatura-auxiliar sin tratamiento de transitorios.

eventos transitorios sintéticos, sigue siendo necesaria una evaluación más exhaustiva en condiciones extremas del mundo real. Además, los experimentos se han realizado con un único tipo de sensor, por lo que es necesaria una validación adicional con diferentes tecnologías de detección para evaluar la generalidad del método.

Otra limitación radica en las mediciones de referencia y el alcance de la evaluación comparativa. El uso de un dispositivo Cairsens como referencia, aunque práctico, no proporciona el nivel de rigor de las estaciones de monitorización certificadas, lo que puede afectar a la precisión de la evaluación. Además, la comparación tiene un alcance limitado, y una evaluación más amplia frente a una gama más amplia de métodos de calibración proporcionaría una comprensión más completa del rendimiento del método propuesto. Estos aspectos se abordarán en futuros trabajos mediante implementaciones ampliadas, conjuntos de datos más diversos y comparativas más exhaustivas.

VII. Conclusión

En este artículo se ha presentado un método de calibración resiliente para sensores de contaminación atmosférica de bajo coste, con el objetivo de mejorar su fiabilidad tanto en condiciones normales como en situaciones adversas. Al abordar de forma conjunta la deriva de la línea de referencia, las no linealidades inducidas por la temperatura y las perturbaciones transitorias, e incorporar una estrategia de calibración adaptativa, el método muestra mejoras prometedoras en cuanto a robustez. Aunque los resultados son preliminares, ponen de relieve el potencial de la calibración resiliente para su aplicación en el mundo real.

El trabajo futuro se centrará en una validación más exhaustiva con conjuntos de datos más largos y diversos, múltiples tipos de sensores y comparaciones más amplias con otros métodos de calibración.

Referencias

[1] Y. Kang, L. Aye, T. D. Ngo, and J. Zhou, "Performance evaluation of low-cost air quality sensors: A review," *Science of The Total Environment*, vol. 818, p. 151769, 2022.

[2] F. Delaine, B. Lebental, and H. Rivano, "In situ calibration algorithms for environmental sensor networks: A review," *IEEE Sensors Journal*, vol. 19, no. 15, pp. 5968–5978, 2019.

[3] A. Bisignano, F. Carotenuto, A. Zaldei, and L. Giovannini, "Field calibration of a low-cost sensors network to assess traffic-related air pollution along the Brenner highway," *Atmospheric Environment*, vol. 275, Art. no. 119008, 2022.

[4] A. Boubrima and E. W. Knightly, "Robust environmental sensing using UAVs," *ACM Trans. Internet Things*, vol. 2, no. 4, pp. 1–20, 2021.

[5] M. Taştan, "Machine learning–based calibration and performance evaluation of low-cost Internet of Things air quality sensors," *Sensors*, vol. 25, 2025.

[6] S. Sousan, R. Wu, C. Popoviciu, S. Fresquez, and Y. M. Park, "Advancing low-cost air quality monitor calibration with machine learning methods," *Environmental Pollution*, vol. 374, Art. no. 126191, 2025.

[7] P. C. Liu, T. I. Chou, S. W. Chiu, and K. T. Tang, "A dual-path deep learning model for low-cost air quality sensor calibration," *IEEE Sensors Journal*, vol. 24, no. 23, pp. 39914–39922, 2024.

[8] M. A. Zaidan, N. H. Motlagh, P. L. Fung, A. S. Khalaf, Y. Matsumi, A. Ding, S. Tarkoma, T. Petäjä, M. Kulmala, and T. Hussein, "Intelligent air pollution sensors calibration for extreme events and drifts monitoring," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1366–1379, 2022.

[9] N. Morand, A. Boubrima, W. Bechkit, and Z. Shaikhanov, "HeatPulse: Thermal attacks on air pollution sensors," in *Proc. 3rd Workshop Security Privacy Connected Embedded Systems (SPICES@EWSN)*, Dec. 2024.

[10] Alphasense Ltd., "AAN 110: Environmental Changes: Temperature, Pressure, Humidity (Application Note)," Sensor Technology House, Great Notley, Essex, UK; Doc. AAN 110/SEP22.

[11] Alphasense Ltd., *NO2-B43F / NO2-B43F+ Nitrogen Dioxide Sensor Datasheet*, Version 1.1, 2019. Available: `https://ametekcdn.azureedge.net/mediafiles/project/oneweb/oneweb/alphasense/products/datasheets/alphasense_no2-b43f_datasheet_en_3.pdf`

[12] Cross, E. S., Williams, L. R., Lewis, D. K., Magoon, G. R., Onasch, T. B., Kaminsky, M. L., Worsnop, D. R., and Jayne, J. T. "Use of electrochemical sensors for measurement of air pollution: correcting interference response and validating measurements." *Atmospheric Measurement Techniques.*

Arquitectura basada en aprendizaje federado y comunicación LoRaWAN con componentes DtS-IoT para optimización del tráfico

Víctor Hidalgo, Alejandro Romero, Teresa Olivares,
Carmen Carrión y M. Blanca Caminero[1]

Resumen— **Este trabajo aborda de forma preliminar el reto que supone la gestión del tráfico urbano durante eventos multitudinarios en ciudades medias con recursos limitados, proponiendo una solución basada en la combinación de redes IoT de bajo consumo mediante LoRaWAN y técnicas de aprendizaje federado. Frente a enfoques tradicionales apoyados en comunicaciones móviles y procesamiento centralizado, esta arquitectura permite desplegar redes temporales de bajo coste, adaptarse a cambios dinámicos en la circulación y preservar la privacidad de los datos al mantenerlos en los nodos locales.**

En este contexto se presenta la arquitectura propuesta desarrollada en el proyecto ITEWAN, orientada a la monitorización y optimización inteligente del tráfico en grandes eventos en la ciudad de Albacete. Más concretamente, se definen los componentes de la arquitectura propuesta, basada en el paradigma *cloud-fog-edge*, se exploran las posibles fuentes de datos susceptibles de ser empleadas en los modelos de aprendizaje automático, así como la viabilidad del despliegue de modelos de aprendizaje federado en entornos de bajo coste. De manera complementaria, se revisan las características y funcionalidades de las comunicaciones vía LoRaWAN, y se establecen los requisitos de diseño necesarios para integrar ambos paradigmas en un sistema eficiente y sostenible de gestión del tráfico urbano.

Palabras clave—**IoT, LoRaWAN, aprendizaje federado, computación en la nube, computación en el borde**

I. Introducción

La gestión eficiente del tráfico urbano en eventos multitudinarios supone un reto significativo para los ayuntamientos de ciudades medias con recursos limitados. Las soluciones IoT (Internet of Things) basadas en LPWAN, como LoRaWAN, permiten desplegar redes temporales y de bajo consumo energético. Sin embargo, la existencia de datos sensibles y la necesidad de respuestas descentralizadas requieren enfoques innovadores. Este artículo expone la arquitectura desarrollada en el marco del proyecto "Diseño de un Sistema de Optimización del Tráfico Urbano mediante Aprendizaje Federado y Comunicación LoRaWAN para la gestión de grandes eventos en Albacete" (ITEWAN) para dar respuesta a estos desafíos.

Las soluciones de monitorización urbana suelen apoyarse en redes móviles tradicionales, lo que implica altos costes y complejidad en el procesamiento centralizado [1]. LoRaWAN [2] se ha posicionado como una alternativa eficiente para despliegues masivos de sensores IoT [3]. Por otro lado, el aprendizaje federado ha demostrado su aplicabilidad en escenarios donde la privacidad es crítica, permitiendo el entrenamiento local de modelos y transfiriendo únicamente los parámetros aprendidos [4]. Sin embargo, la integración de ambos paradigmas en la gestión del tráfico urbano apenas ha sido explorada.

El flujo de tráfico durante eventos como ferias, eventos deportivos o conciertos implica cortes y alteraciones temporales en la circulación. Las arquitecturas actuales no se adaptan bien a cambios dinámicos ni preservan adecuadamente la privacidad de los datos recopilados.

LoRaWAN es un estándar LPWAN ampliamente utilizado, diseñado para aplicaciones IoT que requieren comunicación de largo alcance con un consumo mínimo de energía. Una red LoRaWAN típica consta de cuatro componentes principales: dispositivos finales dotados con sensores/actuadores, gateways, un servidor de red y un servidor de aplicación. Este estándar ha sido ampliamente utilizado y mejorado previamente por miembros del grupo de investigación al que pertenecen los autores [5]. Por su parte, el aprendizaje federado (AF) [6] es una técnica de inteligencia artificial que permite entrenar modelos sin necesidad de centralizar los datos, ya que estos permanecen en los dispositivos donde se generan (ej.: en los servidores de aplicación de LoRaWAN). Los autores están abordando el despliegue de entornos AF sobre dispositivos de bajo coste, con el objeto de mejorar su sostenibilidad y eficiencia energética [7].

La unión de Aprendizaje Federado y redes LoRaWAN va a permitir diseñar una arquitectura de monitorización y aprendizaje inteligente de bajo coste para gestionar los aspectos relacionados con el tráfico urbano en la realización de eventos en la ciudad que implican cierre o cambios en el tráfico de distintas calles de Albacete. Para conseguirlo se plantean los siguientes objetivos:

- Proponer una arquitectura integral que incorpore los estándares, componentes y tecnologías necesarios para mejorar la gestión de la movilidad urbana.
- Estudiar cada componente tecnológico por separado para profundizar en su aprendizaje y revisar el estado del arte.
- Plantear como se podrían integrar estos componentes en distintos casos de uso reales.

[1]Dpto. de Sistemas Informáticos, Universidad de Castilla-la Mancha, e-mail: {victor.hidalgo, alejandro.romero, teresa.olivares, carmen.carrion, mariablanca.caminero}@uclm.es.

II. Contexto

Albacete es la principal aglomeración urbana de Castilla-La Mancha, capital provincial y mayor ciudad de la región, con 174 137 habitantes según el padrón municipal a 1 de enero de 2024. Albacete se considera el principal nodo urbano de la Submeseta Sur tras Madrid, y actúa como motor económico y judicial de la comunidad autónoma. La ciudad se localiza en la intersección de varios corredores viarios y ferroviarios de escala nacional. Esta posición de "cruce de caminos" explica tanto su crecimiento demográfico reciente como la centralidad del tráfico y de la movilidad en su funcionamiento urbano.

A. Conectividad viaria interurbana

La red viaria de alta capacidad que converge en Albacete estructura buena parte de la movilidad motorizada que atraviesa o tiene como origen/destino la ciudad:

- Autovía A-31 (Autovía de Alicante): Conecta Madrid con Alicante, atravesando Albacete de noroeste a sureste y constituyendo el principal eje radial hacia el Levante y la Meseta.
- Autovía A-30 (Autovía Albacete–Cartagena): Une Albacete con Murcia y Cartagena, conectando asimismo con la A-3 y la A-7 a través de la red levantina.
- Autovía A-32 (Autovía Linares–Albacete / Autovía Andrés de Vandelvira): Eje transversal **en desarrollo** que enlaza Albacete con la provincia de Jaén. Su tramo de Circunvalación Sur de Albacete forma parte del cinturón de autovías que rodea la ciudad.

Estas conexiones mediante autovías, se complementan con las diversas carreteras nacionales. Destacan las carreteras N-430 (Badajoz–Valencia, con paso próximo a Albacete y convergiendo con la A-31 en sentido Alicante) y la N-322 (futura A-32), única vía que conecta la provincia de Albacete con la de Jaén, razón por la cual es muy transitada.

Finalmente, existe una red autonómica, con itinerarios como la CM-3203 y otras carreteras regionales que estructuran la conexión de la ciudad con su amplio *hinterland* rural y diversas comarcas. Este haz de autovías sitúa a Albacete como nodo de intercambio entre el eje Madrid–Alicante, el corredor mediterráneo y los itinerarios transversales Andalucía–Levante.

B. Principales polos generadores de tráfico

En la ciudad de Albacete existen una serie de zonas que condensan gran parte del tráfico en la ciudad. Se trata de áreas industriales, zonas comerciales y de ocio y áreas de equipación estructural, que se detallan brevemente a continuación.

B.1 Áreas industriales

En la Figura 1 se muestra la situación de las tres principales áreas industriales de la ciudad de Albacete. Estas son:

- Parque Empresarial Campollano: Situado al norte del casco urbano, es la mayor zona industrial de Castilla-La Mancha, con alrededor de 900 empresas y unos 13 000 trabajadores. El acceso a este Parque Empresarial desde la ciudad sufre de retenciones frecuentes, especialmente en las horas punta de entrada y salida de las empresas. Existe transporte público para conectar con la ciudad, con frecuencias limitadas [8].
- Polígono Industrial Romica: A unos 5 km al norte de la ciudad, con más de 500 empresas y unos 5 000 empleos. Se accede principalmente por la A-32 sentido Requena. La ausencia de transporte público hace que el uso de vehículo privado sea mayoritario.
- Parque Aeronáutico y Logístico de Albacete: Situado al sur, junto al aeropuerto y la Base Aérea de Los Llanos, con hubs logísticos como Airbus Helicopters. No existe transporte público aunque sí que hay lanzaderas para los trabajadores del mismo.

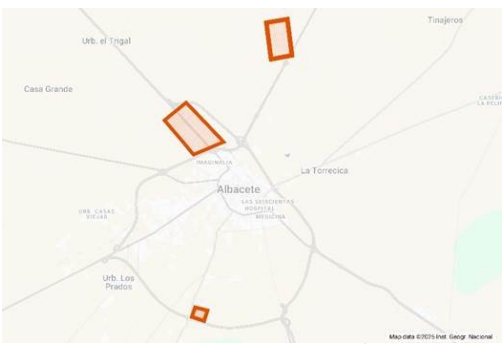


Fig. 1: Principales áreas industriales de Albacete. De arriba abajo, Pol. Ind. Romica, Pol. Ind. Campollano, Parque Aeronáutico y Logístico.

B.2 Zonas comerciales y de ocio

Existen tres polos generadores/atractores de tráfico en la ciudad, con diferentes usos a lo largo del año:

- Centro Comercial Imaginalia: Situado al noroeste de la ciudad, es el mayor centro comercial de Castilla-La Mancha. Dispone de 57 establecimientos, incluyendo diversos comercios, un hipermercado, salas de cine y varios locales de restauración. Genera importante dependencia del automóvil debido a su carácter periférico, pese a disponer de accesos desde la ronda urbana y línea de autobús.
- Institución Ferial de Albacete (IFAB): El recinto ferial de la IFAB ofrece entre 8 000 y 9 000 m^2 de exposición interior y 54 000 m^2 al aire libre. A lo largo del año acoge diferentes ferias y eventos.
- Recinto Ferial: El principal evento que acoge es la Feria de Albacete, que tiene lugar entre el 7 y el 17 de septiembre, de gran atracción regional y nacional. Se alcanzan picos de más de 400.000 visitantes los días de más afluencia, duplicando sobradamente la población de la ciudad. Por tanto, este evento genera restricciones de tráfi-

co y estacionamiento durante los días de feria. Además, a lo largo del año el recinto se emplea para celebrar diversos eventos de menor afluencia.

B.3 Equipamientos estructurales

Hay otros dos puntos en la ciudad que también influyen en gran medida en los flujos de tráfico, como son el Campus Universitario y el Hospital General de Albacete:

- Campus Universitario (UCLM): Situado al sur de la capital, en el barrio Universidad, con conexión directa con el Parque Científico y Tecnológico, el Jardín Botánico y el Estadio de fútbol Carlos Belmonte.
- Hospital General: Perteneciente al Complejo Hospitalario Universitario de Albacete (CHUA), se trata de uno de los hospitales de referencia en Castilla-La Mancha, atendiendo a más de un millón de personas. Situado en la calle Hermanos Falcó, y actualmente en proceso de ampliación, genera flujos continuos de movilidad y condiciona la gestión del estacionamiento y transporte sanitario.

C. Sistema de circunvalaciones urbanas

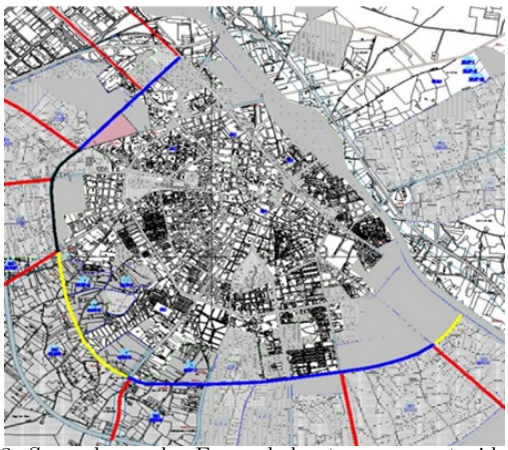
El núcleo urbano de Albacete se rodea de un sistema de rondas de circunvalación, que permiten agilizar el tráfico entre diversas zonas de la ciudad. Esta red se organiza en tres niveles: **primera ronda urbana**, **segunda ronda (AB-20)** y **cinturón exterior de autovías**. El Plan de Movilidad Urbana Sostenible (PMUS) de la ciudad utiliza esta lógica para clasificar la red:

- Primera ronda urbana: Conjunto de avenidas que perimetran parcialmente la ciudad, desde el barrio Industria hasta Medicina. El lado este no está cerrado (zona del Parque Lineal). Dispone de dos carriles por sentido en la mayor parte; donde no existe ronda, las vías se reducen a un carril.



Fig. 2: Primera ronda de circunvalación.

- Segunda ronda (AB-20): Ronda de unos 12 km que conecta la A-31 al oeste con el nudo A-30/A-31 al este. A fecha del documento, un 50 % está construida y recientemente se ha aprobado un nuevo tramo que elevará el porcentaje al 75 % [9].


Fig. 3: Segunda ronda. En azul, los tramos construidos; en amarillo, los tramos sin construir; en negro, los tramos en ejecución. Fuente: Ayuntamiento de Albacete.

- Tercera ronda (cinturón de autovías): Formada por la A-31, A-30 y A-32, que rodean la ciudad al completo y enlazan con todas las carreteras nacionales y autonómicas del entorno.

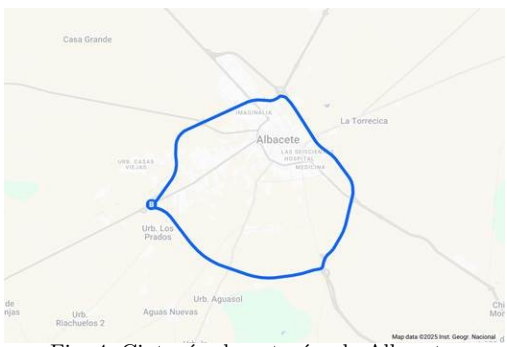


Fig. 4: Cinturón de autovías de Albacete.

D. Casco urbano

Dentro del casco urbano, limitado fundamentalmente por la primera ronda urbana descrita arriba, las calles de la ciudad de Albacete son generalmente de sentido único y un carril. Esto condiciona el flujo de tráfico, sobre todo en momentos puntuales como son las horas de entrada y salida de los colegios, especialmente en días de metereología adversa. Además, varias calles del centro urbano se han peatonalizado durante los últimos años, desviando el tráfico por las calles circundantes. Más aún, el flujo de tráfico durante eventos como ferias o conciertos implica cortes y alteraciones temporales en la circulación.

III. Arquitectura

La Figura 5 ilustra la propuesta del ecosistema ITEWAN. Se trata de una arquitectura que integra dispositivos dotados de sensores IoT, *gateways* LoRaWAN y un sistema de agregación federada que permite entrenar modelos de inteligencia artificial sin centralizar los datos. Los sensores, desplegados en puntos estratégicos, captan información sobre tráfico, ruido y calidad del aire. Por su parte, las capas centrales de computación en el borde y comunicación aprovechan la capacidad de los *gateways* situados en

el borde para realizar preprocesamiento y entrenamiento local. De esta manera, solo se transmiten los parámetros aprendidos hacia el servidor de aplicación en la nube, donde se agregan de forma federada para construir un modelo global. Este diseño, apoyado en LoRaWAN y complementado con enlaces WAN (o incluso satelitales cuando sea necesario), sigue una lógica por capas (*edge–fog–cloud*) que reduce costes de comunicación y computación, mejora la privacidad al evitar el envío de datos en crudo y aporta resiliencia y flexibilidad ante cambios en la topología urbana [10].

De forma transversal a las capas de la arquitectura, el despliegue del aprendizaje federado aporta una implementación de algoritmos de inteligencia artificial que mejora la seguridad y privacidad de los datos recopilados (al no ser necesaria su transmisión hacia servidores externos en la nube). Al tratarse de una aplicación distribuida, se hace necesario también implementar estrategias de orquestación de los componentes involucrados, más concretamente, en lo que respecta a las comunicaciones entre los clientes de aprendizaje federado y su servidor de agregación. Esto se debe a que la disparidad de los recursos hardware desplegados en las capas de adquisición de datos y computación en el borde pueden provocar problemas de sincronismo a la hora de calcular el modelo agregado [11], [12].

En los siguientes apartados se detallan los aspectos más importantes de cada capa de la arquitectura.

A. Adquisición de datos

El sistema puede combinar mediciones obtenidas de los sensores IoT de la infraestructura existente (cámaras en semáforos, paneles informativos y estaciones meteorológicas municipales), enriqueciéndolas con recursos externos, tales como APIs de tráfico y mapas (Google Maps, OpenStreetMap), observación terrestre (Copernicus), predicciones meteorológicas (AEMET) y series históricas (IGN, Google Earth Pro). Se contempla además el uso de indicadores de afluencia derivados de la red móvil (conteo de IMEIs) y reportes ciudadanos. Esta integración daría lugar a un conjunto de datos multimodal (imágenes, audio, texto, geolocalización y medidas numéricas) que permitirá un modelado más completo del sistema de movilidad, garantizando la protección de datos sensibles gracias al paradigma federado.

Más concretamente, es esencial disponer de datos en tiempo real en los puntos de interés críticos identificados en la Sección II (polígonos industriales, zonas comerciales, equipamientos estructurales), así como en la primera ronda de circunvalación, por ser la más cercana al centro urbano y, por tanto la más susceptible de influir en los flujos de tráfico en la ciudad.

Por otro lado, con el objeto de mejorar la sostenibilidad de la arquitectura aprovechando recursos ya disponibles, se propone sacar partido de estaciones de sensorización ya desplegadas en la ciudad [13]. Estas estaciones disponen de alimentación eléctrica (conexión a la red eléctrica complementada por paneles solares en muchos casos) y de conectividad IP, por lo que se plantea integrarlas en la arquitectura propuesta. En primera instancia, se pueden emplear para la **extracción de datos del entorno**, pues recopilan diversos datos medioambientales (nivel de CO_2, ruido, ...) y meteorológicos (temperatura, humedad, dirección del viento...) (ver Figura 6). Además de su panel de visualización, estos registros son accesibles vía API, lo que permite incorporarlos en procesos de analítica más compleja, cruzándolos con otras fuentes de datos, como por ejemplo las previsiones meteorológicas oficiales.

Más allá de la monitorización pasiva, la arquitectura contempla la **instalación de servidores *edge* de bajo coste**. En particular, se puede aprovechar la propia infraestructura física y de red de los paneles actuales para desplegar servidores de bajo consumo que permitan procesar a nivel local los datos recogidos por los sensores de la estación. Estos servidores *edge* podrían incorporar otras fuentes de datos adicionales, como cámaras de vigilancia para conteo de vehículos o personas, facilitando el entrenamiento de los modelos multimodales. De este modo, la convergencia de diversas modalidades sensoriales permite explotar las correlaciones entre el tráfico, el entorno y la actividad ciudadana, logrando una caracterización del entorno urbano precisa y robusta que supera las capacidades de cualquier análisis unimodal convencional.

B. Computación en el borde

Según se ha indicado en el apartado anterior, se plantea la adquisición de datos multimodales combinando distintas fuentes de datos. Algunos de los datos recogidos pueden ser sensibles (por ejemplo, las caras de las personas o las matrículas de los vehículos, en el caso de las cámaras) y por tanto, no es deseable que sean transmitidos en bruto a través de la red hacia servidores en la nube por criterios de seguridad y normativa de privacidad. En este escenario, los nodos desplegados en el borde (*edge*) actúan como una capa crítica que garantiza tanto la escalabilidad del sistema, al optimizar el uso del ancho de banda, como la soberanía de los datos mediante el procesamiento local.

Es en este contexto donde las estrategias de **aprendizaje federado** cobran sentido, al alinearse con los requisitos de privacidad y ahorro en costes de transmisión que requiere el escenario objetivo. Dada la heterogeneidad del hardware en el ecosistema ITEWAN, la selección de modelos debe adaptarse a la capacidad de cómputo de cada nodo. En particular, para dispositivos con memoria limitada, se priorizarán modelos ligeros que minimicen el intercambio de mensajes, como el algoritmo propuesto en [14]. En este enfoque se desplaza el entrenamiento al borde mediante la ejecución local de *k-Nearest Neighbors*(k-NN) y un sistema de votación por mayoría, que garantiza la convergencia del modelo sin saturar el ancho de banda de la red.

En aquellos nodos donde la capacidad de cómpu-

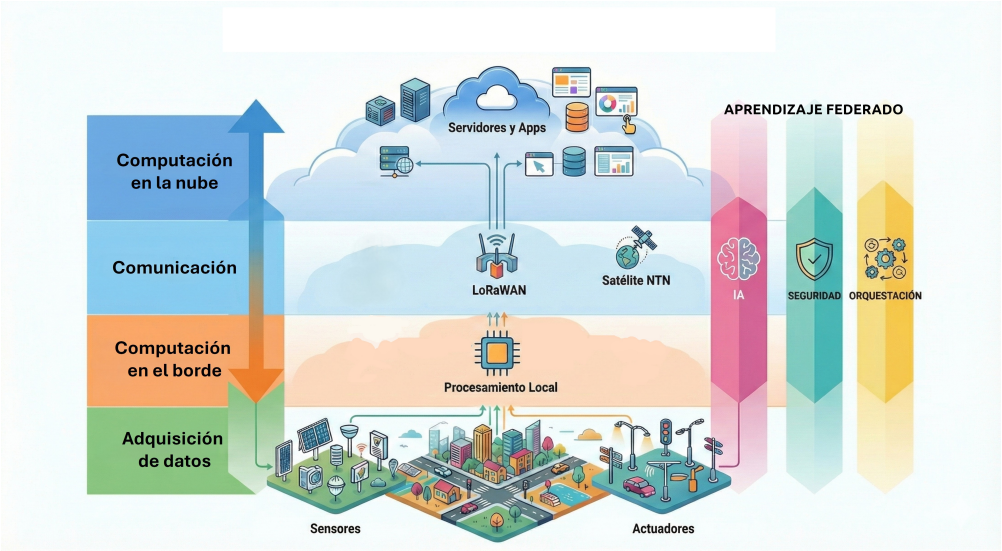


Fig. 5: Arquitectura del ecosistema ITEWAN.

Componente	Rol	Tecnología
Adquisición de datos	Medición tráfico, ruido, contaminación	LoRa
Computación en el borde	Agregación y preprocesamiento, entrenamiento	LoRa, WiFi
Comunicación	Transporte de datos	LoRaWAN
Servidores en la nube	Agregador federado, integración pesos modelos locales	IA (i.e., Python)

Tabla I: Componentes y tecnologías del sistema

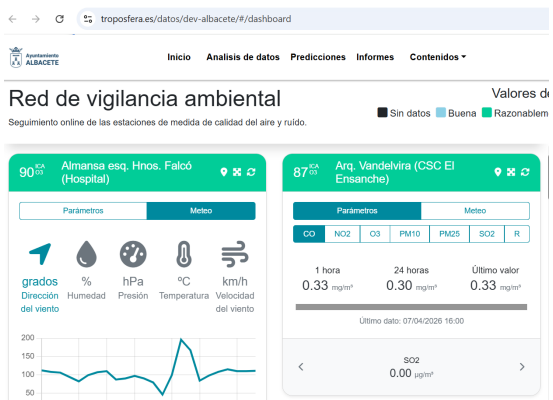


Fig. 6: Ejemplo de datos recogidos por estaciones medioambientales. Fuente: `https://troposfera.es/datos/dev-albacete/#/dashboard`.

to lo permita, se plantea explorar el uso de modelos ligeros basados en YOLOv8 para conteo de vehículos [15]. El objetivo es aprovechar su eficiencia en hardware embebido para obtener métricas de tráfico en tiempo real sin necesidad de transmitir vídeo a la nube. Asimismo, en aquellos servidores con mayor capacidad de cálculo, se pretende investigar el potencial de las técnicas de Fusión de Características Multimodales (MFF) con el fin de correlacionar los datos heterogéneos obtenidos a través de las cámaras, sensores ambientales y fuentes externas para obtener una visión integral y robusta del entorno urbano [16], [17]. Este enfoque se apoya en modelos de éxito existentes en la literatura que demuestran cómo la combinación de flujos visuales y registros ambientales mejora la detección de incidentes urbanos en comparación con análisis unimodales aislados [18].

C. Comunicación

C.1 LoRaWAN

El estándar principal que se utilizará para el envío de datos recogidos por los sensores hasta los servidores ubicados en la nube será LoRaWAN. A medida que la conectividad de IoT crece en todo el mundo, LoRa Alliance [19] está aprovechando la oportunidad para que la tecnología LoRaWAN se convierta en un método de conectividad IoT omnipresente a nivel mundial, de tipo «plug-and-play», integrado en los objetos cotidianos y capaz de aportar un valor cada vez mayor a los usuarios.

LoRaWAN se está convirtiendo rápidamente en una tecnología inalámbrica esencial para un mundo cada vez más inteligente y conectado. Al igual que con las revoluciones tecnológicas anteriores en telefonía móvil, Wi-Fi y Bluetooth, la revolución de la conectividad LoRaWAN está impulsando una nueva generación de dispositivos, usuarios y aplicaciones. LoRaWAN alcanza la máxima accesibilidad al estar construido sobre una base de estándares abiertos, con un ecosistema que ofrece numerosas implementaciones de código abierto que aceleran el tiempo de comercialización. El uso de bandas de frecuencia sin licencia en todo el mundo también significa que, al igual que el Wi-Fi, LoRaWAN puede ser implementado por cualquiera, no solo por grandes empresas con recursos suficientes para permitirse licencias de espectro. LoRaWAN también tiene unos costes de despliegue de infraestructura más bajos que otras tecno-

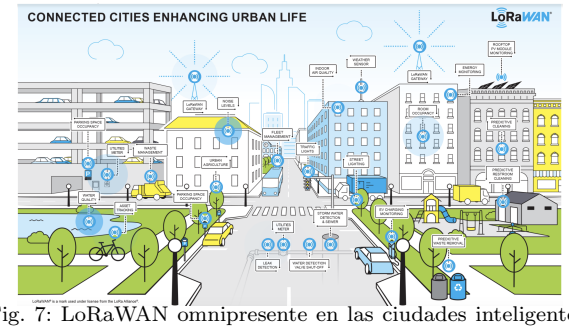


Fig. 7: LoRaWAN omnipresente en las ciudades inteligentes. Fuente: `https://lora-alliance.org`.

logías de conectividad de IoT, con gateways LoRaWAN de un coste comparable al de los puntos de acceso Wi-Fi y servidores de red LoRaWAN tan ligeros y económicos que pueden incluirse en el propio gateway [20]. Como se puede ver en la Figura 7 que nos presenta LoRa Alliance, este estándar formará parte de las *smart cities* con multitud de casos de uso. Algunos ejemplos en la bibliografía que usan LoRaWAN en temas relacionados con el tráfico en las ciudades serían [21] que pretende ser una ayuda en detección de vías libres para vehículos de emergencias en accidentes de tráfico, [22] que plantea el diseño de un sistema de monitorización del problemático ruido del tráfico urbano basado en LoRaWAN. En [23] ya se hace referencia a los sistemas de transporte inteligentes y aparece la previsión del estado del tráfico como factor importante, presentando una arquitectura de comunicaciones para gestion de autopistas basada en LoRaWAN y modelos de aprendizaje profundo. Por otra parte, LoRa Alliance, ha ampliado sus directrices para incluir las velocidades de transmisión de datos de *Long Range - Frequency Hopping Spread Spectrum* (LR-FHSS). Las velocidades LR-FHSS aumentan considerablemente la capacidad de la red y su resistencia a las interferencias, lo que abre la puerta a nuevas oportunidades de implementación en todo el mundo. Como ya se ha comentado, el uso de LoRaWAN para aplicaciones IoT sigue aumentando a un ritmo vertiginoso. La incorporación de la nueva tecnología LR-FHSS al protocolo LoRaWAN garantiza la capacidad de la red de cara al futuro, al tiempo que mantiene sus características de bajo consumo energético [24].

C.2 LoRa

LoRa es una tecnología física patentada para la conectividad de IoT dentro de las redes LPWAN. Se caracteriza por bajas velocidades de transmisión de datos, que suelen oscilar entre 300 bps y 50 Kbps, baja potencia de transmisión y un largo alcance de comunicación, que puede llegar hasta los 10 km en condiciones favorables. LoRa opera utilizando anchos de banda de canal de 125, 250 y 500 kHz en bandas de frecuencia sub-GHz sin licencia y emplea una técnica de modulación patentada denominada «*Chirp Spread Spectrum*» (CSS) desarrollada por Semtech [25]. LoRa soporta seis factores de dispersión (SF), que van desde SF7 hasta SF12. El factor de dispersión define la relación entre la velocidad de símbolos y la velocidad de chips e influye directamente en la velocidad de transmisión de datos, el alcance de la comunicación y el tiempo de emisión. Además de estos parámetros de la capa física, se han propuesto varios mecanismos y arquitecturas de red para mejorar la utilización del canal y la fiabilidad de la comunicación en redes basadas en LoRa. El protocolo de red más ampliamente adoptado construido sobre la capa física LoRa es LoRaWAN, que define la capa de control de acceso al medio (MAC) y suele emplear una topología en estrella de estrellas en la que los dispositivos finales se comunican con pasarelas que retransmiten los datos a un servidor de red central. Esta arquitectura simplifica la gestión de la red y permite implementaciones de IoT a gran escala [26].

Existen distintas publicaciones en la literatura que incorporan LoRa en sus comunicaciones en situaciones posteriores a terremotos o desastres naturales como alternativa fiable a otros estándares por su fiabilidad y características particulares [27], [28]. Otros artículos con un enfoque más teórico presentan modelos basados en redes neuronales y bayesianas para toma decisiones en condiciones de tráfico y parámetros ambientales cambiantes [29].

C.3 Redes no terrestres basadas en satélites

Los sistemas DtS-IoT (*Direct to Satellite IoT*), también conocidos como Internet de las cosas remotas (IoRT), han despertado recientemente gran interés como una solución prometedora de red no terrestre (NTN, *Non Terrestial Network*) debido a sus ventajas en cuanto a cobertura global y conectividad masiva. Además, la conectividad DtS es uno de los componentes adicionales del desarrollo hacia las redes 6G. Puede proporcionar cobertura inalámbrica global para aplicaciones IoT, al tiempo que complementa las redes terrestres. Puede ofrecer conectividad sin interrupciones para aplicaciones en zonas remotas o aplicaciones donde la cobertura terrestre es limitada o puede fallar por distintas causas [30], [31]. Varias empresas como Lacuna Space, Myriota y ORBCOMM ya ofrecen estaciones base y simplifican la conectividad en regiones remotas o de difícil acceso donde es muy complicado el despliegue de infraestructura. Para hacer posibles sistemas DtS-IoT escalables y energéticamente eficientes, se han explorado diversos enfoques, como tecnologías de redes LPWAN. Las empresas de satélites están cubriendo las carencias de conectividad y abordando los retos globales mediante la integración de la conectividad terrestre a través de redes LoRaWAN con las comunicaciones por satélite, con el fin de recopilar datos de sensores remotos, establecer un sistema de comunicación ante desastres naturales o en situaciones de saturación o fallo de las redes terrestres. En entornos de canales de satélites de órbita baja (LEO) Semtech ha propuesto LR-FHSS que ofrece un aumento significativo de la capacidad gracias al salto de frecuencia abordando con éxito tanto los problemas de interferencias como los relacionados con la movilidad, aun-

que todavía queda mucho trabajo por hacer y muchos problemas por resolver [30]. La modulación LoRa y la modulación LR-FHSS pueden coexistir en la misma red LoRaWAN. La incorporación de LR-FHSS también podría ayudar a liberar canales de modulación LoRa ampliando significativamente la capacidad de la red. Lo que suena muy prometedor es el hecho de que LR-FHSS es capaz de admitir millones de dispositivos finales y ofrece un nuevo nivel de fiabilidad para los servicios de IoT [24].

D. Computación en la nube

Los recursos alojados en la nube proporcionarán la capacidad de agregación necesaria para los modelos de aprendizaje federado, a partir de los parámetros enviados por los clientes desplegados en el borde de la red. Si bien esta agregación no será demasiado exigente en recursos (recordemos que el servidor no trabaja sobre los datos en bruto, sino sobre los parámetros del modelo), el hecho de alojarla en la nube proporciona ese punto central necesariamente accesible por todos los clientes.

Además, la nube albergará los *pipelines* de procesamiento de datos que permitirán extraer el valor de los datos recogidos en la capa de adquisición de datos, junto con los interfaces de usuario y paneles de control, accesibles para los usuarios finales. Del mismo modo, parte de los datos procesados servirán para alimentar paneles de información distribuidos por la ciudad, donde se mostrará información de interés para la ciudadanía (rutas alternativas, desvíos, etc.)

E. Aprendizaje Federado

El aprendizaje federado (FL) es un paradigma de aprendizaje automático distribuido idóneo para el escenario propuesto, al permitir a múltiples dispositivos o nodos, denominados *clientes*, entrenar colaborativamente un modelo común sin compartir sus datos locales [4]. En la arquitectura diseñada, este enfoque sustituye la transmisión de los datos en bruto a un servidor central por el envío de actualizaciones paramétricas. Cada cliente entrena un modelo local con sus propios datos y solo envía actualizaciones, como gradientes o parámetros, al servidor central. Este servidor agrega dichas actualizaciones provenientes de todos los clientes participantes para refinar el modelo global, que posteriormente se redistribuye a los clientes para la siguiente ronda de entrenamiento [32], [33].

El proceso global de aprendizaje federado comprende múltiples rondas de entrenamiento colaborativo y, opcionalmente, la evaluación del modelo de forma colaborativa o centralizada. Se basa en dos entidades principales: los participantes que poseen los datos, denominados *clientes*, y el *agregador* o *servidor central*, que coordina todo el proceso.

Dependiendo de la naturaleza y escala de los clientes, el aprendizaje federado puede clasificarse en dos paradigmas principales: *cross-silo* y *cross-device*. En el *FL cross-silo*, el número de clientes es reducido y estos suelen ser entidades fiables y con recursos relativamente abundantes (por ejemplo, organizaciones o centros de datos), lo que permite una participación más estable y sincronizada. Por el contrario, el *FL cross-device* involucra un gran número de dispositivos, típicamente IoT o dispositivos personales, caracterizados por su alta heterogeneidad, disponibilidad intermitente y recursos limitados. Este último escenario es especialmente representativo de entornos Edge/Cloud, donde los clientes operan en la capa de borde mientras que el servidor de agregación se ejecuta en nodos de niebla o en la nube.

La distinción entre estos dos paradigmas resulta clave, ya que condiciona directamente el diseño de los algoritmos de entrenamiento y agregación. En particular, los entornos *cross-device* presentan una elevada heterogeneidad en términos de capacidad de cómputo, almacenamiento, ancho de banda, disponibilidad y recursos energéticos, lo que plantea desafíos importantes en el proceso de entrenamiento [34].

E.1 Estrategia de sincronización

El enfoque predominante en aprendizaje federado, especialmente en escenarios *cross-silo*, es el modelo síncrono, en el que el servidor espera a que una fracción predefinida de clientes complete su entrenamiento local antes de realizar la agregación. El algoritmo más representativo es FedAvg, que combina las actualizaciones de los clientes mediante una media ponderada. Este esquema garantiza una agregación consistente, ya que todas las actualizaciones se realizan sobre versiones relativamente alineadas del modelo global, lo que favorece la estabilidad del entrenamiento y la convergencia. Además, permite un control explícito de la participación de los clientes en cada ronda, lo que facilita la implementación de políticas de selección y muestreo.

No obstante, la aplicación directa del aprendizaje federado síncrono presenta limitaciones importantes en escenarios *cross-device*, donde la heterogeneidad entre los clientes es más marcada. En particular, aparece el denominado *efecto straggler*, en el que clientes más lentos o con recursos limitados retrasan la finalización de la ronda de entrenamiento, lo que afecta tanto a la eficiencia del entrenamiento como a la convergencia del modelo [35], [36]. Como consecuencia, los clientes más rápidos permanecen inactivos mientras esperan al resto, lo que reduce la eficiencia global del sistema y aumenta el tiempo de convergencia.

Para mitigar estas limitaciones, la arquitectura propuesta pretende explorar alternativas al esquema estrictamente síncrono. Por un lado, los enfoques asíncronos eliminan la necesidad de esperar a todos los clientes, permitiendo actualizaciones continuas del modelo global, aunque introducen problemas de obsolescencia en las actualizaciones (*staleness*) debido al uso de gradientes desactualizados. Por otro lado, los métodos semi-asíncronos buscan un equilibrio entre ambos extremos, reduciendo el impacto de los *stragglers*, sin incurrir en un sesgo excesivo hacia los clientes más rápidos.

Esta flexibilidad en la sincronización es fundamental en nuestro entorno para integrar con éxito la Fusión de Características Multimodales (MFF), permitiendo que las actualizaciones de sensores estadísticos simples y los resultados de la analítica de vídeo converjan de manera eficiente en un modelo de inteligencia urbana robusto.

IV. Futuros casos de uso

La arquitectura propuesta no solo permite la monitorización distribuida del tráfico urbano, sino también la activación de mecanismos de respuesta en tiempo casi real sobre distintos actuadores desplegados en la ciudad. En este sentido, la combinación de datos multimodales, computación en el borde, comunicación LoRa/LoRaWAN y agregación inteligente en la nube abre la puerta a servicios avanzados de recomendación, regulación y asistencia a la movilidad. A continuación se describen tres escenarios de aplicación especialmente relevantes para el contexto urbano de Albacete.

A. *Recomendaciones dinámicas de tráfico mediante paneles inteligentes*

Un primer caso de uso consiste en reutilizar la infraestructura de paneles inteligentes ya desplegada en la ciudad para mostrar recomendaciones dinámicas de circulación en momentos de alta demanda. Este escenario resultaría especialmente útil en horas punta recurrentes, como las franjas de entrada y salida laboral, los momentos de recogida en centros educativos, así como en situaciones extraordinarias tales como partidos del Albacete Balompié, operaciones salida y retorno, periodos festivos o los días de mayor afluencia de la Feria de Albacete. En todos estos casos, la disponibilidad de datos en tiempo real procedentes de sensores situados tanto en el interior del casco urbano como en los accesos y salidas de la ciudad permitiría estimar de forma continua el estado de la red viaria y anticipar la formación de retenciones. Un ejemplo de cómo se podrían representar estos datos se encuentra en la Figura 8.

Desde el punto de vista operativo, este servicio se apoyaría en modelos ligeros desplegados en servidores *edge* de bajo coste, capaces de estimar densidad de tráfico, longitud de cola y tiempo esperado de recorrido hacia puntos de interés urbano, tales como el Hospital General, el barrio Imaginalia o el Parque Lineal. A partir de dichas estimaciones, el sistema podría generar mensajes de recomendación comprensibles para la ciudadanía y mostrarlos en paneles de mensajería variable distribuidos por la ciudad. Algunos ejemplos de estos mensajes serían: “congestión de tráfico en circunvalación, zona Universidad, diríjase por Parque Lineal para ahorrar X minutos” o “retención en salida a autovía A-30 por N-430 Carretera de Valencia, diríjase a salidas 4A y 4B por N-301”. De este modo, los paneles dejarían de ser elementos pasivos de información ambiental o de aparcamiento para convertirse en actuadores inteligentes sobre la demanda de movilidad.



Fig. 8: Ejemplo conceptual de recomendaciones dinámicas de tráfico mediante paneles inteligentes, a partir de datos recogidos en accesos periurbanos, rondas de circunvalación y puntos críticos de la ciudad.

En este escenario, la comunicación LoRa/LoRaWAN resulta especialmente relevante en los puntos más alejados del núcleo urbano, donde los accesos a autovías y carreteras nacionales pueden encontrarse a varios kilómetros de la ciudad. La utilización de estas tecnologías permitiría desplegar nodos de sensorización de bajo coste en entradas y salidas periurbanas, manteniendo un consumo energético reducido y una cobertura suficiente para alimentar al sistema con información actualizada. Además, en situaciones de saturación o limitada cobertura terrestre, este caso de uso podría complementarse con soluciones DtS-IoT en aquellos puntos en los que se requiera resiliencia adicional de las comunicaciones.

B. *Reprogramación dinámica de semáforos y cambio de sentido de calles durante la Feria de Albacete*

Un segundo caso de uso de especial interés para la ciudad de Albacete es la reconfiguración dinámica de la regulación viaria durante la Feria. Se trata de un escenario especialmente exigente desde el punto de vista logístico, ya que concentra en un área relativamente reducida una afluencia que, en sus momentos de mayor intensidad, puede llegar a duplicar ampliamente la población habitual de la ciudad. Esta situación genera un incremento muy notable de los desplazamientos motorizados, cambios en los patrones habituales de movilidad, restricciones temporales de tráfico y una mayor presión sobre los itinerarios de acceso y evacuación del entorno ferial. La implantación de la Zona de Bajas Emisiones añade, además, nuevas restricciones operativas que hacen todavía más necesario disponer de mecanismos adaptativos de gestión.

En este contexto, la arquitectura propuesta podría extenderse para actuar directamente sobre la regulación semafórica y sobre la configuración de determinadas calles mediante señalización luminosa inteligente. Más concretamente, se plantea el uso de paneles LED capaces de mostrar, de forma dinámica y según las necesidades detectadas por el sistema, indicaciones de prohibido el paso, sentido único, sentido doble temporal u obligación de giro a la derecha o a la izquierda. Esta funcionalidad permitiría adaptar la red viaria en función de la demanda observada en tiempo real, redistribuyendo flujos de entrada y salida al recinto ferial, habilitando corredores de evacuación o reforzando itinerarios alternativos cuando se detecten situaciones de saturación.

La toma de decisiones podría apoyarse en información procedente de cámaras de conteo, sensores medioambientales, ocupación de aparcamientos, afluencia peatonal y condiciones meteorológicas, todo ello procesado localmente en los nodos *edge* y agregado posteriormente en la nube. Sobre esta base, el sistema podría seleccionar, para cada franja temporal, la configuración semafórica más adecuada y el esquema de circulación temporal que minimice la congestión y reduzca el tiempo de acceso a los principales aparcamientos, al recinto ferial y a las rondas de distribución. No obstante, por tratarse de actuaciones con impacto directo sobre la seguridad vial, este caso de uso requeriría incorporar restricciones fuertes de operación, validación previa de escenarios y supervisión por parte de la autoridad municipal competente.

C. *Integración con la aplicación "Albacete Bus" o desarrollo de una aplicación específica de movilidad*

Un tercer caso de uso consiste en integrar las capacidades de predicción y recomendación de la arquitectura propuesta en la aplicación municipal "Albacete Bus", o alternativamente, en una nueva aplicación específica para la movilidad urbana de Albacete. En este escenario, la aplicación actuaría como un canal software de comunicación bidireccional entre el sistema y la ciudadanía. De este modo, cuando se detectasen retenciones significativas, incidencias puntuales o periodos de alta concurrencia, la plataforma podría recomendar de forma personalizada el uso de determinadas líneas de autobús urbano, desaconsejar otras temporalmente o sugerir combinaciones más eficientes entre desplazamiento a pie, vehículo privado y transporte público.

Este caso de uso resulta particularmente pertinente en una ciudad como Albacete, donde el autobús constituye el único medio de transporte público colectivo. Así, ante episodios de congestión en los accesos al centro, al Hospital General, a Imaginalia o al entorno del Recinto Ferial, la aplicación podría informar sobre tiempos estimados de llegada, incidencias de circulación, alteraciones temporales de recorrido y conveniencia relativa de usar una determinada línea frente al vehículo privado. Del mismo modo, en escenarios de saturación extraordinaria, la aplicación podría advertir de que ciertas líneas no constituyen una alternativa recomendable debido a desvíos, baja regularidad o elevada ocupación.

La relación entre la aplicación y la arquitectura ITEWAN sería, además, de naturaleza simbiótica. Por un lado, la aplicación se beneficiaría de la información inferida por los modelos de predicción de tráfico y por la monitorización distribuida del sistema. Por otro, los propios autobuses urbanos constituirían una fuente de datos de gran valor para mejorar la calidad del modelo global, aportando información sobre velocidad comercial, cumplimiento horario, localización, tiempos de parada e incidencias de servicio. Esta realimentación permitiría enriquecer la caracterización del estado del tráfico y mejorar, de forma progresiva, la calidad de las recomendaciones emitidas tanto a la ciudadanía como a los gestores municipales.

En conjunto, los tres escenarios anteriores ilustran cómo la arquitectura propuesta puede evolucionar desde una función de monitorización inteligente hacia una plataforma integral de actuación sobre la movilidad urbana. La combinación de sensorización distribuida, aprendizaje federado y actuadores físicos y software permitiría no solo describir el estado del sistema viario, sino intervenir sobre él de forma adaptativa, con especial interés en contextos urbanos dinámicos y de elevada variabilidad como los que presenta la ciudad de Albacete.

V. Conclusiones

El proyecto ITEWAN nos puede permitir la viabilidad conceptual de integrar IoT, redes LoRaWAN y aprendizaje federado para la gestión inteligente del tráfico urbano en escenarios dinámicos como los grandes eventos de Albacete. La arquitectura propuesta es una apuesta prometedora que nos puede ofrecer ventajas claras en eficiencia energética, privacidad y escalabilidad, frente a soluciones centralizadas tradicionales. Además, el análisis del contexto urbano y la identificación de fuentes de datos multimodales sientan las bases para futuros desarrollos experimentales y validaciones en entornos reales, consolidando una línea de investigación con alto potencial de transferencia tecnológica y aplicación en ciudades medianas.

Agradecimientos

Este trabajo ha sido financiado por MCIN/AEI/10.13039/501100011033 y el Fondo Europeo de Desarrollo Regional (FEDER), "Una forma de hacer Europa" (ref. PID2024-158682OB-C32), por la subvención 2023-GRIN-34056 financiada por la Junta de Castilla-La Mancha para Grupos de Investigación Consolidados, y por la Cátedra Ciudad de Albacete de la Universidad de Castilla-La Mancha.

Referencias

[1] Iqra Rafiq, Anzar Mahmood, Sohail Razzaq, S. Hassan M. Jafri, and Imran Aziz, "Iot applications and challenges in smart cities and services," *The Journal of Engineering*, vol. 2023, no. 4, pp. e12262, 2023.

[2] LoRa Alliance, "Lorawan l2 1.0.4 specification," https://resources.lora-alliance.org/technical-specifications/ts001-1-0-4-lorawan-l2-1-0-4-specification, 2020, Accessed: 2026-04-07.

[3] Mohammed Jouhari, Nasir Saeed, Mohamed-Slim Alouini, and El Mehdi Amhoud, "A survey on scalable lorawan for massive iot: Recent advances, potentials, and challenges," *IEEE Communications Surveys Tutorials*, vol. 25, no. 3, pp. 1841–1876, 2023.

[4] Omair Rashed Abdulwareth Almanifi, Chee-Onn Chow, Mau-Luen Tham, Joon Huang Chuah, and Jeevan Kanesan, "Communication and computation efficiency in federated learning: A survey," *Internet of Things*, vol. 22, pp. 100742, 2023.

[5] C. Garrido-Hidalgo, L. Roda-Sanchez, F. J. Ramírez, A. Fernández, and T. Olivares, "Efficient online resource allocation in large-scale lorawan networks: A multi-agent approach," *Computer Networks*, 2023.

[6] B. Liu, N. Lv, Y. Guo, and Y. Li, "Recent advances on federated learning: A systematic survey," *Neurocomputing*, vol. 597, 2024.

[7] V. Hidalgo, C. Carrión, and B. Caminero, "Deployment of federated learning on a low-cost distributed infrastructure," in *Proceedings of the 2nd IEEE International Conference on Federated Learning Technologies and Applications (FLTA 2024)*, 2024.

[8] Asociación de Empresarios de Campollano, "Parque Empresarial Campollano," https://adeca.com/parque-empresarial-campollano/, 2026.

[9] El Digital de Albacete, "Un nuevo tramo de la AB-20, en marcha: "Una obra de gran envergadura y enorme interés para el crecimiento y desarrollo de Albacete"," https://acortar.link/2R0N9l, octubre 2025.

[10] Manuel Castillo-Cara Blanca Caminero Giovanny Mondragón-Ruiz, Alonso Tenorio-Trigoso and Carmen Carrión, "An Experimental Study of Fog and Cloud Computing in CEP-based Real-Time IoT Applications," *Journal of Cloud Computing*, vol. 10-32, 2021.

[11] Ahmed Imteaj, Urmish Thakker, Shiqiang Wang, Jian Li, and M. Hadi Amini, ""A Survey on Federated Learning for Resource-Constrained IoT Devices"," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 1–24, Jan. 2022.

[12] Víctor Hidalgo, Maria Blanca Caminero, and María del Carmen Carrión, "Despliegue de aprendizaje federado semiasíncrono sobre una infraestructura distribuida de bajo coste," in *Actas de las Jornadas SARTECO*, 2025.

[13] eSMARTcity.es, "Albacete monitoriza los niveles de contaminación ambiental y acústica de la ciudad mediante estaciones de medición de Kunak," https://acortar.link/PQbToO, enero 2022.

[14] M Baqer, "Lightweight federated learning approach for resource-constrained internet of things," *Sensors*, vol. 25, no. 18, pp. 5633, 2025.

[15] Quan Wang, Guangfei Ye, Fu Li, Qidong Chen, Jin Jiang, Songyang Zhang, and Farhan Ullah, "Yolo-light: A real-time lightweight model based on limited edge computing for traffic scene detection and tracking," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–16, 2026.

[16] Sharnil Pandya, Gautam Srivastava, Rutvij Jhaveri, M. Rajasekhara Babu, Sweta Bhattacharya, Praveen Kumar Reddy Maddikunta, Spyridon Mastorakis, Md. Jalil Piran, and Thippa Reddy Gadekallu, "Federated learning for smart cities: A comprehensive survey," *Sustainable Energy Technologies and Assessments*, vol. 55, pp. 102987, 2023.

[17] Wei Huang, Dexian Wang, Xiaocao Ouyang, Jihong Wan, Jia Liu, and Tianrui Li, "Multimodal federated learning: Concept, methods, applications and future directions," *Information Fusion*, vol. 112, pp. 102576, 2024.

[18] Hui Jiang, Shihan Chen, Tao Ouyang, Min Lin, Xiaodong Zhang, Qimeng Zhang, and Xu Chen, "Mfel-h2b: Multimodal federated edge learning with heterogeneity-aware balancing across modalities and models," *High-Confidence Computing*, p. 100384, 2026.

[19] "LoRa Alliance," https://lora-alliance.org, 2026.

[20] "LoRa Alliance positions LoRaWAN as the leading standard for massive IoT connectivity," *LoRa Alliance Press Release*, 2026.

[21] Ramesh G.P. Rachana-P. Jangam, N.R., "A review on smart road traffic management system using LoRaWAN," in *2022 Distributed Computing and Optimization Techniques. Lecture Notes in Electrical Engineering*, 2022.

[22] Brtka V. Jotanović-G. et al. Dobrilović, D., "The urban traffic noise monitoring system based on LoRaWAN technology"," in *2022 Wireless Netw 28, 441–458*, 2022.

[23] Lynda Mokdad Jun Ling Mi Chen, Jalel Ben-Othman, "Urban expressway traffic state forecast via graph neural network and LoRaWAN communication," *IEEE Internet of Things Journal*, vol. 12, 2025.

[24] -, "LoRaWAN Protocol Expands Network Capacity with New Long Range Frequency Hopping Spread Spectrum Technology," https://blog.semtech.com/, 2020, Accedido el 14-04-2026.

[25] A. A. B. Sajak M. Ayoub Kamal, M. M. Alam and M. Mohd Su'ud, "Requirements, deployments, and challenges of LoRa Technology: A survey," *Computational Intelligence and Neuroscience*, vol. 1, 2023.

[26] M. Alves E. Sallum, N. Pereira and M. Santos, "Improving quality-of-service in LoRa low-power Wide-Area Networks through optimized radio resource management," *Journal of Sensor and Actuator Networks*, vol. 9, 2020.

[27] V. Ranasinghe y otros, "Rapid and resilient LoRa leap: A novel multi-hop architecture for decentralised earthquake early warning systems," *Sensors*, vol. 24, 2024.

[28] A. Jacob C. M. Raju A. Vithayathil, A. Mohamed and J. Jacob, "LoRa based wireless network for disaster rescue operations," *IEEE Conference on Advances in Computing and Communications (ICACC)*, vol. 24, 2021.

[29] Senlin Luo Malak Abid Ali Khan, "Dual resource allocation for enhancing vehicle tracking using deep neural network and bayesian LoRa," *Expert Systems with Applications*, vol. 303, 2026.

[30] SOOYEOB JUNG et al.), "LR-FHSS Transceiver for Direct-to-Satellite IoT Communications: Design, Implementation and Verification," *IEEE Open Journal of Vehicular Technology*, vol. 303, 2025.

[31] Konstantin Mikhaylov Muhammad Asad Ullah and Hirley Alves, "An Overview of Direct-to-Satellite IoT: Opportunities and Open Challenges," in *2023 IEEE 9th World Forum on Internet of Things (WF-IoT)*, 2023.

[32] M.H. Alsharif, R. Kannadasan, W. Wei, K.S. Nisar, and A.-H. Abdel-Aty, ""A contemporary survey of recent advances in federated learning: Taxonomies
, applications, and challenges," *Internet of Things (Netherlands)*, vol. 27, 2024.

[33] Helio N. Cunha Neto, Jernej Hribar, Ivana Dusparic, Diogo Menezes Ferrazani Mattos, and Natalia C. Fernandes, ""A Survey on Securing Federated Learning: Analysis of Applications
, Attacks, Challenges, and Trends," *IEEE Access*, vol. 11, pp. 41928–41953, 2023.

[34] Tesfahunegn Minwuyelet Mengistu, Taewoon Kim, and Jenn-Wei Lin, ""A Survey on Heterogeneity Taxonomy
, Security and Privacy Preservation in the Integration of IoT, wireless sensor networks and federated learning," *Sensors*, vol. 24, no. 3, pp. 968, 2024.

[35] Parimala Boobalan, Swarna Priya Ramu, Quoc-Viet Pham, Kapal Dev, Sharnil Pandya, Praveen Kumar Reddy Maddikunta, Thippa Reddy Gadekallu, and Thien Huynh-The, ""Fusion of Federated Learning and Industrial Internet of Things: A survey"," *Computer Networks*, vol. 212, pp. 109048, July 2022.

[36] T. El-Sayed, A. Moustafa, M. Elrashidy, and A. El-Sayed, ""Optimizing Federated Learning Approach: Literature Survey and Open Points"," in *2023 3rd International Conference on Electronic Engineering (ICEEM)*, 2023.

Arquitecturas, lenguajes y sistemas tolerantes a fallos (1/2)

Implementación en FPGA de una Neurona de Impulsos en Tiempo Real con Salida de Pulsos Pseudoaleatorios

Dionicio Yukitaro Kono Garcia[1], María de Lourdes Rivas Becerra[1], María Regina Rivas Becerra [1] Juan José Raygoza Panduro[2], Edwin Christian Becerra Alvarez [2] Susana Ortega Cisneros [3]

Resumen— **En este trabajo se presenta la implementación en tiempo real del prototipo en hardware de una neurona de impulsos en FPGA, que genera secuencias en impulsos pseudoaleatorios. El prototipo de la neurona fue desarrollado en hardware digital, reproduciendo comportamientos característicos de sistemas neuromórficos mediante señales tipo *spike*. Para la adquisición de la señal, se emplearon dos metodologías basadas en comunicación UART a 9600 baudios: una integrada en la FPGA y otra mediante un microcontrolador ESP32. El consumo de recursos de la implementación del diseño fue inferior al 1 %, evidenciando alta eficiencia y escalabilidad. La señal generada se evaluó con la prueba de *Linear Complexity*, del conjunto de pruebas estadísticas del NIST obteniendo niveles de pseudoaleatoriedad mayores a 0.01, que es el mínimo requerido por el NIST. Los resultados validan la viabilidad del sistema para la generación de señales pseudoaleatorias en tiempo real y su aplicación en hardware reconfigurable.**

Palabras clave— **FPGA, neurona de impulsos, NIST, señales pseudoaleatorias, tiempo real.**

I. Introducción

En el año 2003 fue presentado el modelo de Izikevich, que fue desarrollado para entender la estructura de los trenes de impulso. De la cual se obtienen 20 comportamientos distintos de neuronas pulsantes, utilizando (1), (2) y (3). Donde, $'$ es la derivada con respecto al tiempo, v el potencial de membrana, u es la variable de recuperación, a la escala de tiempo de la variable de recuperación u, b representa la sensibilidad de variable de recuperación u a las fluctuaciones de subumbral del potencial de membrana, c es el valor de reset que adopta el potencial de membrana después del impulso y d describe el reset después del impulso de la variable de recuperación u [1], [2], [3].

$$v' = 0{,}04v^2 + 5v + 140 - u + I \quad (1)$$

$$u' = a(bv - u) \quad (2)$$

$$si\ v \geq +30mV\ entonces\ \{v \leftarrow c, \quad u \leftarrow u + d \quad (3)$$

Este tipo de neuronas pulsantes se basan en modelos matemáticos anteriores, como el Leaky Integrate-and-Fire, Lapicque, Hodgkin y Huxley, Fitzhugh-Nagumo y Hindmarsh Rose y Morris-Lecar [3]. Algunas aplicaciones de estas neuronas pulsantes son en la clasificación de caracteres, el análisis de datos, en generadores de números pseudoaleatorios (PRNG),los cuales son sistemas deterministas que buscan que las secuencias de salida tengan una apariencia aleatoria. Estos sistemas dependen de un dato inicial conocido como semilla, que es proporcionado por un generador de números aleatorios (RNG) [4][5][6].

Estos generadores se caracterizan por utilizar variables físicas como el ruido de circuitos electrónicos, efectos cuánticos, movimientos de mouse, combinación de la temperatura y la hora, entre otros. Estos sistemas deben ser evaluados por diferentes pruebas estadísticas, que son de utilidad para medir los niveles de aleatoriedad y pseudoaleatoriedad. Un conjunto de pruebas muy utilizado son las propuestas por el NIST, que son 15 pruebas estadísticas que evalúan de forma independiente las secuencia producidas por el sistema. Donde el nivel de aleatoriedad se considera satisfactorio solo si *P-value* que es el valor probabilístico, es mayor o igual a 0.01. De lo contrario el resultado es completamente no aleatorio [4], [5], [6], [7], [8].

II. Neurona adaptación de frecuencia con salida de impulsos pseudoaleatoria

El diseño del prototipo en hardware de la neurona corresponde al comportamiento de adaptación de frecuencia, caracterizado por trabajar a una frecuencia que disminuye gradualmente hasta alcanzar una menor que la inicial. Para obtener este comportamiento, se sustituyen las variables a, b, c, d, v e I en (1), (2), y (3) de acuerdo a los parámetros proporcionados por Izhikevich. para la implementación en hardware, se realizó una máquina de estados como se observa en la Fig. 1, que tiene una palabra binaria fija de 9 bits que hace que la neurona produzca secuencias de impulsos periódicos[9].

En la Fig. 2 se observa el bloque general del prototipo en hardware de la neurona con salida de impulsos pseudoaleatorios, que fue diseñada inspirada en las neuronas pulsantes del modelo de Izikevich. Esta consiste en tomar la neurona con comportamiento de adaptación de frecuencia y conectarla a un generador de números pseudoaleatorios (PRNG). La salida del PRNG se conecta a un registro para almacenar los datos y pasarlos a la máquina de estados de la neurona en hardware. Para que cada estado

[1]Dpto. de Electro-Fotónica, Universidad de Guadalajara, e-mail: {dionicio.kono, maria.rivas2804, maria.rivas9199}@alumnos.udg.mx.

[2]Dpto. de Electro-Fotónica, Universidad de Guadalajara, e-mail: {juan.rpanduro, edwin.becerra}@academicos.udg.mx

[3]CINVESTAV del IPN, Campus Guadalajara, e-mail: susana.ortega@cinvestav.mx

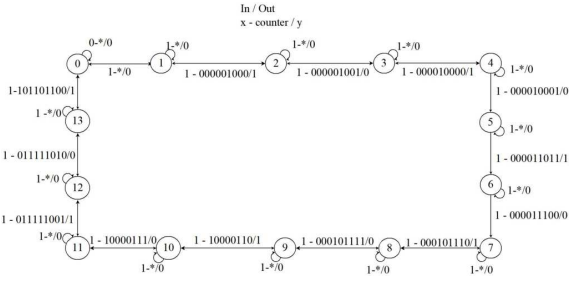


Fig. 1: Máquina de estados de la neurona adaptación de frecuencia [2].

esté cambiando constantemente la palabra binaria y se obtenga una salida de impulsos pseudoaleatorios en la neurona.

En este trabajo se muestra el prototipo de la neurona que se verificó en simulación y se implementó en tiempo real, los datos generados por la neurona en hardware se almacenaron en un archivo de texto TXT y para el análisis de los niveles de pseudoaleatoriedad se analizó mediante la prueba estadística *Linear Complexity* propuesta por el NIST, que demuestra que el diseño del prototipo en hardware de la neurona produce secuencias de impulsos complejos y pseudoaleatorios.

Los métodos utilizados para la adquisición de los impulsos pseudoaleatorios del prototipo, utilizan la comunicación serial UART, configurada a una velocidad de 9600 baudios en un microcontrolador ESP32 y una tarjeta FPGA. El ESP32 se utiliza en conjunto con una tarjeta FPGA, donde se implementa el diseño de la neurona, permitiendo trasmitir los datos leídos por el ESP32 y posteriormente se captura la información mediante una computadora por comunicación USB. Sin embargo, en la segunda metodología se implementa de forma directa la UART en la tarjeta FPGA.

Además, se demostró que su implementación en dispositivos reconfigurables es flexible, debido a los recursos utilizados en la FPGA y los resultados estadísticos del nivel de pseudoaleatoriedad de los impulsos producidos.

III. Adquisición de datos en tiempo real

Para la adquisición de datos en tiempo real, se emplearon dos metodologías diferentes de transmisión entre la FPGA y la computadora. En ambos casos se utilizó el protocolo de comunicación serial UART, configurado a una velocidad de 9600 baudios. Para comparar el desempeño de dos enfoques distintos de captura y transferencia de datos.

A. Dispositivo utilizado

El prototipo en hardware de la neurona fue implementado en la tarjeta de desarrollo *Basys 3*, que integra una FPGA de la familia *Artix-7 XC7A35T-1CPG236C*. Los recursos de este dispositivo, permiten la implementación eficiente de modelos neuronales digitales en tiempo real. que desde el punto de vista físico, la tarjeta *Basys 3* proporciona diversos elementos que resultan útiles para la implementación y validación del sistema neuronal. Entre ellos destaca el oscilador de 100 MHz, que sirve como fuente de reloj principal para la operación del prototipo de la neurona y la sincronización de los módulos internos.

La tarjeta cuenta con una interfaz USB-UART integrada, que permite la comunicación directa con una computadora sin necesidad de hardware adicional. Este recurso fue utilizado para la transmisión de los datos generados por el prototipo de la neurona, facilitando su adquisición, visualización y almacenamiento en tiempo real. Los pines de entrada/salida (GPIO) disponibles en la tarjeta, permiten la conexión con instrumentos externos como osciloscopios o analizadores lógicos, para la validación experimental de las señales generadas. En conjunto, estas características físicas hacen de la plataforma una herramienta adecuada para el desarrollo, prueba y análisis de sistemas neuronales digitales implementados en FPGA.

B. Uso de ESP32 como puente de adquisición

En esta metodología se utilizó un ESP32 como intermediario entre la FPGA y la computadora, como se observa en la Fig. 3. El prototipo en hardware de la neurona adaptación de frecuencia fue implementado en la FPGA, donde la salida de impulsos pseudoaleatorios fue transmitida al ESP32 mediante dos pines digitales. Uno dedicado a la señal de datos del generador y otro a la señal de reloj CLK, funcionando como intermediario entre la FPGA y la computadora.

Los datos capturados por el ESP32 fueron trasmitidos a la computadora mediante comunicación serial UART. Los cuales se almacenaron en un archivo de texto en ASSCII hexadecimal por la herramienta Realterm, y usando un programa en C++ se convirtió a ASSCII binario para poder analizarlo por la prueba *Linear Complexity* de las pruebas estadísticas del NIST.

C. Implementación directa de UART en FPGA

En la segunda metodología que se observa en la Fig. 4, se implementó directamente un módulo de transmisión UART en la FPGA. En este caso, los datos generados por el prototipo de la neurona fueron procesados internamente en la FPGA mediante un registro SIPO (*Serial-In Parallel-Out*), que agrupa los bits producidos en palabras de 8 bits. Donde cada byte se envía a un bloque transmisor UART configurado a 9600 baudios, que se encarga de transmitir los datos a la computadora a través de la interfaz serial. Al igual que en la metodología anterior, los datos son almacenados en un archivo de texto en ASSCII hexadecimal haciendo uso de Realterm y convertidos a ASSCII binario para someterlo a la prueba *Linear Complexity* del NIST.

D. Frecuencias del sistema

El prototipo en hardware de la neurona adaptación de frecuencia produce datos a una tasa aproximada de 100 Mbps. Superando la capacidad de transmisión

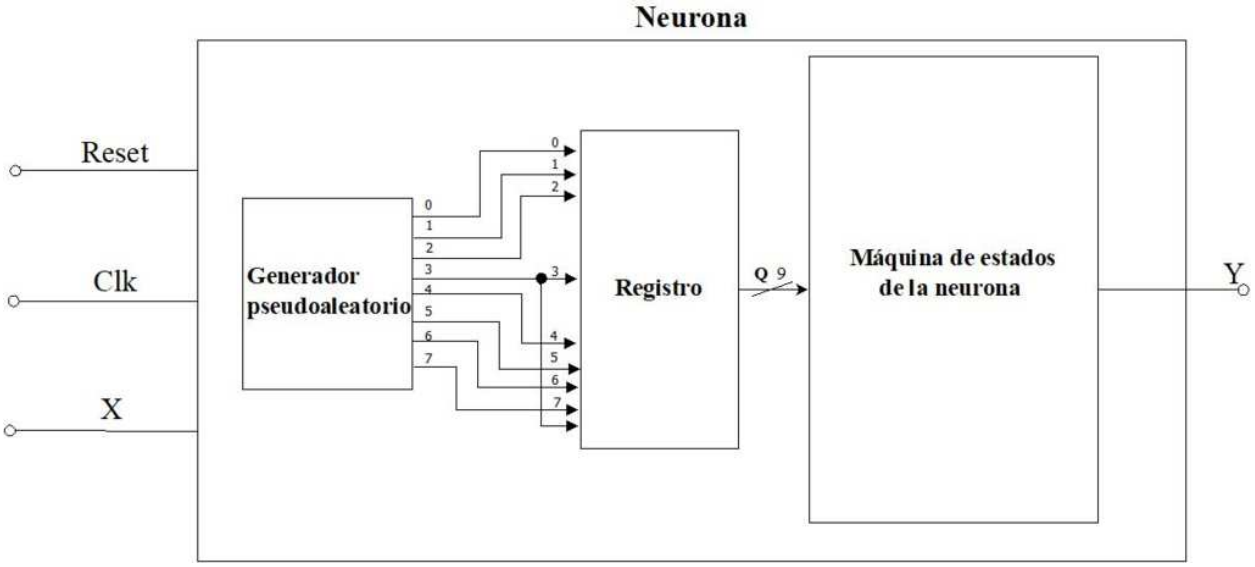


Fig. 2: Diseño del prototipo en hardware de la neurona con salida de impulsos pseudoaleatorios.

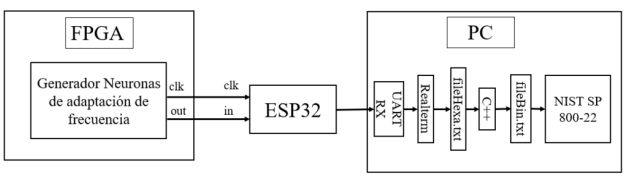


Fig. 3: Arquitectura del método con ESP32.

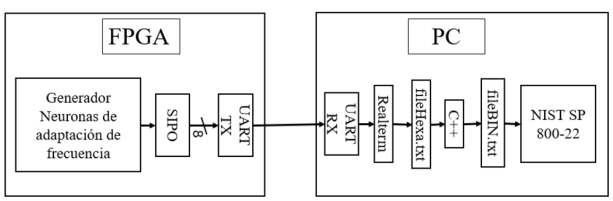


Fig. 4: Arquitectura del método con FPGA.

del protocolo UART configurado a 9600 baudios. En esta configuración, la UART permite transmitir hasta 9600 símbolos por segundo, lo que equivale aproximadamente a 76,800 bps considerando tramas de 8 bits.

Esta discrepancia entre la tasa de generación de datos y la capacidad del canal de comunicación introduce un problema de desajuste de tasas (rate mismatch), que impide la correcta observación y adquisición de la señal en tiempo real. Para abordar este problema, se implementó una estrategia de adaptación de tasa (rate matching) a nivel de hardware, donde se ajusta la frecuencia de operación del sistema para alinearla con la velocidad efectiva de transmisión.

Como resultado, se determinó que el sistema debía operar a una frecuencia cercana a 76 kHz, para permitir establecer una correspondencia directa entre la generación y la transmisión de datos, y evitar pérdidas de información, facilitando la visualización continua de la señal. Esta adaptación garantiza que cada bit generado pueda ser transmitido sin saturar el canal de comunicación, permitiendo así una caracterización precisa del comportamiento del prototipo de la neurona en tiempo real.

IV. Resultados

El diseño del prototipo en hardware de la neurona de impulsos fue descrito en VHDL, utilizando la herramienta Vivado de Xilinx AMD. Se verifica por medio de la simulación el comportamiento del diseño. Así como los porcentajes de ocupación utilizados en el dispositivo reconfigurable FPGA *Artix-7 XC7A35T-1CPG236C*.

A. Simulación y porcentajes de ocupación en la FPGA Artix-7 XC7A35T-1CPG236C

En la Fig. 5 se muestra la simulación obtenida del prototipo en hardware de la neurona, con salida de impulsos pseudoaleatoria a una frecuencia de 100 MHz. Se resalta parte de la respuesta de impulsos pseudoaleatorios en un tiempo de 110 us a 140 us, donde, se muestra mayor cantidad de impulsos generados, indicando una frecuencia inicial mayor que va disminuyendo gradualmente. La respuesta en impulsos de simulación fue impresa en un archivo de texto para someterla a la prueba *Linear Complexity* del conjunto de pruebas del NIST.

En la Tabla I se presenta la comparación de ocupación de recursos entre las implementaciones de adquisición con FPGA y FPGA conectada al ESP32. Se observa que ambas soluciones utilizan una fracción muy reducida de los recursos disponibles en el dispositivo, destacando un consumo inferior al 1 % en LUTs y Flip-Flops, y aproximadamente el 1 % en memoria BRAM. Asimismo, no se emplean bloques DSP en ninguna de las implementaciones. Estos resultados evidencian la alta eficiencia en el uso de recursos, permitiendo la integración de módulos adicionales sin comprometer la capacidad del dispositivo, siendo especialmente relevante para sistemas embebidos y arquitecturas mcomplejas [10].



Fig. 5: Simulación del prototipo uno para la generación de impulsos pseudoaleatorios con una frecuencia de 100 MHz.

Tabla I: Porcentajes de ocupación de recursos entre ambas metodologías para la adquisición y procesamiento digital de datos en la FPGA *Artix-7 XC7A35T-1CPG236C*.

Recurso	**FPGA**	**FPGA Y ESP32**	**Disponible**
LUT	76 (0.37 %)	49 (0.24 %)	20,800
Flip-Flops	83 (0.20 %)	44 (0.11 %)	41,600
BRAM	0.5 (1.00 %)	0.5 (1.00 %)	50
DSP	0 (0.00 %)	0 (0.00 %)	90

B. Metodología en tiempo real con el ESP32

La señal de salida del prototipo de la neurona en tiempo real y la señal de CLK se conectaron al ESP32. Donde la captura del dato presente en el pin de salida se hace cuando se presenta un flanco de subida en el CLK.

Con el sistema trabajando a una frecuencia de 76 kHz, el microcontrolador captura la secuencia en impulsos del prototipo de la neurona. Sin embargo, se presentan datos atípicos explicados por la relación temporal entre la señal de la neurona y el instante de muestreo utilizado por el microcontrolador. En algunos casos, el valor lógico del dato se presenta muy cercano al flanco de muestreo que dispara la interrupción, lo que puede provocar que la lectura ocurra cuando el dato aún no se ha estabilizado o ya ha cambiado de estado. Como consecuencia, el sistema puede capturar un valor incorrecto.

En la Fig. 6 se observa la arquitectura que incorpora un módulo ESP32 como dispositivo intermediario entre la FPGA y la PC. En este caso, la FPGA envía los datos hacia el ESP32, el cual actúa como puente de comunicación, reenviando la información hacia la computadora.

C. Metodología en tiempo real con UART integrado

Para este método se implementó el protocolo UART directamente en la FPGA. A través de un registro de desplazamiento actualizado a la misma frecuencia del reloj del sistema, se realizó el agrupamiento de los bits generados en paquetes de 1 byte. Este esquema permite seriar la información, de modo que cada conjunto de 8 bits es encapsulado y transmitido mediante el protocolo UART, facilitando su adquisición y el análisis en tiempo real.

Por otro lado, en la Fig. 7 se muestra la arquitectura de adquisición de datos en la que la FPGA se conecta directamente a una PC mediante comunicación UART. Donde, la FPGA genera los datos provenientes del prototipo de la neurona y los transmite en formato hexadecimal hacia la computadora. Al igual que en la metodología anterior, los datos se reciben en la PC, se almacenan en un archivo de texto con valores hexadecimales y se procesan para convertirlos a formato binario. El resultado de esta conversión se guarda en un archivo de texto, permitiendo así su análisis detallado a nivel de bit, que es necesario en las pruebas estadísticas del NIST.

La diferencia entre las metodologías influye directamente en los resultados obtenidos por la prueba *Linear Complexity* del conjunto de pruebas estadísticas del NIST. Donde en simulación se obtiene una secuencia libre de latencias, ruido ambiental y efectos físicos, lo que permite evaluar el comportamiento teórico del sistema bajo condiciones completamente controladas.

Sin embargo, en la implementación directa con FPGA, al no existir un elemento intermedio entre el emisor y el receptor, la probabilidad de pérdida o alteración de bits es baja. Preservando las propiedades estadísticas originales de la secuencia generada, permitiendo que la prueba refleje de manera más fiel la secuencia producida por el prototipo en hardware de la neurona de adaptación de frecuencia implementado. Por otro lado, al incorporar el ESP32 como intermediaria, se introduce una mayor probabilidad de ruido, así como un incremento en la latencia entre la señal generada y su adquisición. Estos factores pueden tener efectos como la introducción de errores o pérdidas de bits que pueden generar patrones artificiales o dependencias que reduzcan la complejidad.

Por lo tanto, los resultados obtenidos mediante la prueba de complejidad lineal no solo dependen del sistema, sino también del método de adquisición y de las condiciones físicas del entorno. Resaltando la importancia de considerar la cadena completa de medición como parte integral del sistema, especialmente en aplicaciones donde se evalúan propiedades estadísticas de señales digitales en tiempo real.

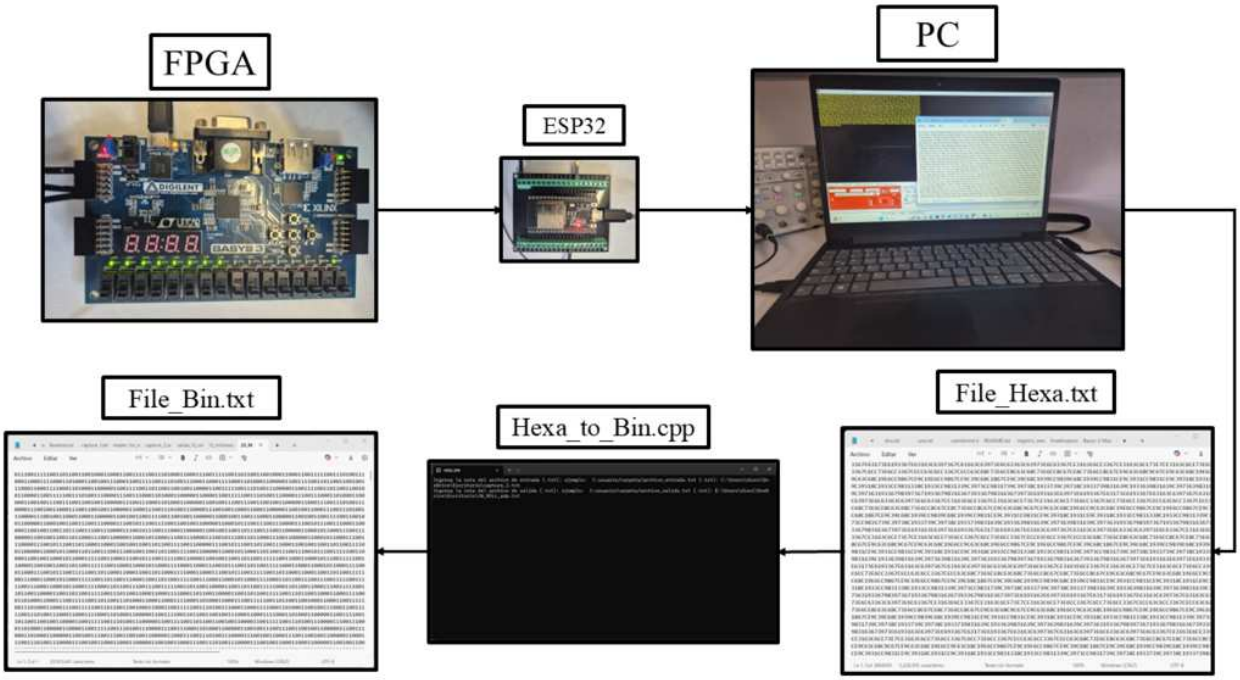


Fig. 6: Arquitectura del método de adquisición en tiempo real utilizando ESP32 como puente de comunicación entre la FPGA y la computadora.

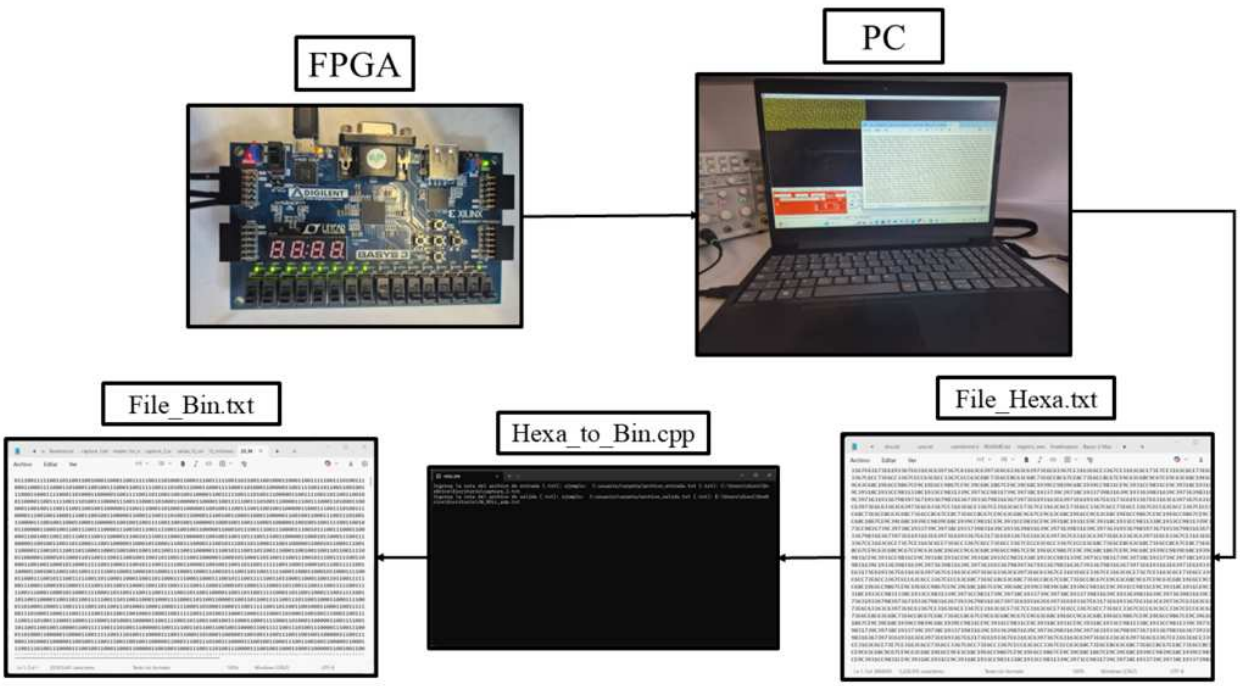


Fig. 7: Arquitectura del método de adquisición en tiempo real utilizando ESP32 como puente de comunicación entre la FPGA y la computadora.

D. Resultado de la prueba Linear Complexity del NIST

Del prototipo en hardware de la neurona, se obtuvieron las secuencias de impulsos producidos en simulación y en tiempo real para evaluar el nivel de pseudoaleatoriedad mediante la prueba *Linear Complexity* del NIST, de acuerdo con su complejidad y distribución de los impulsos producidos, como se observa en la Fig. 8. Las secuencias fueron particionadas en un millón de bits y en 10 millones de bits, de los cuales, por cada caso, se aplicó la prueba estadística. Donde los niveles superaron el mínimo requerido por el NIST, que es de $P\text{-}value = 0.01$ para determinar que la secuencia es pseudoaleatoria, como se observa en la Tabla III.

En simulación la secuencia en impulsos tiene una media de pseudoaleatoriedad de 0.855208, siendo este el valor de *P-value* para ambas longitudes de la secuencia . Por otro lado, cuando se implementó en el dispositivo ESP32, la media de los datos estuvo aproximadamente en un nivel de pseudoaleatoriedad de 0.7938195, con un máximo de $P\text{-}value = 0.985805$, correspondiente a un millón de bits, y, al implemen-

Tabla II: Comparación entre metodologías de captura de datos.

Característica	**Metodología con ESP32 como puente**	**Metodología con UART implementado en FPGA**
Complejidad de implementación	Baja, el manejo de comunicación se realiza en el microcontrolador.	Media, requiere diseñar e implementar el módulo UART y lógica adicional.
Velocidad de captura	Limitada por la capacidad del microcontrolador.	Mayor eficiencia al realizar el procesamiento directamente en hardware.
Pérdida de datos	Mayor probabilidad por tener un sistema intermedio.	Baja probabilidad de pérdida de datos, ya que los bits se agrupan y transmiten con registros SIPO dentro del mismo FPGA.
Latencia del sistema	Mayor latencia debido al dispositivo intermedio.	Menor latencia al eliminar el microcontrolador intermedio.
Flexibilidad	Alta, el microcontrolador permite agregar procesamiento o nuevas interfaces fácilmente.	Menor flexibilidad, cualquier modificación requiere cambios en el diseño hardware.
Uso de recursos de FPGA	Bajo, la FPGA solo genera los datos.	Mayor, debido al uso de registros SIPO y del módulo transmisor UART.
Estabilidad de transmisión	Dependiente del software del microcontrolador.	Determinística al estar implementada completamente en hardware.
Herramienta de captura en PC	Datos recibidos mediante RealTerm.	Datos recibidos mediante RealTerm.

tarlo en la FPGA, la media fue de 0.8733955, con un máximo de *P-value* de 0.910728, correspondiente a de millón de bits.

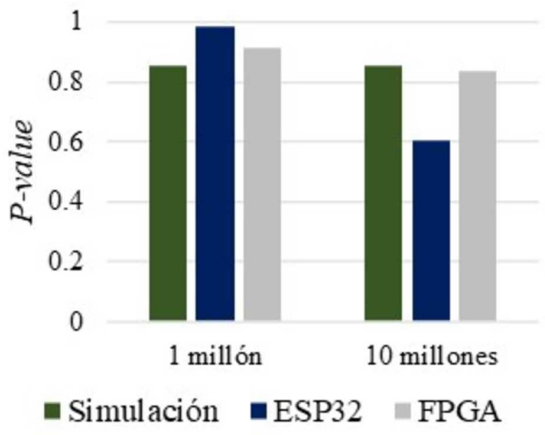


Fig. 8: Niveles de pseudoaleatoriedad de la salida en impulsos de la neurona en simulación y tiempo real.

Tabla III: Niveles de pseudoaleatoriedad de la secuencia de impulsos producida por el prototipo en hardware de la neurona.

	1 millón	10 millones
Simulación	0.855208	0.855208
ESP32	0.985805	0.601834
FPGA	0.910728	0.836063

En la gráfica de la Fig. 9, se muestran los promedios obtenidos de las secuencias en simulación, implementada en el ESP32 y en la FPGA. Donde, el promedio del nivel de pseudoaleatoriedad fue de 0.85520800 para la secuencia de impulsos obtenida en simulación del prototipo, para el ESP32 fue de 0.79381950 y en la FPGA de 0.87339550. Con esto se demuestra que el prototipo en hardware de la neurona adaptación de frecuencia produce impulsos pseudoaleatorios adecuadamente distribuidos y complejos en simulación e implementado en dispositivos reconfigurables.

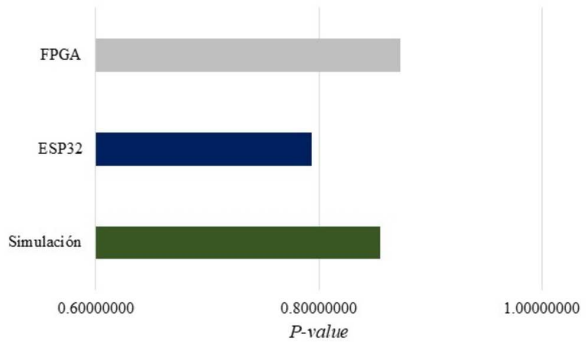


Fig. 9: Promedio de los niveles de pseudoaleatoriedad de la salida en impulsos de la neurona en simulación y tiempo real.

V. Conclusiones

En este trabajo se demostró que la implementación en tiempo real del prototipo en hardware de la neurona de impulsos, puede producir secuencias de impulsos pseudoaleatorios complejos y adecuadamente distribuidos según los resultados obtenidos por la prueba *Linear Complexity* del conjunto de pruebas estadísticas del NIST. Para la implementación del prototipo, se desarrollaron dos metodologías para la adquisición de los impulsos de salida de la neurona, ambas basadas en el protocolo UART a una velocidad de 9600 baudios.

En ambos métodos se muestran diferencias en los porcentajes de ocupación en los recursos disponibles en la FPGA, donde, el consumo en la mayoría de estos llegó a ser inferior al 1 % en la implementación directa en la FPGA y al hacer uso del ESP32. Esto indica que el diseño propuesto es escalable, permitiendo la integración de múltiples neuronas o módulos adicionales sin comprometer significativamente la capacidad del dispositivo.

Además, la adquisición de datos permitió validar el comportamiento pseudoaleatorio de la neurona mediante la prueba *Linear Complexity* del conjunto de pruebas del NIST. La cual mostró valores de *P-value* dentro de rangos aceptables tanto en simulación como en las implementaciones en hardware (FPGA y

ESP32). Se observó que el nivel más alto de *P-value*, corresponde al implementarlo en la FPGA haciendo uso del UART integrado. Además, se observa que, incluso al modificar la longitud de la secuencia, se obtuvieron valores superiores al umbral de aceptación establecido, definido como un p-value mayor o igual a 0.01.

Estos resultados indican que la señal generada conserva propiedades pseudoaleatorias adecuadas independientemente de la plataforma de implementación, validando así la robustez del modelo propuesto frente a variaciones en hardware. Es importante considerar que el sistema de adquisición puede introducir valores atípicos, que pueden repercutir en los resultados obtenidos y influir en las mediciones en la evaluación estadística.

Agradecimientos

El presente trabajo ha sido apoyado por la Secretaría de Ciencia, Humanidades, Tecnología e Innovación SECIHTI y por la Universidad de Guadalajara.

Referencias

[1] I. C. Matadamas Ortiz, "Aplicación de las redes neuronales pulsantes en el reconocimiento de patrones y analisis de imágenes," M.S. thesis, Instituto Politécnico Centro de Investigación en Computación, 2014.

[2] M. R. Rivas Becerra, "Diseño en hardware de un kernel utilizando neuronas de impulsos con activación limitada en el tiempo," M.S. thesis, Universidad de Guadalajara, Centro Universitario de Ciencias Exactas e Ingenierías, 2026.

[3] M. Okun K. Toth, "The impact og hodgkin-huxley models on dendritic research," *The Journal of Physiology*, vol. 601, no. 15, pp. 3091–3102,https://doi.org/10.1113/JP282756, 2022.

[4] Sonia Akter, Kasem Khalil, and Magdy Bayoumi, "Efficient pseudo random number generator (prng) design on fpga," in *2024 IEEE 17th Dallas Circuits and Systems Conference (DCAS)*, 2024, pp. 1–5.

[5] Bhagyashree Ingle and Milind Nemade, "Random number generators in cryptography and simulation: Comparative analysis of trng and prng designs," in *2025 3rd International Conference on Computational Intelligence and Network Systems (CINS)*, 2025, pp. 1–6.

[6] Benjamin Williams, Albert Carlson, and Sai Ranganath Mikkilineni, "Permuting the outputs of rngs is not secure," in *2026 IEEE 16th Annual Computing and Communication Workshop and Conference (CCWC)*, 2026, pp. 1301–1305.

[7] Berlekamp-Massey, *Encyclopedia of Cryptography and Security*, chapter Berlekamp-Massey Algorithm, Springer, 2011.

[8] A. Rukhin; J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, and et al., "A statistical test suit for random and pseudorandom number generators for cryptographic applications," Tech. Rep., National Institute of Standards and Technology U.S. Department of Commerce, 2010, url: https://www.nist.gov/publications/statistical-test-suite-random-and-pseudorandom-number-generators-cryptographic.

[9] I. Macias Mendoza, J.J. Raygoza Panduro, Edwin Christian Becerra Alvarez, M. Jimenéz Rodríguez, J.L González Vidal, and J.R. Reyes Barón, "DiseÑo comportamental de neuronas de impulso en hardware reconfigurable basadas en el modelo de izhikevich (behavioral design of spiking neurons in reconfigurable hardware based on izhikevich model," *Pistas Educativas*, vol. 43, no. 141, 2022, ISSN:2448-847X.

[10] Ian Kuon and Jonathan Rose, "Fpga architecture: Survey and challenges," *Foundations and Trends in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2007.

[11] M. L. Rivas Becerra, J.J. Raygoza Panduro, S. Ortega Cisneros, Edwin Christian Becerra Alvarez, and J.D. Rios Arrañaga, "Design of a new neuro-generator with a neuronal module to produce pseudorandom and perfectly pseudorandom sequences," *Electronics*, vol. 13(10), 2024, doi:https://doi.org/10.3390/electronics13101955.

[12] M. L. Rivas Becerra, J.J. Raygoza Panduro, Edwin Christian Becerra Alvarez, S. Ortega Cisneros, and J.L.G. Vidal, "Design and hardware implementa-tion of a highly flexible prng system for nist-validated pseu-dorandom sequences.," *Chips*, vol. 4, 2025.

[13] M. L. Rivas Becerra, Juan José Raygoza P., Edwin Christian Becerra Alvarez, Jaime Rios A., Mario Jiménez, and Susana Ortega-Cisneros, "Impulse neurons: Phasic bursts and tonic bursts, to generate pseudorandom sequences," in *2023 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*, 2023, vol. 7, pp. 1–6, doi:10.1109/ROPEC58757.2023.10409418.

[14] E.M. Izhikevich, "Simple model of spiking neurons," vol. 14, no. 6, pp. 1569–1572, doi: 10.1109/TNN.2003.820440, 2003.

[15] Mihaly Lazar and Laszlo T. Toth, "Perfect recovery and sensitivity analysis of time encoding machines," *IEEE Transactions on Circuits and Systems I*, vol. 51, no. 10, pp. 2060–2073, 2004.

Medición de rendimiento energético en entornos HPC mediante el uso de xCAT

Miguel Baños González [1], Jorge Valle Neila [2], Javier Corral García[1], Juan Antonio Rico Gallego[1], Félix O. García[3]

Resumen— **Durante los últimos años se han producido avances en el entorno de Computación de Altas Prestaciones (HPC) orientados a mejorar la eficiencia energética de los Centros de Procesamiento de Datos, especialmente con el auge de tecnologías con consumos especialmente altos como los LLM, o los modelos de IA generativa. En este contexto, se presenta una metodología experimental, empírica y reproducible para evaluar de forma simultánea el tiempo de ejecución, el consumo energético y la potencia en entornos de HPC, abordando la limitada atención que tradicionalmente recibe el análisis energético a nivel de nodo en el software científico. La propuesta se valida en el clúster LUSITANIA II (COMPUTAEX) mediante el estudio de 26 técnicas de optimización, comparando versiones estándar y eficientes bajo distintos niveles de optimización del compilador GCC v.7.2.0 (O0 a O3). Los resultados evidencian una fuerte correlación lineal entre mejoras en tiempo y energía ($R^2 = 0,993$), con reducciones energéticas de hasta $99,86\,\%$, aunque con variaciones dependientes del nivel de compilación, la potencia media y el tamaño del problema. Como contribución, se define un pipeline completo de medición y análisis que integra instrumentación temporal, muestreo de potencia, alineamiento de trazas, cálculo energético y generación automatizada de resultados, facilitando un análisis reproducible y sistemático de la eficiencia energética en HPC y aportando evidencia experimental de que las optimizaciones de rendimiento también conllevan mejoras significativas en consumo energético.**

Palabras clave— **Computación de Altas Prestaciones, Eficiencia Energética, Medición de Potencia y Energía, Optimización de Código, Green Software.**

I. Introducción

El crecimiento de la demanda de Computación de Altas Prestaciones (HPC) y la diversificación de sus usuarios han favorecido el desarrollo de software centrado principalmente en requisitos funcionales, relegando la optimización del rendimiento y del consumo de recursos. En este contexto, la eficiencia energética adquiere un papel clave en los centros de datos que alojan sistemas HPC, donde el consumo energético impacta directamente en costes operativos y sostenibilidad. A diferencia del rendimiento temporal, la energía depende no solo del tiempo de ejecución, sino también del perfil de potencia, lo que introduce posibles desacoples entre ambas métricas. Por ello, resulta necesario abordar la optimización desde una perspectiva integral que considere conjuntamente tiempo, energía y potencia, analizando los compromisos existentes entre estas magnitudes.

[1]COMPUTAEX, Centro de Supercomputación de Extremadura, 10071 Cáceres, España, e-mail: {miguel.banos, javier.corral, juanantonio.rico}@computaex.es
[2]Universidad de Extremadura, e-mail: jvallene@alumnos.unex.es
[3]Universidad de Castilla-La Mancha, e-mail: felix.garcia@uclm.es

En este contexto, se analiza la eficiencia energética de diversas técnicas de optimización aplicadas a código C/C++ en entornos HPC reales mediante una metodología experimental reproducible que compara versiones estándar y optimizadas de cada técnica bajo distintos niveles de compilación, evaluando de forma sistemática el tiempo de ejecución, la energía consumida y la potencia media. El análisis se centra en un conjunto representativo de técnicas, incluyendo transformaciones de bucles, mejoras de localidad de memoria y estrategias de paralelización, permitiendo no solo cuantificar las mejoras obtenidas, sino también identificar dependencias con el nivel de optimización, el tamaño del problema y el comportamiento de la potencia.

Los experimentos han sido validados en un entorno HPC real (el supercomputador LUSITANIA II de la Fundación COMPUTAEX[1]), permitiendo caracterizar de forma precisa el impacto energético de cada técnica. A partir de la comparación entre versiones estándar y eficientes, se calculan porcentajes de mejora en tiempo, energía y potencia, lo que facilita el análisis de patrones y casos no triviales en los que una reducción del tiempo de ejecución no implica necesariamente un menor consumo energético. Este enfoque permite determinar si las optimizaciones orientadas al rendimiento contribuyen también a mejorar la eficiencia energética global, aportando una visión más completa y aplicable a la operación eficiente de infraestructuras HPC.

II. Trabajos relacionados

El crecimiento de la infraestructura digital y de los sistemas HPC ha incrementado significativamente la demanda energética de los centros de datos (CPD), impulsado por el aumento del tráfico de datos y de los servicios digitales [2]. Aunque el consumo eléctrico ha mostrado cierta estabilización gracias a mejoras de eficiencia y a la consolidación en centros de hiperescala, la tendencia futura sigue siendo incierta ante cargas cada vez mayores [2]. En HPC, esta presión es aún más evidente: el coste energético y de refrigeración puede superar al de adquisición, condicionando la viabilidad de las infraestructuras [3], [4]. Este cambio ha desplazado el foco desde el rendimiento absoluto hacia el rendimiento bajo restricciones energéticas, reflejado en la coexistencia de rankings como *Top500* y *Green500*, que evidencian la tensión entre rendimiento y eficiencia [5], [6].

El consumo energético en HPC puede traducirse a impacto ambiental mediante la huella de carbono (CO_2e), calculada como el producto entre la energía

consumida y la intensidad de carbono del suministro eléctrico [7]. En este contexto, el impacto más relevante suele situarse en el Alcance 2 (electricidad consumida), aunque puede ampliarse considerando el PUE del CPD y otros alcances [7], [8]. Las magnitudes implicadas son elevadas: sistemas HPC pueden alcanzar consumos anuales del orden de decenas de GWh, lo que se traduce en miles de toneladas de CO_2e [7]. Esto implica que pequeñas mejoras porcentuales en eficiencia energética pueden generar reducciones absolutas significativas cuando se agregan en el tiempo.

Las técnicas de programación eficiente consisten en modificaciones localizadas del código orientadas a mejorar el rendimiento sin alterar la funcionalidad, especialmente relevantes en rutas críticas de ejecución [9]. Estas técnicas han sido sistematizadas y evaluadas experimentalmente, mostrando mejoras significativas en tiempo de ejecución, particularmente en entornos HPC [10], [11], [12].

La evaluación del consumo energético depende del método empleado (medición directa, modelos internos como RAPL o simulación) y de la granularidad considerada [13]. En HPC, la medición a nivel de nodo resulta especialmente relevante para capturar el impacto global de la ejecución, aunque introduce retos de sincronización y comparabilidad [6]. Iniciativas como *Green500* han contribuido a estandarizar la medición de potencia, pero las diferencias metodológicas siguen afectando a la interpretación de resultados [4]. Además, los modelos de estimación pueden perder precisión en entornos complejos, lo que refuerza la necesidad de medición empírica para una atribución fiable del consumo [14].

El estado del arte muestra que la eficiencia energética es una restricción estructural en HPC, aunque la evaluación del software continúa dominada por métricas de rendimiento temporal. Frente a ello, se plantea la necesidad de analizar conjuntamente tiempo, energía y potencia mediante mediciones reproducibles y a nivel de nodo, asegurando una correcta atribución del consumo durante la ejecución. Este enfoque permite caracterizar los compromisos entre rendimiento y eficiencia, proporcionando una visión más completa y alineada con los requisitos reales de operación en infraestructuras HPC.

III. Métodos y Materiales

En esta propuesta se define una metodología experimental orientada a medir de forma simultánea y atribuible el tiempo de ejecución y el consumo energético en HPC, evitando la variabilidad inherente al sistema (DVFS, interferencias, estado térmico). Para ello, la potencia se muestrea durante la ejecución real y la energía se obtiene integrando únicamente dentro de la ventana temporal medida, garantizando coherencia entre ambas magnitudes. Este enfoque se organiza como un pipeline completo que abarca desde la instrumentación hasta el análisis, permitiendo evaluar conjuntamente eficiencia temporal, energética y su relación.

El framework compara dos variantes funcionalmente equivalentes de cada código (*standard* y *efficient*), compiladas con `GCC` bajo niveles `-O0` a `-O3`. Las comparaciones se realizan siempre dentro del mismo nivel de optimización para aislar el efecto de la técnica. A partir de tiempo y energía se deriva la potencia media, lo que permite identificar desacoples entre aceleración y consumo:

- **Energía (J)** por integración numérica (regla del trapecio):

$$E \approx \int_{t_0}^{t_1} P(t)\,dt \approx \mathrm{trapz(P, t)},$$

- **Potencia media (W)** como media aritmética de las muestras del intervalo (interpretación coherente con $E/\Delta t$ cuando el muestreo es suficientemente denso),
- **Duración** $\Delta t = t_1 - t_0$ para trazabilidad.

Esta metodología asegura que las diferencias observadas se deben exclusivamente a la técnica aplicada.

El siguiente pseudocódigo (Algorithm 1) describe cómo se capturan simultáneamente tiempo y potencia. En él se detalla el núcleo principal de la metodología:

- Se inicia el muestreo de potencia antes de ejecutar el experimento.
- Se ejecuta primero el bloque *standard*, midiendo múltiples muestras de tiempo y registrando marcas de inicio y fin.
- Se introduce una pausa controlada para separar claramente los perfiles energéticos.
- Se ejecuta el bloque *efficient* bajo el mismo esquema.
- Finalmente, se detiene el muestreo de potencia.

Algorithm 1 Metodología básica de medida.

```
procedure EjecutarExperimento
  N ← 10;  reps ← 600000000;  S ← 180      ▷ parámetros fijos
  IniciarMuestreoPotencia                  ▷ comienza la traza de potencia
Bloque estándar
  MarcarInicio(standard)                   ▷ timestamp inicio (std)
  for k ← 1 to N do
    muestras_std[k] ← MedirTiempoBloque(standard, reps)   ▷ 1 muestra temporal
  end for
  MarcarFin(standard)                      ▷ timestamp fin (std)
  GuardarResumenTiempo(standard, muestras_std)   ▷ CSV: tiempos + estadísticos
  Dormir(S)                                ▷ separación std/eff y cola energética
Bloque eficiente
  MarcarInicio(efficient)                  ▷ timestamp inicio (eff)
  for k ← 1 to N do
    muestras_eff[k] ← MedirTiempoBloque(efficient, reps)   ▷ 1 muestra temporal
  end for
  MarcarFin(efficient)                     ▷ timestamp fin (eff)
  GuardarResumenTiempo(efficient, muestras_eff)   ▷ CSV: tiempos + estadísticos
  DetenerMuestreoPotencia                  ▷ termina la traza de potencia
end procedure
```

Este algoritmo garantiza que la traza de potencia

cubre ambos bloques completos (estándar y eficiente) y que las diferencias observadas se deben únicamente al comportamiento del código.

A. Alineamiento temporal.

Durante las pruebas realizadas de forma premilinar para testear el funcionamiento de la metodología, se observó que las marcas de tiempo y las trazas de potencia no estaban perfectamente sincronizadas. Para paliar esto, se introdujo un algoritmo de alineamiento para calcular una cantidad de desfase (*offset*) óptimo, de forma que se pudiesen desplazar las ventanas temporales hasta que coincidiesen con los picos reales de consumo energético.

El algoritmo recorre con una ventana deslizante de tiempo, con ancho igual a la duración del test, buscando en qué offset se maximiza la potencia medida dentro de la ventana de tiempo. De esta manera, se busca que los picos de potencia medida, correspondientes a la duración de los tests, estén dentro de la ventana de medición. Para evitar que este desplazamiento pueda ecoger erroneamente un pico que correspnda a una prueba distinta, el valor de offset está limitado a 60 segundos antes y después de la marca de inicio medida desde el framework. Adicionalmente, la pausa existente entre los tests también facilita que este cálculo sea correcto.

B. Cálculo energético y métricas.

Una vez alineadas las ventanas, la energía se obtiene integrando la potencia (regla del trapecio) y se calcula la potencia media y la duración. Con estos valores se derivan métricas de mejora en tiempo, energía y potencia entre *standard* y *efficient*. Este enfoque permite analizar no solo la aceleración, sino también el coste energético asociado, identificando patrones no triviales y *trade-offs* entre rendimiento y eficiencia energética en HPC.

C. Materiales

En esta sección se describen los elementos empleados para sustentar la validación experimental. En particular, se presentan (i) el conjunto de técnicas de optimización evaluadas, (ii) el entorno de compilación utilizado para generar las variantes *standard* y *efficient* bajo distintos niveles de optimización, y (iii) la plataforma HPC donde se ejecutan las pruebas, con el objetivo de dejar explícitas las condiciones de trabajo y facilitar la trazabilidad y la reproducibilidad de los resultados.

Algorithm 2 Estimación del *offset* maximizando la potencia media dentro de las ventanas.

```
procedure EstimarOffset(tiempo_s, potencia_W, ventanas)
  (mejorOffset, mejorPuntuacion) ← (0, −∞)          ▷ inicialización
  for offset ← −60 to 60 step 0,5 do                ▷ offsets candidatos
    puntuacion ← CalcularPuntuacionOffset(offset, tiempo_s, potencia_W, ventanas)   ▷ desplazar ventanas y evaluar
    if puntuacion > mejorPuntuacion then
      (mejorOffset, mejorPuntuacion) ← (offset, puntuacion)   ▷ seleccionar el máximo
    end if
  end for
  return mejorOffset                                ▷ offset final
end procedure
```

C.1 Software de compilación

El software de compilación utilizado es `GCC` v7.2.0. La elección de esta versión responde a un criterio práctico ya que se trata de la versión instalada de forma estable en el supercomputador donde se han realizado las pruebas. De este modo, se evita introducir variabilidad externa derivada de cambios de versión, diferencias de configuración o dependencias adicionales, lo que refuerza la trazabilidad y la reproducibilidad de los resultados obtenidos. En este contexto, `GCC` constituye una opción adecuada para compilar código fuente en C y es, en términos generales, una herramienta de uso extendido en este tipo de entornos.

Como se ha detallado previamente, el compilador dispone de cuatro niveles principales de optimización (`-O0`, `-O1`, `-O2` y `-O3`). En consecuencia, la metodología experimental contempla la compilación y evaluación de cada variante (*standard* y *efficient*) bajo cada uno de estos niveles, de forma que el efecto de una técnica se analiza manteniendo fijo el contexto de compilación. Además, las técnicas estudiadas son complementarias a los niveles de optimización: se aplican como transformaciones a nivel de código fuente sobre la variante *efficient*, y su impacto se cuantifica comparándola con la variante *standard* dentro del mismo nivel `-Ok`, evitando así mezclar el efecto del compilador con el de la técnica evaluada.

C.2 Entorno experimental: LUSITANIA II

Todas las pruebas descritas (incluida la calibración del número de repeticiones y mediciones, así como de los parámetros de las técnicas) se han ejecutado en nodos pertenecientes al clúster LUSITANIA II, en un entorno de memoria distribuida con las siguientes características:

- 168 IBM System x iDataPlex dx360 M4,
- 2 procesadores Intel E5-2670 SandyBridge-EP por nodo,
- 8 cores por procesador a 2.6,GHz (16 cores por nodo; 2688 cores totales),
- 20 MB de caché,
- 32 GB de RAM por nodo.

Este aspecto es relevante porque la metodología se valida en un sistema HPC real, con gestor de colas (SLURM), monitorización a nivel de nodo y condiciones representativas de producción. Esto permite que los resultados obtenidos no se limiten a entornos controlados o de laboratorio, sino que reflejen el comportamiento en infraestructuras reales, reforzando así su validez externa y su aplicabilidad práctica.

D. Implementación

El sistema se implementa como un pipeline completo de medición y análisis en entorno HPC. En

la fase de ejecución, cada experimento genera dos ficheros sincronizados: (i) un CSV de performance con tiempos instrumentados (muestras, estadísticas y marcas de inicio/fin de los bloques *standard* y *efficient*), y (ii) un CSV de energy con la traza de potencia a nivel de nodo (pares potencia–timestamp con resolución de milisegundos). Ambos se emparejan mediante una convención de nombres y se procesan conjuntamente: se normalizan los tiempos, se aplica el alineamiento por *offset* y se calcula energía por integración dentro de cada ventana de ejecución.

La instrumentación temporal se realiza en C mediante `Watchtime`, con sobrecarga despreciable, siguiendo un patrón homogéneo en todos los experimentos: medición repetida de la carga de trabajo, almacenamiento de muestras y generación de marcas legibles de inicio/fin. Estas marcas permiten delimitar las ventanas que posteriormente se cruzan con la traza energética. El resultado por bloque se exporta a CSV incluyendo tiempos individuales, media, desviación estándar y tiempo normalizado por iteración. En paralelo, la potencia se captura mediante `rvitals` (xCAT) a través de `nodepow`, ejecutado desde un wrapper de SLURM que realiza muestreo periódico (3 s), genera timestamps precisos y escribe en un buffer temporal independiente por job, evitando colisiones en ejecuciones concurrentes.

A nivel de fiabilidad experimental, la implementación incorpora varias medidas clave: ejecución en nodo exclusivo (`--exclusive`) para evitar interferencias, exclusión de nodos con lecturas anómalas, separación temporal entre bloques mediante pausas controladas (180 s) para evitar solapamientos térmicos y facilitar la segmentación de la traza, y mecanismos robustos de escritura (buffer + writer en background). El sistema también contempla casos incompletos o corruptos en el emparejado `performance/energy`, descartando automáticamente ejecuciones no válidas. Finalmente, un módulo de post-proceso automatizado recorre todos los resultados, aplica el mismo pipeline de alineamiento e integración definido en la metodología y genera tablas estructuradas que constituyen la base numérica del análisis.

IV. Resultados globales de las 26 técnicas

En esta sección se analiza el comportamiento de las 26 técnicas listadas en la Tabla I, evaluadas con los distintos niveles de optimización de GCC (`O0`–`O3`) considerando mejoras porcentuales en energía, tiempo y potencia media ($P = E/t$). Valores positivos indican reducción (beneficio), mientras que negativos reflejan empeoramiento respecto a la línea base. La potencia media refleja el balance relativo entre energía y tiempo; cuando ambos mejoran proporcionalmente, su cambio es cercano a cero, mientras que desbalances generan variaciones significativas.

La Tabla I resume los porcentajes de mejora para cada técnica y nivel, permitiendo identificar una serie de conclusiones:

1. Existe una proporcionalidad entre reducción de tiempo y energía global.
2. Se muestra una relación entre el efecto de las técnicas y el nivel de optimización
3. Desacoples visibles en potencia: cuando la técnica produce un efecto notable en tiempo y potencia, la potencia diferencia de potencia es baja.

Los casos que destacan por no aparecer esta equivalencia energía–tiempo: T03, T05, T09, T12, T16, T17 y T23. Estos casos refuerzan la necesidad de evaluar el efecto sobre la energía de todas las técnicas, ya que las mejoras en tiempo no suponen mejoras de energía en casos concretos.

La figura 1 muestra que, para la mayoría de técnicas, energía y tiempo mejoran de forma equivalente, mientras que la potencia refleja pequeños desbalances según la proporción entre ambos. Las técnicas están ordenadas respecto a la cantidad de mejora media de la tres variables. Se observa que:

- Energía y tiempo suelen mejorar juntas; potencia varía según el balance.
- Patrones de potencia: mejora mínima, empeora mínima o equivalente, o invariante.

La comparación O0 vs O3 (Figura 2) confirma que energía y tiempo mantienen correspondencia, mientras que la potencia presenta variaciones irregulares. El análisis por nivel indica mejoras medias globales:

- O0: $\sim 21\,\%$ energía, $21{,}8\,\%$ tiempo.
- O3: $\sim 27\,\%$ energía, $26{,}1\,\%$ tiempo.

La potencia media permanece cercana a cero ($-0{,}5\,\%$ a $1{,}1\,\%$), consistente con la equivalencia energía-tiempo dominante.

Los diagramas de trade-off muestran:

- Tiempo vs energía: relación casi lineal ($R^2 = 0{,}993$), confirmando fuerte acoplamiento.
- Tiempo vs potencia y energía vs potencia: dispersión elevada ($R^2 \sim 0{,}18$), evidenciando que potencia no sigue patrón uniforme.

Como conclusiones de los resultados generales de las técnicas, podemos extraer principalmente cuatro:

- Energía y tiempo están fuertemente acopladas; la relación lineal es consistente a nivel global.
- Existen técnicas con comportamientos singulares por nivel (T03, T23, T12, T05, T09, T16, T17) donde la potencia varía notablemente. En general, se observa que en estos casos la relación energía–tiempo no se mantiene. Esto puede deberse a diversidad de razones (técnicas con uso más intensivo de memoria, por ejemplo). Este punto se analizará en futuros trabajos.
- Potencia media se mantiene cercana a cero y no constituye un indicador directo de beneficio global.
- Rankings confirman T10 y T21 como las técnicas que obtienen los mejores resultados consistentemente en todas las métricas.

En resumen, la mejora energética se explica mayo-

Tabla I: Tabla resumen de los resultados de las 26 técnicas

	Nombre de la Técnica	O0			O3		
		Energía (%)	Potencia (%)	Tiempo (%)	Energía (%)	Potencia (%)	Tiempo (%)
T01	Bit fields	37,18	-0,90	33,50	21,16	1,90	18,34
T02	Bit sets	91,02	-0,44	91,01	91,00	-0,33	91,01
T03	Boolean return	24,81	1,40	23,25	-2,15	-4,44	0,47
T04	Cascaded function calls	34,82	3,40	32,51	95,00	1,06	94,87
T05	Chains of pointers	18,22	-3,67	17,41	-1,69	-0,13	3,09
T06	Common subexpression elimination	26,37	-3,73	29,79	17,92	0,75	10,10
T07	Constants inside loops	23,69	-0,02	23,68	-13,67	-0,03	-13,52
T08	Constructor initialization lists	7,34	-0,18	7,54	6,31	-0,01	6,48
T09	Dead code elimination	-4,43	2,07	0,00	4,42	1,90	0,47
T10	Division by a power-of-two denominator	16,58	-4,18	16,46	3,85	2,74	0,78
T11	Exception handling	99,81	5,40	99,79	99,88	13,36	99,86
T12	Function inlining	37,88	-4,40	38,72	3,21	-7,37	0,00
T13	Global variables	0,26	0,72	-0,65	-1,63	0,21	-1,94
T14	Global variables within loops	-1,71	0,08	-1,85	65,92	0,57	65,81
T15	Initialization versus assignment	62,82	-2,27	62,23	77,13	5,64	73,84
T16	Integer versus character	-1,91	-0,82	1,83	3,60	-1,55	1,40
T17	Invariant IF statements within loops	-4,80	0,04	-4,92	86,56	0,83	85,73
T18	Linear search	15,48	0,20	15,30	14,29	1,52	12,63
T19	Loop count down	-29,78	0,60	-30,54	-14,68	-1,00	-13,50
T20	Loop unrolling	36,74	-3,21	38,79	41,56	0,35	41,31
T21	Mapping structures	-7,86	-3,09	-4,55	72,62	-3,03	73,41
T22	Multiplication by a power-of-two factor	-5,20	-0,06	-5,10	-14,16	-0,32	0,77
T23	Passing structures by reference instead of value	85,86	2,71	85,36	94,04	9,75	93,13
T24	Pointer aliasing	6,81	-2,44	6,67	7,84	0,33	9,61
T25	Pre-increment versus post-increment	9,84	-0,24	10,06	-38,29	0,40	-36,16
T26	Row-major accesing	-3,61	-0,11	-3,52	0,01	-4,96	1,07

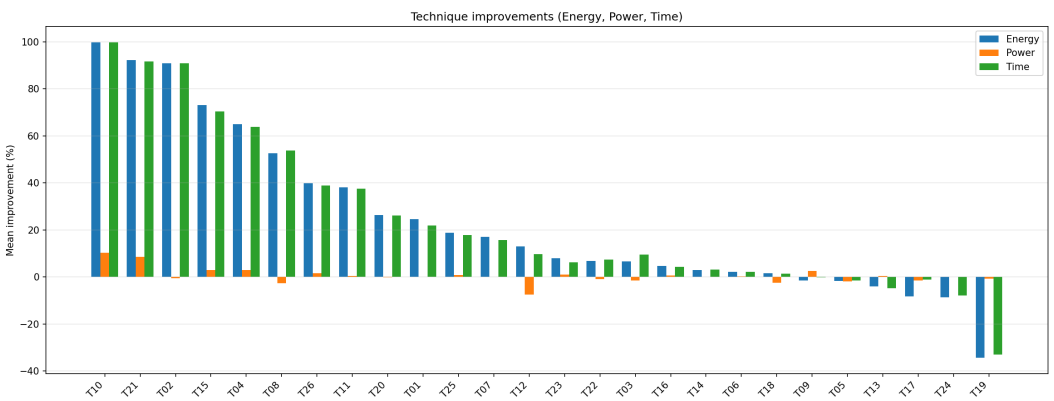
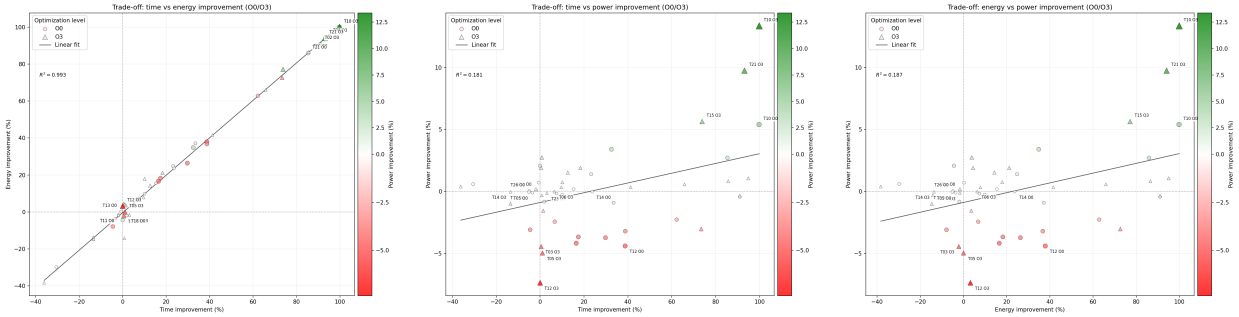


Fig. 1: Resultados globales (Conjunto de las 26 técnicas): resumen por técnica (comparativa global).



Fig. 2: Resultados globales (Conjunto de las 26 técnicas): comparación O0 vs O3.



(a) Tiempo vs energía (b) Tiempo vs potencia (c) Energía vs potencia

Fig. 3: Resultados globales (Conjunto de las 26 técnicas): trade-offs entre tiempo, energía y potencia.

ritariamente por reducción temporal equivalente; la potencia refleja desbalances específicos, siendo relevante solo en casos de desacople por técnica y nivel.

V. Análisis de las mejores técnicas

En las figuras Fig. 4a, 4b, 4c se pueden observar las 10 mejores técnicas en términos de reducción del consumo energético, potencia media y tiempo, respectivamente.

De un primer vistazo se observa que, para el caso de energía y tiempo (Fig. 4a, 4b), las 10 mejores técnicas son las mismas. Esto concuerda con las conclusiones extraídas en el apartado anterior acerca de la relación entre energía y tiempo. También se observa que los valores de reducción de tiempo y energía son muy parecidos, con una variación de aprox. $\pm 2\,\%$.

Por otra parte, respecto al comportamiento de la potencia, las técnicas que aparecen en esta gráfica no coinciden completamente, concretamente T08, T11, T20, T01 aparecen en tiempo y energía y no en potencia, mientras que T09, T23, T25, T16 y T13 aparecen potencia pero no en tiempo ni en energía. Como ya se comentaba en la anterior sección, se observa que estas técnicas corresponden con las que tienen una diferencia mínima en tiempo o energía. En general, se observa que la reducción de potencia no tiene implicación sobre el consumo energético ni el tiempo de ejecución.

A. *Bit fields*

La técnica *Bit fields* proporciona una reducción energética consistente en todos los niveles de optimización, con un comportamiento especialmente favorable en `-O0`, donde la mejora es significativamente mayor que en niveles superiores. A partir de `-O1`, tanto la energía como el tiempo muestran reducciones estables y muy similares entre sí, lo que indica que el impacto de la técnica se mantiene uniforme una vez activadas las optimizaciones del compilador.

En términos de relación entre métricas, energía y tiempo evolucionan de forma claramente acoplada, mientras que la potencia permanece prácticamente invariante. Esto sugiere que la técnica actúa reduciendo el trabajo total sin alterar de forma significativa la intensidad computacional, ofreciendo un beneficio global medio cercano al 24,67 %.

B. *Bit sets*

Presenta un comportamiento extraordinariamente estable entre todos los niveles de optimización, con valores absolutos de energía y tiempo prácticamente constantes. Esta estabilidad se traduce en mejoras porcentuales casi idénticas en todos los niveles, situadas en torno al 91 %, lo que indica que el efecto de la técnica es independiente del compilador.

Además, energía y tiempo están completamente alineados, mostrando reducciones equivalentes en todos los casos. Como consecuencia, la potencia se mantiene prácticamente constante, evidenciando que la técnica reduce proporcionalmente ambas magnitudes. En conjunto, se trata de una de las técnicas más robustas y efectivas.

C. *Cascaded function calls*

En este caso, el beneficio depende fuertemente del nivel de optimización, aumentando progresivamente desde `-O0` hasta `-O3`. La mejora energética pasa de valores moderados a alcanzar niveles muy elevados en `-O3`, lo que indica una clara sinergia con el compilador.

El tiempo sigue una evolución similar, aunque ligeramente por debajo de la energía, lo que genera pequeñas mejoras en potencia en niveles intermedios. En conjunto, la técnica se beneficia de las optimizaciones del compilador, amplificando su impacto conforme aumenta el nivel.

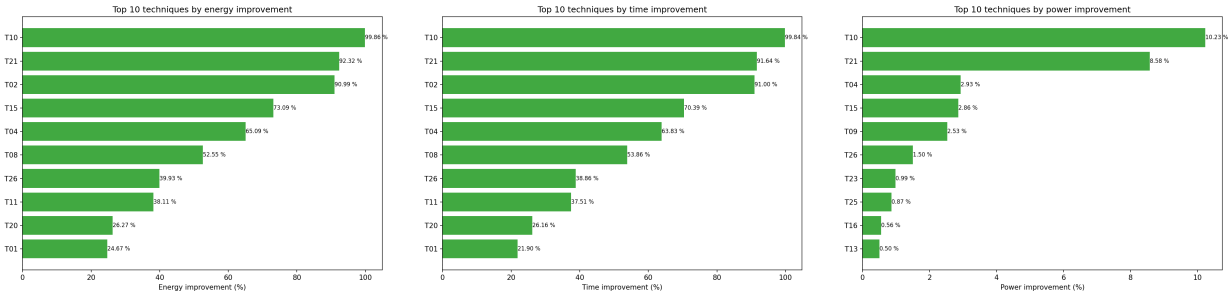
D. *Dead code elimination*

Esta técnica es dependiente del nivel de optimización, sin un patrón claro en energía. En `-O0`–`-O2` resulta contraproducente, aumentando el consumo energético, mientras que solo en `-O3` muestra una mejora positiva, aunque limitada.

El tiempo, por su parte, apenas varía, lo que genera un desacoplo claro entre energía y tiempo. Esta falta de coherencia entre métricas y niveles hace que la técnica no sea recomendable de forma general, siendo útil únicamente en casos concretos y evaluados específicamente.

E. *Exception handling*

La técnica *Exception handling* ofrece mejoras extremadamente altas y constantes en todos los niveles de optimización, con reducciones cercanas al 100 %



(a) Top 10 por energía (b) Top 10 por tiempo (c) Top 10 por potencia

Fig. 4: Resultados globales (Conjunto de las 26 técnicas): ranking top 10 según energía, tiempo y potencia.

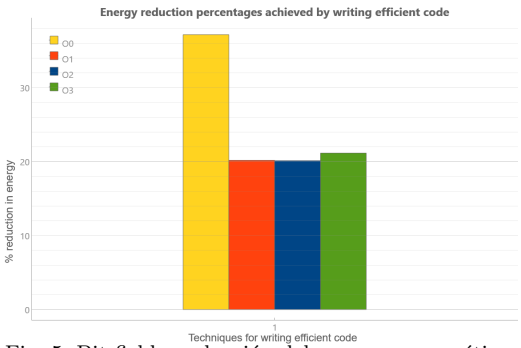


Fig. 5: Bit fields: reducción del consumo energético.

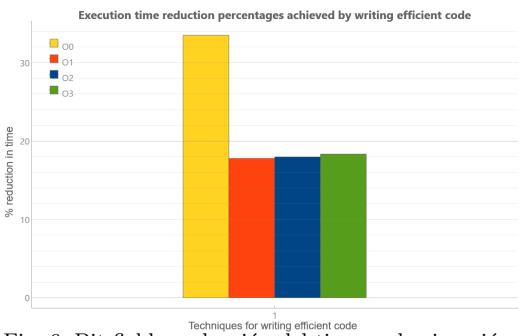


Fig. 6: Bit fields: reducción del tiempo de ejecución.

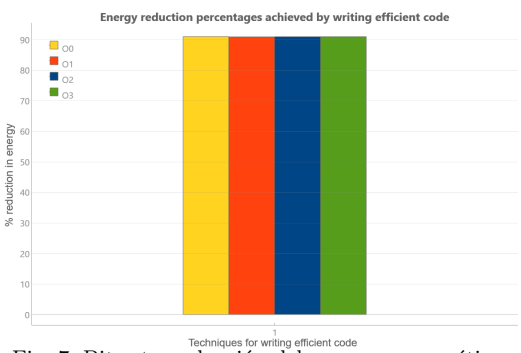


Fig. 7: Bit sets: reducción del consumo energético.

tanto en energía como en tiempo. La única diferencia relevante se observa en `-O0`, donde el consumo eficiente es ligeramente mayor, aunque sin impacto significativo en el resultado global.

Energía y tiempo están completamente alineados, mientras que la potencia aumenta progresivamente con el nivel. Este comportamiento indica que la técnica actúa de forma independiente del compilador, proporcionando un beneficio muy elevado y consistente en cualquier contexto.

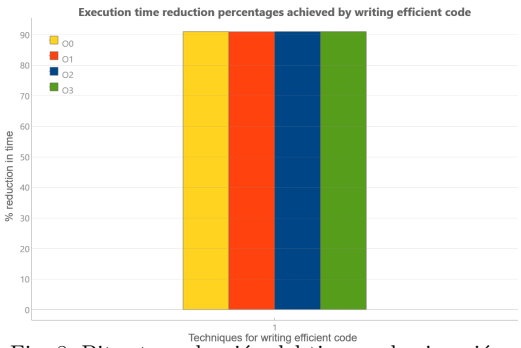


Fig. 8: Bit sets: reducción del tiempo de ejecución.

F. *Function inlining*

En *Function inlining* se observa un fuerte contraste entre `-O0` y niveles superiores. En `-O0`, la técnica mejora significativamente tanto energía como tiempo, mientras que en `-O1`–`-O3` el beneficio energético se mantiene pero el tiempo apenas mejora.

Este desacoplo provoca valores negativos en potencia y evidencia que el compilador ya captura gran parte del beneficio temporal en niveles altos. Así, la técnica resulta especialmente útil en ausencia de optimización, pero su impacto se reduce y se concentra únicamente en energía en niveles superiores.

G. *Initialization versus assignment*

La técnica *Initialization versus assignment* presenta mejoras elevadas y consistentes en energía y tiempo, con valores cercanos al 70 % en todos los niveles. La mejora es menor en `-O0`, pero se estabiliza en niveles superiores, donde alcanza sus máximos.

Energía y tiempo evolucionan de forma muy simi-

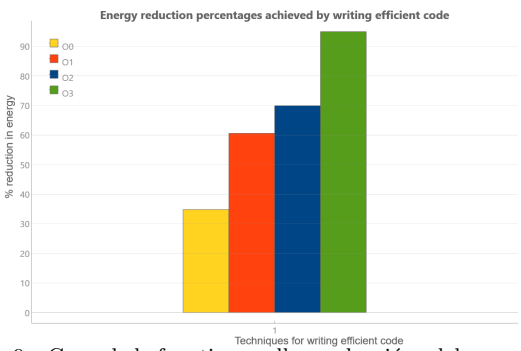


Fig. 9: Cascaded function calls: reducción del consumo energético.

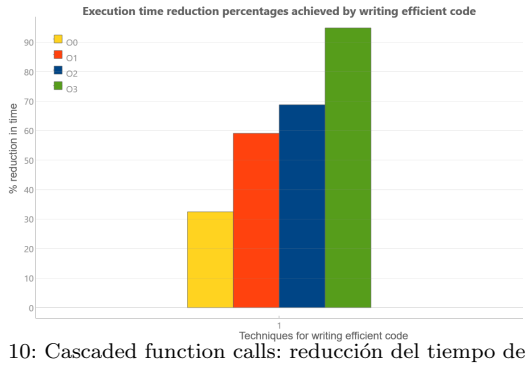


Fig. 10: Cascaded function calls: reducción del tiempo de ejecución.

lar, lo que indica un acoplamiento fuerte entre ambas métricas. Además, la técnica potencia el efecto del compilador, transformando casos donde el rendimiento empeora en *standard* en mejoras claras al aplicar la optimización.

H. Integer versus character

La técnica *Integer versus character* presenta un comportamiento irregular en energía, dependiendo fuertemente del nivel de optimización. Puede ser contraproducente en `-O0`, neutra en `-O2` y beneficiosa en `-O1` y `-O3`, aunque con mejoras pequeñas.

El tiempo, en cambio, mejora de forma consistente en todos los niveles, lo que genera un desacoplo con la energía y una potencia negativa. Esto obliga a aplicar la técnica de forma selectiva, evaluando su impacto según el nivel de compilación.

I. Passing structures by reference instead of value

La técnica *Passing structures by reference instead of value* ofrece mejoras muy elevadas en energía y

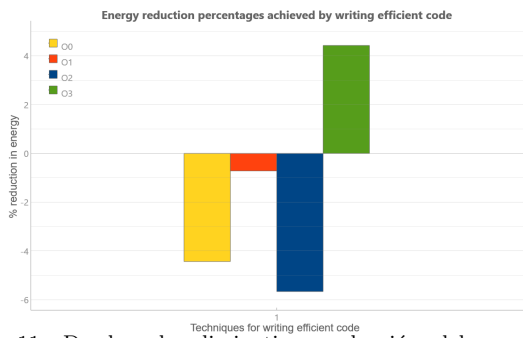


Fig. 11: Dead code elimination: reducción del consumo energético.

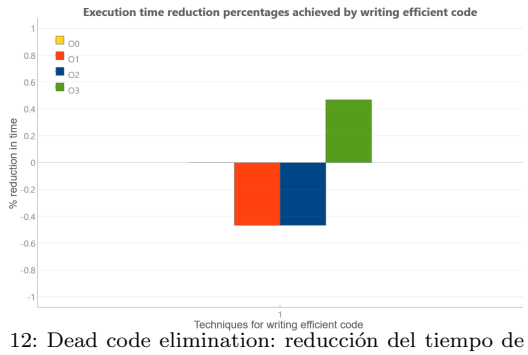


Fig. 12: Dead code elimination: reducción del tiempo de ejecución.

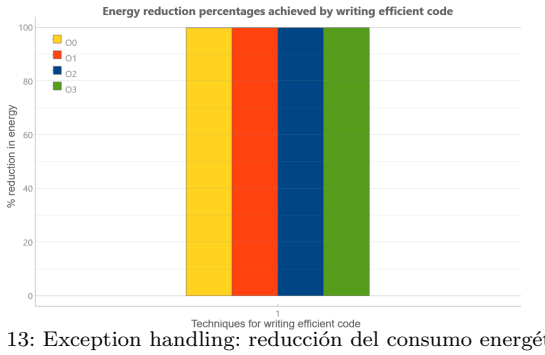


Fig. 13: Exception handling: reducción del consumo energético.

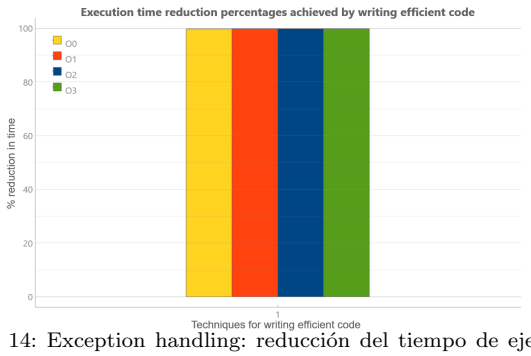


Fig. 14: Exception handling: reducción del tiempo de ejecución.

tiempo, superiores al 90 % en promedio. Aunque existe cierta variabilidad entre niveles, todos presentan beneficios significativos, destacando `-O2` como el caso más favorable.

Energía y tiempo evolucionan de forma muy similar, lo que indica un comportamiento consistente, mientras que la potencia muestra variaciones moderadas. La interacción con el compilador es relevante, ya que en algunos niveles amplifica notablemente el efecto de la técnica.

VI. Conclusiones

Este trabajo presenta una metodología de medida simultánea de tiempo y energía en entornos HPC, implementada como un pipeline completo desde la ejecución en SLURM hasta la generación de tablas y figuras listas para su posterior análisis pormenorizado. En un solo proceso se integra el muestreo de potencia dentro de ventanas temporales de test, corrige desfases entre marcas de tiempo y automatiza la agregación de resultados.

A nivel de resultados, se han testeado 26 técnicas de programación eficiente ya conocidas y presentadas

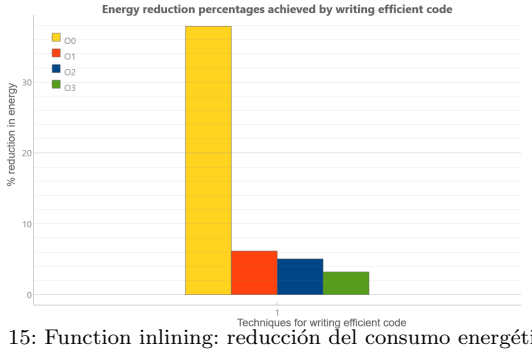


Fig. 15: Function inlining: reducción del consumo energético.

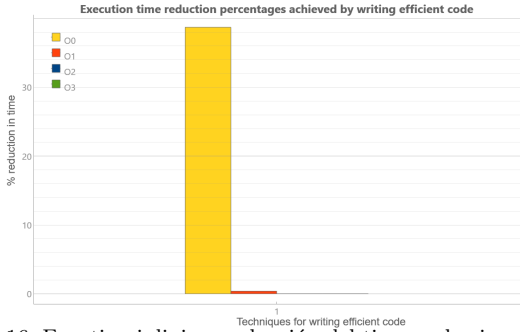


Fig. 16: Function inlining: reducción del tiempo de ejecución.

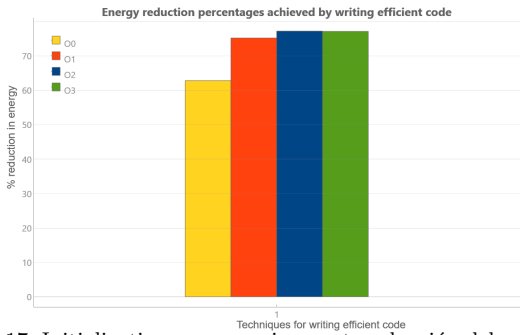


Fig. 17: Initialization versus assignment: reducción del consumo energético.

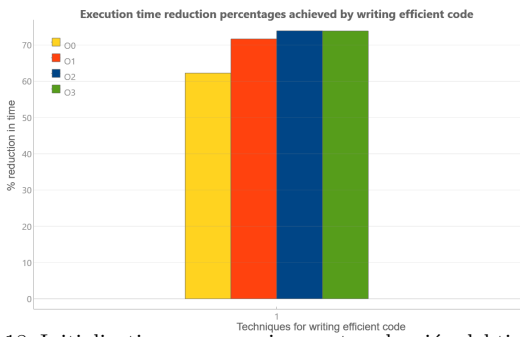


Fig. 18: Initialization versus assignment: reducción del tiempo de ejecución.

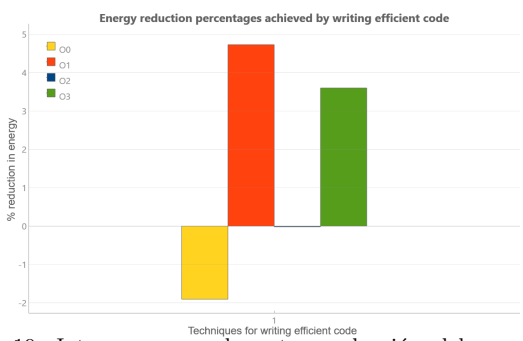


Fig. 19: Integer versus character: reducción del consumo energético.

anteriormente. Se observa una relación aproximadamente proporcional entre la reducción de tiempo de ejecución y la reducción del consumo energético a nivel de nodo, tal y como refleja el *trade-off* tiempo vs. energía (se puede ver en la Figura 3a, con un $R^2 = 0{,}993$). Esta proporcionalidad se puede observar también al analizar los rankings entre las distintas técnicas probadas: las 10 estrategias con mejores resultados en energía coinciden con las 10 mejores en tiempo (Figuras 4a y 4b), con técnicas como T10 y T21 apareciendo en las posiciones altas de ambas

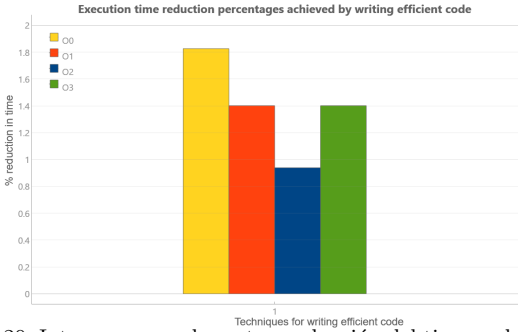


Fig. 20: Integer versus character: reducción del tiempo de ejecución.

métricas.

No obstante, la tendencia global no elimina la necesidad de analizar técnica a técnica y nivel a nivel. Las interacciones técnica–compilador son no monótonas: las mejoras pueden cambiar de magnitud o incluso de signo al pasar de `-O0` a `-O3`, lo que justifica realizar las comparaciones siempre dentro del mismo nivel de optimización. La potencia media ($\bar{P} = E/t$) actúa como indicador de desacople: cuando $\Delta\bar{P}$ muestra variaciones apreciables, señala que energía y tiempo no han reducido de forma estrictamente proporcional y que el caso requiere revisión cuidadosa (Figuras 3b y 3c). Además, ciertas técnicas presentan sensibilidad cualitativa al tamaño del problema, lo que impide extrapolar conclusiones sin considerar el parámetro que controla el *working set* y la presión sobre memoria. Estos resultados conectan de forma natural con los dos criterios internacionales de evaluación en supercomputación — rendimiento máximo (Top500) y eficiencia energética (Green500)— y sugieren que optimizaciones de código efectivas en latencia pueden trasladarse, en condiciones comparables, a reducciones proporcionales del consumo energético asociado.

Como ampliaciones futuras en esta linea, destacan principalmente dos: por un lado, comparar la medición a nivel de nodo con alternativas a nivel CPU (p. ej., contadores hardware, RAPL, estimadores, etc) para evaluar precisión, coste experimental y atribución, manteniendo la coherencia temporal entre potencia integrada y ejecución medida. Por otra parte, también se pueden realizar pruebas para analizar la interacción en combinaciones de técnicas respecto al consumo energético, tratando de buscar sinergias entre las distintas técnicas. En otro respecto,

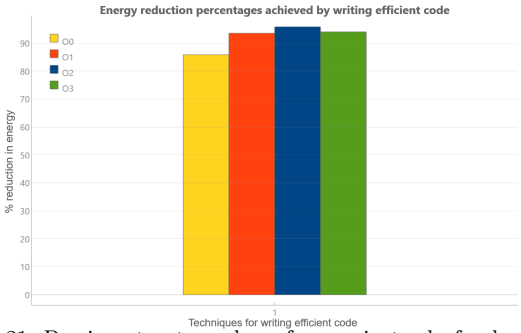


Fig. 21: Passing structures by reference instead of value: reducción del consumo energético.

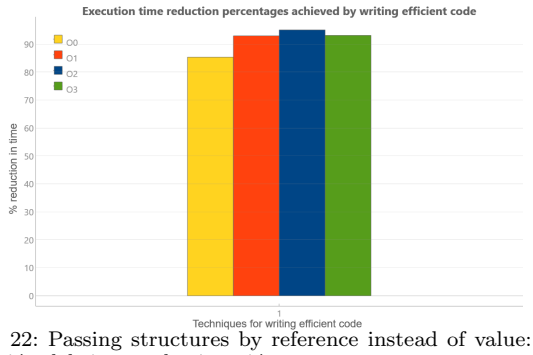


Fig. 22: Passing structures by reference instead of value: reducción del tiempo de ejecución.

pero también relacionado, se podrían trasladar estas técnicas a otros lenguajes de programación a fin de analizar el consumo, tanto a nivel del efecto de las técnicas como a nivel comparativo entre los distintos lenguajes analizados.

Agradecimientos

Esta investigación ha sido cofinanciada por el proyecto «EfiHPC: Entorno software para el desarrollo de códigos de programación eficientes en centros de supercomputación» con número de expediente PI-0173-C22, financiado con fondos del Mecanismo de Recuperación y Resiliencia de la Unión Europea - NextGenerationEU, a través del Programa Investigo (Decreto 137/2021, de 15 de diciembre), de la Consejería de Educación y Empleo de la Junta de Extremadura.

Referencias

[1] Fundación COMPUTAEX, "Lusitania ii," `https://computaex.es/lusitania/lusitania-ii/`, 2025, Consultado el 18 de diciembre de 2025.

[2] Arman Shehabi, Sarah J Smith, Eric Masanet, and Jonathan Koomey, "Data center growth in the united states: decoupling the demand for services from electricity use," *Environmental Research Letters*, vol. 13, no. 12, pp. 124030, dec 2018.

[3] C.-H. Hsu, W.-C. Feng, and J.S. Archuleta, "Towards efficient supercomputing: a quest for the right metric," in *19th IEEE International Parallel and Distributed Processing Symposium*, 2005, pp. 8 pp.–.

[4] Balaji Subramaniam, Winston Saunders, Thomas R. W. Scogland, and Wu-chun Feng, "Trends in energy-efficient computing: A perspective from the green500," in *Proceedings of the 4th International Green Computing Conference (IGCC 2013)*, Arlington, VA, USA, June 2013, Consultado el 29 de enero de 2026.

[5] TOP500.org, "Top500," `https://top500.org/`, 2025, Consultado el 16 de diciembre de 2025.

[6] TOP500.org, "Green500 — top500," `https://top500.org/lists/green500/`, 2025, Consultado el 16 de diciembre de 2025.

[7] Ministerio para la Transición Ecológica y el Reto Demográfico, *Guía para el cálculo de la huella de carbono y para la elaboración de un plan de mejora de una organización*, Ministerio para la Transición Ecológica y el Reto Demográfico, Nov. 2025, Coordinación y elaboración: Oficina Española de Cambio Climático. NIPO: 280-14-241-8. (En el documento figura © 2023.).

[8] Pablo Aparici Villalta, "Computational metrics analysis for climate models," 2025, Grau en Enginyeria Informàtica (Pla 2010).

[9] J. Corral García, "Paralelización automática y estrategias de desarrollo del código eficiente para aumentar el rendimiento en centros de supercomputación," http://dehesa.unex.es/handle/10662/12693, 2021.

[10] Javier Corral-García, José Luis González-Sánchez, and Miguel Ángel Pérez-Toledano, "Evaluation of strategies for the development of efficient code for raspberry pi devices," *Sensors (Switzerland)*, vol. 18, 11 2018.

[11] Javier Corral-García, Felipe Lemus-Prieto, José Luis González-Sánchez, and Miguel Ángel Pérez-Toledano, "Analysis of energy consumption and optimization techniques for writing energy-efficient code," *Electronics (Switzerland)*, vol. 8, 10 2019.

[12] Javier Corral-García, Felipe Lemus-Prieto, and Miguel Ángel Pérez-Toledano, "Efficient code development for improving execution performance in high-performance computing centers," *Journal of Supercomputing*, vol. 77, pp. 3261–3288, 4 2021.

[13] Chen Guo, Song Ci, Yanglin Zhou, and Yang Yang, "A survey of energy consumption measurement in embedded systems," *IEEE Access*, vol. 9, pp. 60516–60530, 2021, Publisher: IEEE.

[14] Kenneth O'Brien, Ilia Pietri, Ravi Reddy, Alexey Lastovetsky, and Rizos Sakellariou, "A survey of power and energy predictive models in hpc systems and applications," *ACM Computing Surveys*, vol. 50, no. 3, 2017.

[15] Claudio Vellage, "LaTeX tutorials," 2017.

[16] Optimum Energy Products Ltd., "WattsUP Pro Owner Manual," .

[17] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*, 2000, pp. 83–94.

[18] Mancebo, Javier et al, "Feetings: Framework for energy efficiency testing to improve environmental goal of the software," *Sustainable Computing: Informatics and Systems*, vol. 30, pp. 100558, 2021.

[19] Gutierrez, Maria et al, "Green-in machine learning at a glance," *Computer*, vol. 56, no. 6, pp. 35 – 43, 2023.

[20] J. Cancho Casado, "Framework avanzado para la optimización automática de código para la mejora del rendimiento del software en entornos de computación de alto rendimiento," 2025.

[21] M. Baños González, "Análisis y mejora del rendimiento en el desarrollo de software mediante el uso de estrategias eficientes de programación para infraestructuras de supercomputación," 2023.

[22] A. Mejías Velasco, "Aplicación de aprendizaje automático a la mejora del rendimiento de código en computación de altas prestaciones," 2025.

[23] W. Adzet Mauchline, "Aplicación de modelos de procesamiento de lenguaje natural a la optimización de código fuente en c/c++," 2025.

[24] Intel Corporation, "Running average power limit (rapl) energy reporting," `https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html`, 2025, Consultado el 16 de diciembre de 2025.

[25] IBM Corporation, "xcat — extreme cloud administration toolkit," `https://xcat.org/`, 2025, Consultado el 16 de diciembre de 2025.

[26] Rui Pereira et al, "Ranking programming languages by energy efficiency," *Science of Computer Programming*, vol. 205, pp. 102609, 5 2021.

[27] Alberto Gordillo, Coral Calero, Mª Ángeles Moraga, Félix García, João Paulo Fernandes, Rui Abreu, and João Saraiva, "Programming languages ranking based on energy measurements," *Software Quality Journal*, vol. 32, no. 4, pp. 1539–1580, Dec 2024.

[28] Kurt Guntheroth, *Optimized C++: proven techniques for heightened performance*, "O'Reilly Media, Inc.", 2016.

[29] Ministerio para la Transición Ecológica y el Reto Demográfico (MITECO), *Huella de carbono 2023: Evolución 2019–2023*, Ministerio para la Transición Ecológica y el Reto Demográfico, May 2025, Mayo 2025. Coordinación y elaboración: Oficina Española de Cambio Climático.

[30] Antoine Petitet, R. C. Whaley, Jack Dongarra, A. Cleary, and Piotr Luszczek, "Hpl – a portable implementation of the high-performance linpack benchmark for distributed-memory computers," `https://www.netlib.org/benchmark/hpl/`, 2018, Última revisión: 2 de diciembre de 2018; versión 2.3.

[31] TOP500 Project, "The linpack benchmark," `https://top500.org/project/linpack/`, 2026, Página del proyecto LINPACK utilizada para el ranking TOP500.

Diseño de un Emulador para RISC-V Implementado en Metalenguaje

Elías Coronado Salazar[1], Juan Pablo De la Cerda González [1], Edwin Christian Becerra Álvarez[2], Juan José Raygoza Panduro [2], Dionicio Yukitaro Kono Garcia [3] y Jorge Rivera Dominguez [4]

Resumen— **El desarrollo y validación de sistemas basados en la arquitectura RISC-V puede depender tanto de hardware físico como de emuladores, lo cual limita dicha tarea. Asimismo, los emuladores disponibles actualmente se enfocan principalmente en configuraciones genéricas de la arquitectura, ofreciendo soporte limitado e incluso inexistente para extensiones y modificaciones personalizadas. Debido a ello, y como parte del desarrollo de un procesador RISC-V diseñado a medida, surge la necesidad de contar con una herramienta flexible y accesible para la verificación de instrucciones y el análisis del comportamiento del microprocesador, incluso antes de disponer de una implementación física en hardware. Por lo cual en este trabajo se presenta el diseño e implementación de un emulador de RISC-V (RV32I). Para el diseño del emulador se utiliza el metalenguaje Leo, demostrando que es posible crear emuladores con metalenguajes. Además, se compara su funcionamiento con el emulador QEMU, verificando su correcto desempeño mediante 145 pruebas, las cuales supera de manera adecuada. También se implementan cuatro aplicaciones: (1) promediador de ocho números, (2) codificador César, (3) filtro FIR y (4) filtro IIR; en los cuales se comprueba su funcionamiento comparando la respuesta del emulador implementado con QEMU y Excel, donde se obtiene un error máximo de 8 % para el FIR y 4,5 % para el IIR. Finalmente, es importante señalar que los resultados obtenidos en el emulador son similares a los obtenidos tanto con QEMU como Excel.**

Palabras clave— **Metalenguaje, Emulador RISC-V, Lenguaje Ensamblador**

I. Introducción

CONFORME ha evolucionado la computación, el diseño de procesadores ha experimentado diversos paradigmas para adaptase a las nuevas necesidades de procesamiento. Cabe señalar que dentro de estos cambios se tienen dos arquitecturas que han marcado la tendencia, la arquitectura CISC (del inglés, *Complex Ins-truction Set Computer*) y la RISC (del inglés, *Reduced Instruction Set Computer*).

La arquitectura CISC surge en la década de los 60's y 70's, donde su enfoque se motiva por el deseo de simplificar los compiladores y la búsqueda de una mejora en las prestaciones [1].

En dicha arquitectura CISC se buscó generar una instrucción máquina para cada sentencia de lenguaje de alto nivel, lo cual ayuda a generar códigos más compactos, con la intención de producir programas más pequeños y rápidos. Esto presenta una ventaja de tener un uso más eficiente de la memoria de programa, al tener menor cantidad de instrucciones, y también se tiene un ahorro en la cantidad de bytes en el tamaño de las instrucciones, con lo que se puede aprovechar de otra manera los bits de palabra.

Sin embargo, actualmente ya no es una limitación el tamaño de la memoria, por lo que esta ventaja ya no es tan significativa. Por otro lado, una desventaja de esta arquitectura es que al tener instrucciones más complejas, la dificultad de su diseño en hardware aumenta considerablemente, además, de necesitar más espacio de silicio y transistores. Por lo cual hay un mayor requerimiento de corriente, lo cual implica mayor consumo de potencia, y al tener más etapas interconectadas necesita de un mayor tiempo de ejecución, con lo que puede ocurrir que bajo ciertas circunstancias una instrucción requiera una mayor cantidad de ciclos de reloj lo que se debe tener en cuenta a la hora de utilizarlas.

Por otro lado, la arquitectura RISC surge en la década de los 80's como resultado del estudio de la creciente complejidad de los procesadores CISC [2], [3].

Los procesadores CISC contaban con cientos de instrucciones, de las cuales algunas de ellas rara vez se utilizaban. Esto condujo a un cambio en el paradigma de diseño, el cual consistía en desarrollar una arquitectura con un conjunto menor de instrucciones, las cuales deberían ser simples, eficientes y rápidas. Las características de estas instrucciones son: conjunto de instrucciones reducido y uniforme, solo las instrucciones de carga y almacenamiento acceden a la memoria, todas las demás operaciones se realizan entre registros internos, pipelining eficiente, compiladores optimizados.

Todo procesador RISC debe ser capaz de realizar todas las operaciones que realiza un procesador CISC. Sin embargo, para realizar operaciones más complejas realiza varias operaciones sencillas de manera secuencial.

Existen varios procesadores diseñados bajo la filosofía RISC, entre ellos destacan: MIPS (del inglés, *Microprocessor without Interlocked Pipeline Stages*), ARM (del inglés, *Advanced RISC Machines*, RISC-V.

[1]Dpto. de Ingeniería Electro-Fotónica, Universidad de Guadalajara, e-mail: {elias.coronado4517, juan.delacerda1742}@alumnos.udg.mx

[2]Dpto. de Ingeniería Electro-Fotónica, Universidad de Guadalajara, e-mail: {edwin.becerra, juan.rpanduro}@academicos.udg.mx

[3]Dpto. de Ingeniería Electro-Fotónica, Universidad de Guadalajara, e-mail: dionicio.kono@alumnos.udg.mx

[4]CINVESTAV del IPN, Campus Guadalajara, e-mail: jorge.rivera@cinvestav.mx

II. RISC-V

RISC-V es una arquitectura de conjunto de instrucciones ISA (del inglés, *Instruction Set Architecture*) de código abierto, ha estado causando sensación en el mundo de la arquitectura informática. La ISA es la especificación técnica que define la manera en que un procesador funciona desde el punto de vista del software, es decir, es el puente entre el hardware y el software, siendo el lenguaje básico que entiende un procesador.

Una ISA abierta es una arquitectura de conjunto de instrucciones cuyo diseño y especificaciones están disponibles públicamente y pueden ser usadas, implementadas, modificadas y compartidas libremente, sin necesidad de pagar licencias ni tener permiso de una empresa propietaria. Esto ha permitido una gran aceptación en la comunidad de desarrolladores y la industria ya que su futuro no está sujeto al destino o capricho de alguna corporación, cuestión que ha dañado gravemente a muchas ISAs en el pasado [4].

RISC-V se diseña desde el principio (por la Universidad de California, Berkeley, en 2010) como una arquitectura de conjunto de instrucciones ISA modular, extensible y abierta [5].

En lugar de ser una ISA fija como x86 o ARM, RISC-V está estructurada en módulos independientes o extensiones, que se pueden combinar según los requerimientos del sistema, es decir, adopta un enfoque modular.

El núcleo base es obligatorio y garantiza un mínimo funcional común en todas las implementaciones, permitiendo la incorporación de extensiones opcionales según las necesidades del procesador. Además, es importante señalar que existen varias versiones del conjunto base dependiendo del tamaño de palabra del procesador: RV32I, RV64I o RV128I, donde "RV" significa RISC-V, el número indica el tamaño de palabra de la arquitectura en bits y la "I" representa el conjunto de instrucciones enteras.

La Tabla I muestra los módulos que se pueden incluir en el procesador. Las instrucciones de punto flotante de simple precisión se designan como "F" y añade instrucciones de punto flotante con precisión simple (32 bits) que cumplen el estándar IEEE 754, registros flotantes y control de estados que guarda la configuración y el estado del sistema de punto flotante [6], [7].

Las instrucciones vectoriales (*Vectorial Extension*) se designan como "V" y agregan soporte de procesamiento vectorial de datos, lo cual permite procesar múltiples datos en paralelo con una sola instrucción, cabe señalar que es muy usado en IA, gráficos y cálculos científicos [7].

Tabla I: Módulos RISC-V.

Conjunto	Instrucciones	Descripción
RV32I	40	Base entera (obligatoria).
M	8	Multiplicación y división.
A	11	Instrucciones atómicas.
F	26	Flotante simple precisión.
D	26	Doble precisión.
Q	27	Precisión cuádruple.
C	30	Instrucciones comprimidas.
B	50	Op. manipulación de bits.
V	200	Instrucciones cectoriales.

El conjunto base RV32I es el módulo principal sobre el cual trabajan los demás módulos, por lo que este debe contener todos los bloques funcionales mínimos de una CPU debe tener. Este conjunto de elementos conectados forma un procesador funcional, cuando se requieren agregar módulos al procesador, se hace necesario realizar ajustes y ampliaciones en componentes del hardware, pero manteniendo siempre la compatibilidad con RV32I [8].

En RV32I existen varios tipos de registros, como los de tipo entero, los cuales tienen una función específica. El procesador cuenta con 32 registros de 32 bits, enumerados desde 0x00 hasta 0x31, cada uno de ellos tiene un nombre ABI (del inglés, *Application Binary Interface*). Estos nombres son alias simbólicos que se usan en lugar de los nombres numéricos (x0, x1, ..., x31) para identificar los registros según su función o propósito dentro de una llamada a función o convención de programación.

Como se mencionó anteriormente en la arquitectura RISC-V, en lugar de tener un conjunto de instrucciones gigante como en x86, se divide en módulos o extensiones que se pueden combinar según las necesidades del procesador. A su vez, cada módulo divide su conjunto de instrucciones en tipos de instrucciones mediante su funcionalidad y formato. La Tabla II muestra los tipos de instrucciones que existen en RISC-V.

A. Emuladores y Metalenguaje

La teoría de la emulación combina ideas de arquitectura de computadoras, lenguajes de programación y principios de la máquina de Turing [6], [5], [7]. Al entender esta teoría es fundamental para cualquier desarrollador que desee comprender cómo funcionan internamente los procesadores, los lenguajes y las máquinas de virtuales [6], [7].

Gracias a los emuladores es posible ejecutar programas antiguos, correr software para arquitecturas diferentes a la que tiene el hardware y construir emuladores experimentales y educativos.

Un emulador es capaz de imitar el comportamiento interno de otros sistemas, reproduciendo sus instrucciones, memoria, entradas, salidas, interrupciones y arquitectura [6].

Esto se logra mediante procesos de interpretación, traducción, simulación del hardware y las reglas formales que permiten reproducir cada instrucción del sistema original [6] y [7].

Tabla II: Tipos de instrucciones en RISC-V.

Tipo	Nombre	Descripción
R-Type	Register-Type	Op. entre registros.
I-Type	Immediate-Type	Uso de inmediato.
S-Type	Store-Type	Instr. de almacenamiento.
B-Type	Branch-Type	Instr. de salto condicional.
U-Type	Upper-Type	Inmediato de 20 bits altos.
J-Type	Jump-Type	Inst. de salto incondicional.

La tesis de Church-Turing: "Todo sistema computacional suficientemente complejo puede simular a otro" [5], [8]. El emulador crea un "mapa de memoria" donde almacena: código, datos, *heap*, *stack* [6], [7]. Por otro lado, los emuladores más avanzados imitan también: cachés, ALU, predicción de saltos y microinstrucciones [6].

Actualmente existen diversas herramientas de software que permiten emular la arquitectura RISC-V para el desarrollo de software, la verificación de arquitecturas y la enseñanza de sistemas computacionales. Estas herramientas reproducen el comportamiento de un procesador RISC-V sin necesidad de contar con hardware físico, lo que facilita el desarrollo y prueba de aplicaciones y sistemas operativos.

Uno de los emuladores más utilizados es QEMU, una plataforma de emulación de código abierto que soporta múltiples arquitecturas de procesador, incluida RISC-V. Este emulador permite reproducir el comportamiento de sistemas completos, incluyendo memoria, dispositivos de entrada y salida y controladores de interrupciones, lo que posibilita ejecutar sistemas operativos como Linux en un entorno virtual. Además, utiliza técnicas de traducción binaria dinámica para mejorar el rendimiento de la ejecución del código emulado [7].

Otro emulador ampliamente utilizado es Spike, considerado el emulador de referencia para la arquitectura RISC-V. Este implementa un modelo funcional del conjunto de instrucciones ISA y se utiliza principalmente para validar implementaciones de procesadores y verificar el comportamiento de las instrucciones definidas en la especificación oficial de la arquitectura RISC-V [5].

En el ámbito educativo, una de las herramientas más populares es RARS, un emulador que permite ensamblar y ejecutar programas escritos en lenguaje ensamblador para la arquitectura RISC-V. Esta herramienta proporciona una interfaz gráfica que facilita la visualización de registros, memoria y ejecución paso a paso de las instrucciones, lo que la convierte en una herramienta útil para el aprendizaje de arquitectura de computadoras y programación en ensamblador [8].

Renode es otro emulador relevante, orientado al desarrollo y prueba de sistemas embebidos. Permite modelar plataformas completas basadas en RISC-V, incluyendo procesadores, memoria y diversos periféricos. Esto facilita el desarrollo y la validación de firmware y software embebido antes de disponer de hardware físico, lo cual es particularmente útil en procesos de desarrollo y pruebas automatizadas [9].

Finalmente, OVPsim es un simulador y emulador de alto rendimiento diseñado para la simulación rápida de procesadores y plataformas basadas en diferentes arquitecturas, incluyendo RISC-V. Esta herramienta es utilizada principalmente en entornos de investigación e industria debido a su capacidad para ejecutar simulaciones de manera eficiente y analizar el comportamiento del sistema bajo distintas configuraciones de hardware [10].

La Tabla III muestra el estado del arte para los emuladores más utilizados para emular la arquitectura RISC-V.

El emulador tradicionalmente se implementa usando un lenguaje de alto nivel, sin embargo, otra alternativa es usar un metalenguaje. Para la implementación del emulador se utiliza el metalenguaje Leo, el cual se desarrolla en el Laboratorio de Sistemas Embebidos de la Universidad de Guadalajara [11].

Un metalenguaje es un lenguaje de programación que se utiliza para describir otro lenguaje de programación. El metalenguaje Leo es flexible, ya que cuenta con más de 100 palabras reservadas. Esto le da la capacidad de poder, incluso, crear un lenguaje desde cero completamente. El propósito principal del metalenguaje es brindar una forma de comunicación precisa y concisa para describir las características, estructuras y comportamientos de los lenguajes de programación u otros sistemas formales [12], [13].

III. Implementación

Se implementa el módulo RV32I debido a que constituye la base para los procesadores RISC-V de 32 bits, donde dicho procesador cuenta con un conjunto de 37 instrucciones y un ancho de palabra de 32 bits. La implementación del emulador se realiza en tres etapas: asignación de memoria, implementación de registros y finalmente, implementación de instrucciones.

Mediante el metalenguaje Leo se establece una memoria de 100 celdas, donde cada celda ocupa un espacio de una palabra doble de 32 bits (para una arquitectura de 64 bits). En la Figura 1 se muestra dicho proceso de asignación de memoria.

Tabla III: Emuladores para RISC-V.

Emulador	Tipo de sim.	Uso Principal
QEMU	Emulador de sistema completo	Desarrollo de sistemas operativos y firmware.
Spike	Sim. de ISA	Validación de arquitecturas y referencia de ISA.
RARS	Sim. educativo	Enseñanza de ensamblador RISC-V.
Renode	Sim. de plataformas	Desarrollo y pruebas de firmware.
OVPsim	Sim. de alto rendimiento	Investigación y desarrollo industrial.

```
1 #-----------Memory allocation------------
2 100 integers
3 allocate
```

Fig. 1: Código para la implementación de la memoria.

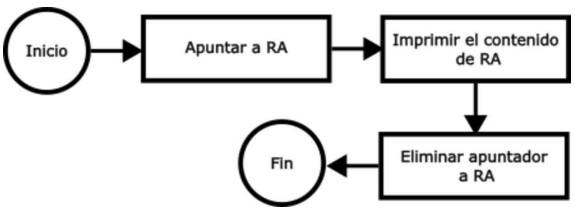


Fig. 2: Diagrama de flujo para la implementación de la palabra `RA`.

El RV32I cuenta con 32 registros de tipo entero, donde cada uno de estos se implementa como una variable inicializada en cero. Además, para los registros se implementaron dos palabras permitiendo acceder y mostrar el contenido de cada uno. En la Figura 4 se muestra la implementación de dichas palabras para el registro `RA`.

La palabra `RA@` regresa el valor del registro `RA`. Primeramente, se almacena la cadena de texto *RA: "* en el apuntador de memoria, PS (del inglés, *Pointer Stack*). A continuación, se imprime el contenido del PS en la consola y se elimina su contenido. Ahora, se apunta a la dirección del registro `RA` y se almacena su contenido en la pila de datos DS (del inglés, *Data Stack*), se elimina el apuntador a `RA`. Una vez que se tiene el contenido del registro, se configura el formato de salida deseado, en este caso hexadecimal con ocho dígitos. Finalmente se imprime el contenido del DS en la consola y se ejecuta un retorno de carro. El diagrama de flujo para la implementación de esta palabra se muestra en la Figura 3.

Por otra parte, la palabra `RA.` imprime su contenido en la terminal. Primeramente, se apunta a la dirección del registro `RA`. A continuación, se imprime en consola el contenido del registro y finalmente se elimina el apuntador al registro. El diagrama de flujo para la palabra `RA.` se muestra en la Figura 2.

Para la implementación las instrucciones se agruparon por tipo de instrucción, debido a que su diseño es similar. En la Figura 7 se muestra la implementación de la instrucción ADDI, la cual realiza una suma de un registro con un dato inmediato, y guarda el resultado en un registro destino.

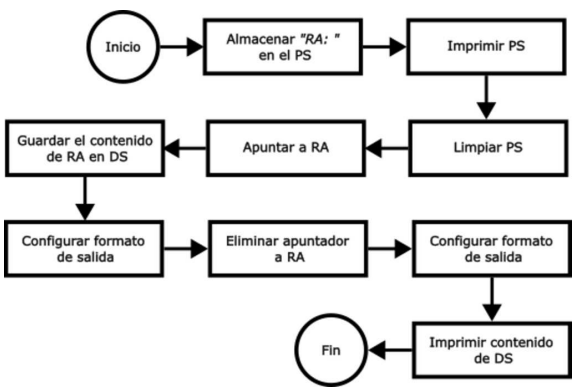


Fig. 3: Diagrama de flujo para la implementación de la palabra `RA@`

```
1  #-----------Register RA------------
2  : RA.
3      "RA: " $. free
4      RA @ pdrop
5      16 base
6      1 zeros
7      9 digits
8      . cr
9  end
10 # ----------------------------------
11 : RA@
12 RA @ pdrop
13 end
```

Fig. 4: Código para la implementación de las palabras `RA.` y `RA@`

La Figura 5 muestra la instrucción `ADDI`, aquí se observa cómo se van leyendo las palabras que contiene la instrucción.

La línea 2 define la palabra `ADDI`, mientras que en la línea 3 la palabra `parameter (,)`, detiene la ejecución de la palabra `ADDI` y regresa el control al código que la mando llamar, permitiendo ejecutar la siguiente instrucción que se encuentre, para de esta manera obtener los parámetros de la instrucción.

Para ello primero lee la siguiente palabra en la instrucción, `RD` y regresa a la ejecución de la palabra donde `call` apunta a la dirección del registro leído, `RD`, ingresando la en el PS. A continuación, se detiene nuevamente la ejecución de la instrucción y se regresa nuevamente el control al flujo principal donde se ejecuta la siguiente instrucción, `R1`, y nuevamente con `call` se apunta a la dirección del registro `R1` en el PS y con `@` se almacena el contenido del registro en el DS. Nuevamente con `,` se regresa el control al flujo principal y se ejecuta la siguiente instrucción, la cual es almacenar el valor constante en el DS. Ahora se elimina el apuntador a `R1` ya que se almacena su contenido en DS, línea 4. Ahora se ejecuta la operación de suma `+`, lo cual realiza la suma de las dos celdas superiores en DS. Con la operación realizada se guarda el resultado en el registro que apunta PS, que es `RD` y se elimina el apuntador a `RD` dejando el estado de las pilas como estaban antes de ejecutar la instrucción ADDI. El contenido de las pilas conforme se ejecutan las instrucciones se muestra en la Figura 6.

Las demás instrucciones lógico-aritméticas tienen la misma estructura que la instrucción `ADDI`, solo se cambia la operación que se tenga que realizar. Para `SLTI`, `SLTUI`, `ANDI`, `ORI`, `XORI`, `SLLI` y `SRLI` se sustituye por `s<`, `<`, `and`, `or`, `xor`, $\ll$ y $\gg$, respectivamente.

Las instrucciones de carga permiten leer datos desde la memoria. En la Figura 9 se muestra la imple-

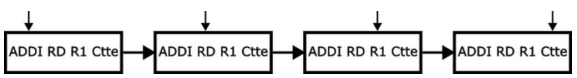


Fig. 5: Instrucción ADDI.

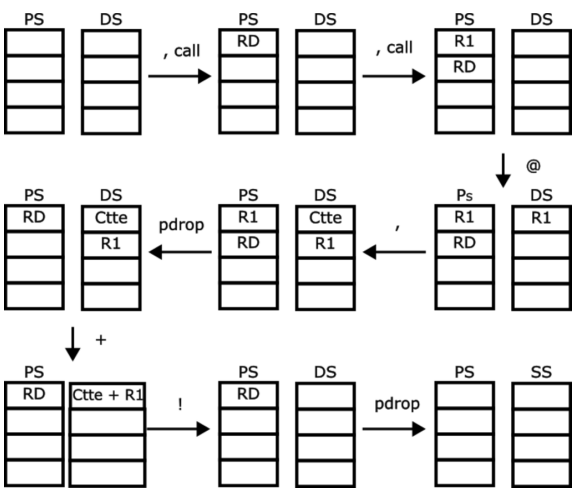


Fig. 6: Estado de las pilas `PS` y `DS` durante la ejecución de la instrucción `ADDI`.

```
#-----------Instruction ADDI------------
: ADDI
    , call , call @ ,
    pdrop
    + ! pdrop
end
```

Fig. 7: Código para la implementación de la instrucción `ADDI`.

mentación de la instrucción `LB`, la cual carga un byte de datos en el registro destino `RD`, extendiendo el signo.

Cuando el número a guardar es positivo, en el registro destino se guardan los 8 bits de información y los 24 restantes se completan con ceros, mientras que cuando el número a cargar es negativo, el registro destino debe almacenar los 8 bits de información y los 24 restantes se deben completar con unos. El diagrama de flujo se muestra en la Figura 8.

La palabra `LBU` es similar a `LB`, solo que en esta no importa el signo, por lo que se omite la verificación del signo y se guarda en `R1` el contenido de DS. Las palabras `LH` y `LW` son similares a `LB`, solo que en estas cambia la cantidad de desplazamientos, para `LH` se utilizan 16 desplazamientos y para `LW` no existen desplazamientos.

Las instrucciones implementadas se muestran en la Tabla IV, donde se puede observar que la instrucción `LB` contiene el mayor número de palabras para su implementación, mientras que `LUI` es la instrucción que menos palabras requiere en su definición.

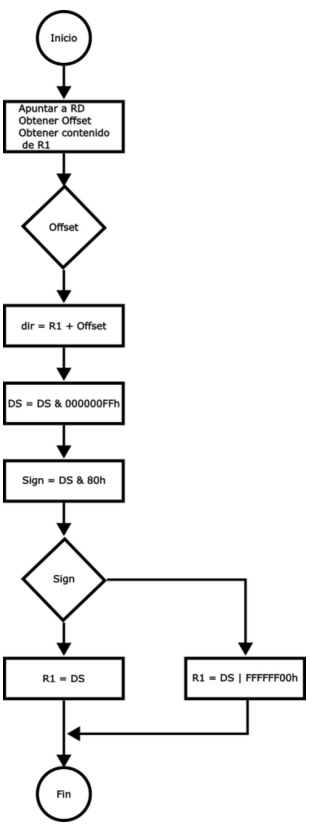


Fig. 8: Diagrama de flujo de la implementación de la instrucción `LB`.

```
#-----------Instruction LB------------
: LB
    , call , , call @ pdrop pswap
        +
    if dup 0 =
        { drop @ }
    else
        {
                dup symbols shift @ swap
                -1 * symbols shift
        }
                24 << 24 >> pswap
                dup 80h and
                if { ffffff00h or }
                ! pdrop
end
```

Fig. 9: Código para la implementación de la instrucción `LB`

.

Tabla IV: Instrucciones implementadas en el Emulador.

Instrucción	Palabras	Instrucción	Palabras
ADDI	10	AND	13
SLTI	10	SLL	13
SLTUI	10	SRL	13
XORI	10	SRA	31
ORI	10	LB	44
ANDI	10	LBU	36
SLLI	10	LH	44
SRLI	10	LHU	36
SRAI	28	LW	32
ADD	13	SB	42
SUB	13	SH	42
SLT	13	SW	30
SLTU	13	LUI	8
XOR	13	AUIPC	10
OR	13		

IV. Resultados Experimentales

Para las pruebas se creó un archivo nuevo llamado *riscv_test.leo* en el que se llevaron a cabo las verificaciones para cada instrucción de manera individual, donde se incluye el archivo *riscv.leo* que contiene la definición de las instrucciones del RV32I.

Aquí, se crearon dos variables que actúan como contadores para cuantificar la cantidad de pruebas que pasaban *passes* y las que no lo hacían *fails*. Cada palabra se probó en cinco ocasiones tratando de cubrir diferentes escenarios para ver cómo se comportaba la palabra. El resultado de las pruebas se muestra en la Figura 10. Se observa que todas las instrucciones pasaron las pruebas de manera individual. En total se pasaron 145 pruebas y no hubo pruebas fallidas.

Para validar el funcionamiento del emulador se seleccionaron algunas pruebas. Para la primera se im-

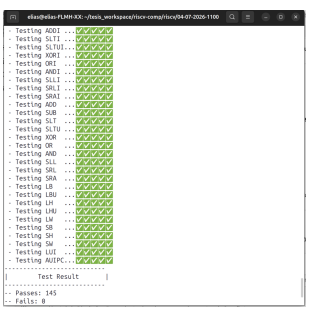


Fig. 10: Resultados de las pruebas para las instrucciones individuales.

plementó un algoritmo que calcula el promedio de ocho números.Para ello se seleccionaron los siguientes números: 10, 20, 30, 40, 50, 60, 70 y 80. Dichos valores se almacenan en memoria utilizando metalenguaje Leo. La Figura 11 muestra el algoritmo del promedio de 8 números. El resultado de implementar este algoritmo en el emulador en EMOS se observa en la Figura 12, mientras que el emulador de QEMU se observa en la Figura 13. Se puede apreciar que ambos emuladores llegan al mismo resultado, 45, que es el promedio de los ocho números antes mencionados.

Para la segunda prueba se implementa un codificador Cesar. Dicha codificación es un método de cifrado muy antiguo que se usaba para mandar mensajes militares secretos. Este es un cifrado por sustitución donde cada letra del mensaje original se reemplaza por una letra que se encuentra en un número fijo de posiciones más adelante en la tabla ASCII. Este método utiliza una *llave* la cual es el número de posiciones que se desplaza el alfabeto. En la Figura 14 se muestra el algoritmo.

En las Figura 15 y 16 se muestra el resultado de la implementación del codificador Cesar usando una llave de 3 para el texto *HOLA-MUNDO* usando el emulador en EMOS y en QEMU respectivamente. Se puede ver que el resultado en ambos emuladores es *KROD0PXQGR*.

En la tercera prueba se implementa un filtro FIR promediador de cuatro puntos, para una señal de entrada de 50 valores. La ecuación para implementar el filtro se muestra en la Ecuación 1. El resultado de la implementación en EMOS se muestra en la Figura 17. En la Figura 18 se muestran la señal de entrada como la salida filtrada. La entrada es una señal rampa ligeramente afectada por ruido y se puede ver que el filtro atenúa significativamente el ruido pre-

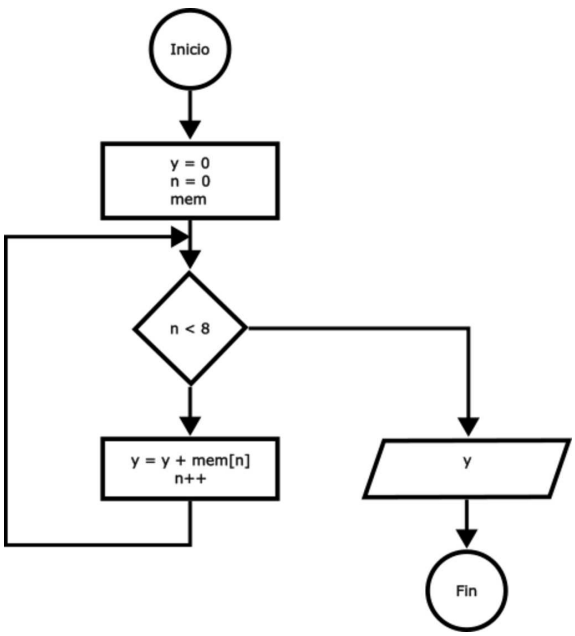


Fig. 11: Algoritmo del promedio de ocho números.

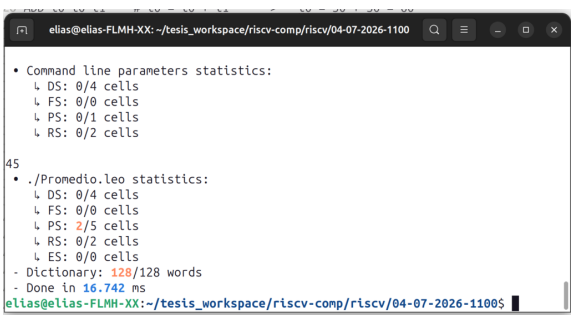


Fig. 12: Resultado del promedio en EMOS.

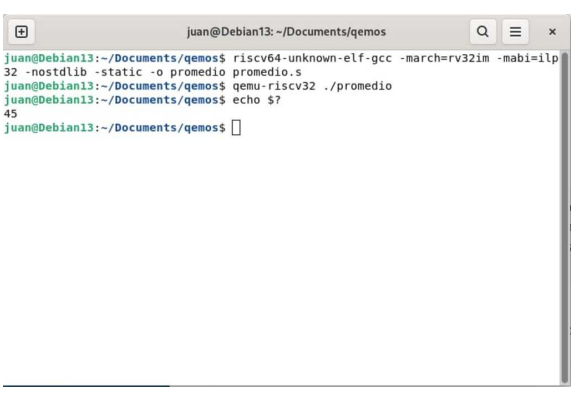


Fig. 13: Resultado del promedio en QEMU.

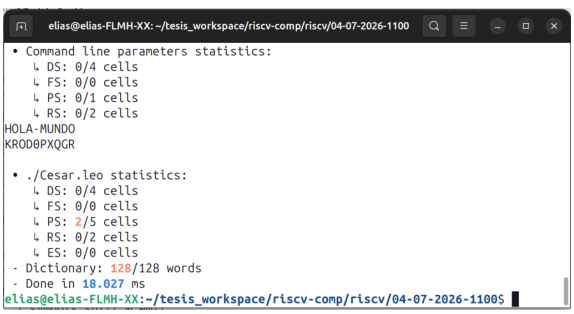


Fig. 14: Algoritmo del codificador Cesar.



Fig. 15: Resultado del codificador Cesar en EMOS.

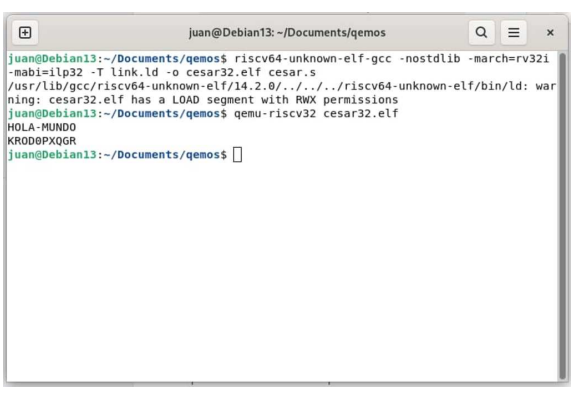


Fig. 16: Resultado del codificador Cesar en EMOS.

sente en la señal de entrada. Además, se compara la salida del filtro implementado en el emulador con el filtro implementado en Excel, ver Figura 19.

Se observa que existe una pequeña discrepancia en las respuestas, esto es a causa de que en Excel se utiliza coma flotante mientras que en el emulador solo se utilizan enteros, los valores fraccionarios se descartan. El error entre la respuesta del filtro en el emulador de EMOS y el emulador QEMU se muestra en la Figura 20, se puede observar que el error máximo es de aproximadamente 8 % y se presenta al principio del filtrado.

$$y[n] = \frac{1}{4}\sum_{k=0}^{3} x[n-k] \tag{1}$$

Finalmente se implementa un filtro IIR EMA (del inglés, *nential Moving Average*). Se selecciona este filtro porque tiene un comportamiento parecido al FIR implementado anteriormente, aunque no es igual. La ecuación del filtro IIR EMA se muestra en 2.

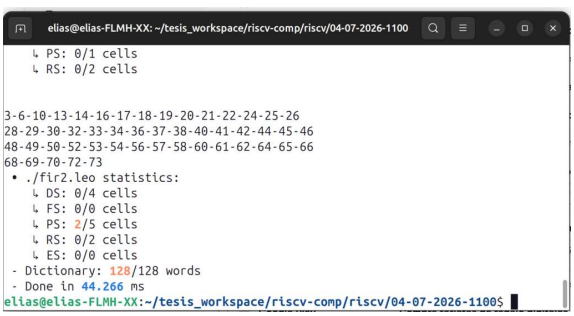


Fig. 17: Resultado del filtro FIR.

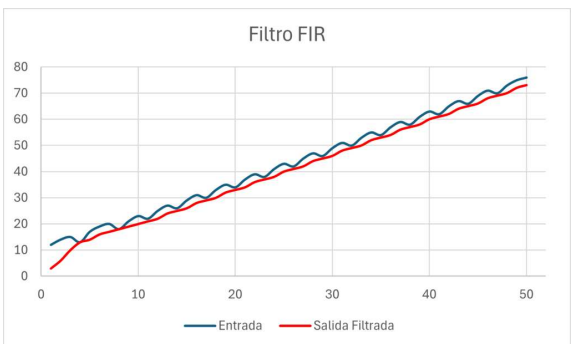


Fig. 18: Efecto del filtrado sobre la señal de entrada.

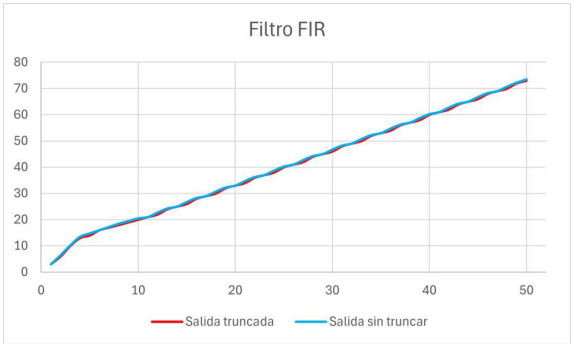


Fig. 19: Filtrado en EMOS vs filtrado en Excel.

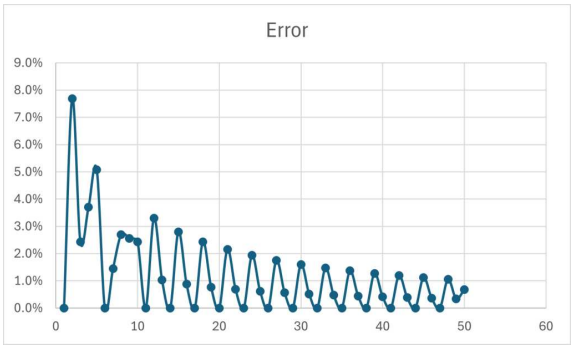


Fig. 20: Porcentaje de error entre el valor obtenido en EMOS y QEMU para el filtro FIR.

$$y[n] = 0{,}25\,x[n] + 0{,}75\,y[n-1] \tag{2}$$

La Figura 21 muestra la salida del emulador. En la Figura 22 se muestran las señales de entrada y salida del filtro. En este caso el filtro también atenúa significativamente el ruido presente en la señal de entrada, como en el caso del filtro FIR.

La Figura 23 muestra la comparación de la salida del filtro implementado en el emulador contra el filtro implementado en Excel. La diferencia observable es debida al truncamiento de los valores para el emulador, ya que solo maneja números enteros. El error entre la respuesta del filtro en el emulador de EMOS y el emulador QEMU se muestra en la Figura 24, se puede observar que el error máximo es de aproximadamente 4,5 % y se presenta al principio del filtrado.

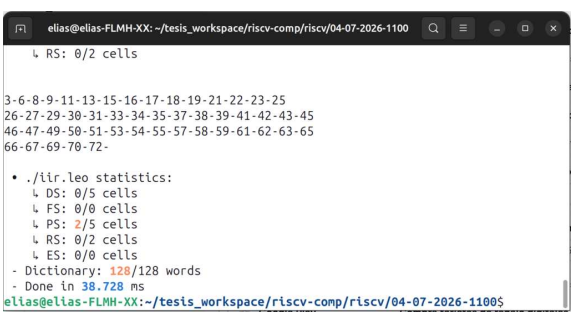


Fig. 21: Resultado del filtro IIR.

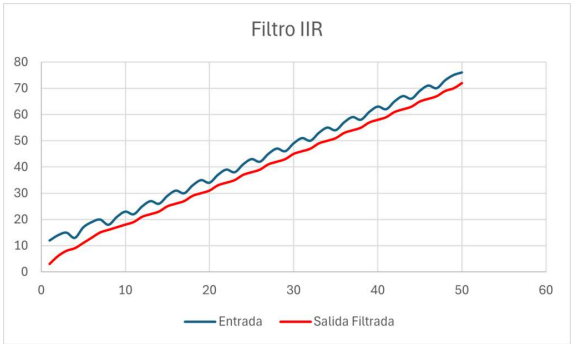


Fig. 22: Efecto del filtrado sobre la señal de entrada.

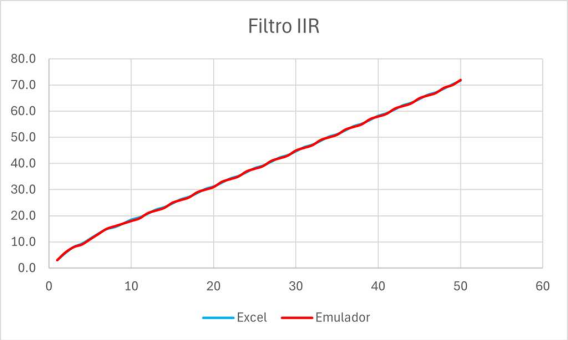


Fig. 23: Filtrado en EMOS vs filtrado en Excel.

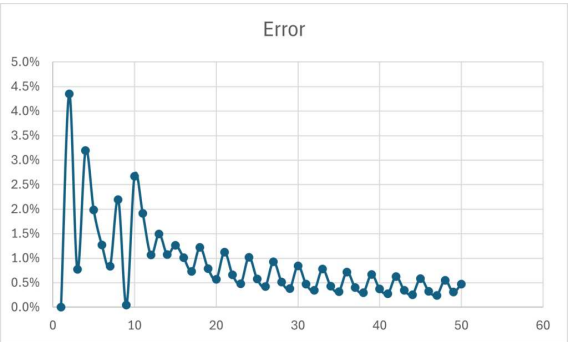


Fig. 24: Porcentaje de error entre el valor obtenido en EMOS y QEMU para el filtro IIR.

V. Conclusiones

En este trabajo se implementan las instrucciones con las que cuenta el RV32I. Se implementaron usando el metalenguaje Leo y se comprobó su comportamiento obteniendo resultados satisfactorios.

Los registros se declararon como variables de 32 bits y se crearon dos palabras para cada registro para poder mostrar el dato en consola o almacenarlo en la cima de la pila, estos no son parte del set de instrucciones del RV32I pero ayudan a acceder al contenido de los registros de manera rápida.

Las instrucciones aritméticas y lógicas junto con las instrucciones tipo R fueron las que presentaron menor dificultad en su implementación, a excepción de la instrucción de corrimiento aritmético en la cual fue necesario hacer uso de máscaras y operaciones or para lograr la extensión de signo. Por otra parte, las instrucciones de carga y almacenamiento requirieron de operaciones de corrimiento ya que los registros son de 32 bits y cuando leen datos de memoria almacenan 32 bits, para solo leer o escribir 8 o 16 bits fue necesario utilizar corrimientos para eliminar los bits no deseados.

En cuanto a las pruebas individuales, se realizaron cinco pruebas por cada instrucción, las cuales se seleccionaron de manera que se pudiera probar la mayor cantidad de escenarios posibles con la menor cantidad de números. Por ejemplo, para el caso de la suma se realizaron sumas con números negativos, positivos y con el cero, las cuales fueron satisfactorias.

Las cuatro aplicaciones implementadas también fueron favorables, se compararon los resultados obtenidos del emulador implementado con el emulador QEMU, dando los mismos resultados. Con respecto a las simulaciones de los filtros, estas se compararon con los resultados obtenidos de los filtros implementados en Excel, obteniéndose resultados aceptables. Los resultados del emulador se contrastaron con los obtenidos en Excel truncando y sin truncar los decimales, donde se observó que el error en cada medición era menor al 0.6 %, para el caso de los datos utilizados en la prueba.

Además, se analizó la viabilidad de incorporar mecanismos de prueba automatizados usando el metalenguaje Leo, lo que le permite detectar errores y hacer mejoras en el código.

Agradecimientos

Expresamos el más sincero agradecimiento a la Secretaría de Ciencia, Humanidades, Tecnología e Innovación (SECIHTI) y a la Universidad de Guadalajara por facilitar la investigación y desarrollo para el presente trabajo.

Referencias

[1] A. Clements, *Principles of Computer Hardware, 5th ed.*, Oxford University Press, Oxford, U.K., 2020.
[2] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach, 6th ed.*, Morgan Kaufmann, Burlington, MA, USA, 2019.
[3] D. A. Patterson and C. H. Séquin, "Design and implementation of risc i," 1981.
[4] A. Waterman and K. Asanovic, *The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA*, University of California, Berkeley, CA, USA, 2019.
[5] A. Waterman and K. Asanovic, *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.2*, EECS Department, University of California, Berkeley, CA, USA, 2017.
[6] A. Tanenbaum and T. Austin, *Structured Computer Organization, 6th ed.*, Pearson, México, 2013.
[7] F. Bellard, "Qemu, a fast and portable dynamic translator," 2005, USENIX Annual Technical Conference (FREENIX Track), Anaheim, CA, USA, pp. 41–46.
[8] The QEMU Project, "Qemu documentation," 2024.
[9] Antmicro, "Renode: Development framework for embedded systems," 2023.
[10] Imperas Software Ltd., "Ovpsim user guide," 2022.
[11] Edwin Christian Becerra Alvarez, "Embedded metalanguage operating system," `https://github.com/labcibernetica/EMOS`, 2026, Accessed: 2026-02-03.
[12] D. Chávez, "Intérprete de metalenguaje para aplicaciones en tiempo real, tesis de licenciatura," 2024.
[13] E. C. Becerra-Alvarez, J. J. Raygoza-Panduro, J. Rivera-Domínguez, S. Ortega-Cisneros, J. L. González-Vidal, and F. Sandoval-Ibarra, "Foremb: A forth-inspired, real-time interpreter for embedded systems," 2023, Proc. 12th International Conference on Embedded Systems.
[14] W. Stallings, *Computer Organization and Architecture: Designing for Performance, 12th ed.*, Pearson, 2021.
[15] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface, 5th ed.*, Morgan Kaufmann, Burlington, MA, USA, 2017.
[16] D. A. Patterson and A. Waterman, *The RISC-V Reader: An Open Architecture Atlas, 1st ed.*, Strawberry Canyon LLC, Berkeley, CA, USA, 2017.
[17] The RARS Project, "Rars: Risc-v assembler and runtime simulator," 2023.
[18] A. S. Tanenbaum, *Organización de computadoras, 5ª ed.*, Pearson, México, 2015.
[19] M. Sipser, *Introduction to the Theory of Computation, 3rd ed.*, Cengage Learning, 2013.
[20] A. Puga, "La máquina de turing," .

Hacia la eficiencia en la agricultura de precisión: diseño de una plataforma de sensores IoT y computación en el borde

Santiago Díaz-Romero[1*], Simón Pérez Martín[1*], Antonio Rios-Navarro[1], Ángel Jiménez-Fernández[1] y Alejandro Linares-Barranco[1]

Resumen— **Este trabajo presenta el desarrollo de una estación meteorológica de monitorización y toma de datos mediante el uso de diferentes sensores, cuyo objetivo es la predicción del comportamiento vegetal mediante el análisis continuo de variables ambientales y del suelo. El sistema se basa en una Raspberry Pi 5 como unidad central, encargada de gestionar una red de sensores que monitorizan parámetros como presión barométrica, temperatura, radiación ultravioleta y humedad del suelo, complementados con una unidad de medición inercial y un receptor GPS para el posicionamiento geoespacial.**

La arquitectura, implementada principalmente en Python, permite la adquisición periódica de datos, su almacenamiento en formatos estructurados y su transmisión. Estos datos pueden ser visualizados posteriormente visualizados a través de paneles de control o importados en otras herramientas a través de ficheros .csv. Asimismo, el sistema incorpora tecnología de comunicación LoRa, lo que permite garantizar su operatividad en entornos agrícolas remotos con conectividad limitada. En conjunto, la solución propuesta integra sensorización heterogénea avanzada, procesamiento de datos y comunicaciones eficientes para apoyar la toma de decisiones en agricultura de precisión.

Palabras clave— **Sensores, agricultura de precisión, IoT, IA, big data, edge computing.**

I. Introducción

ELEMENTOS como el aumento de productividad, la gestión del riego o la detección y eliminación de plagas han sido desafíos constantes en el entorno agrícola. Hoy día, además se enfrenta a otros retos como la sostenibilidad. En este escenario, la agricultura de precisión ha emergido como un paradigma clave, apoyado en tecnologías digitales avanzadas como el Internet de las Cosas (IoT por sus siglas en inglés), la inteligencia artificial y el análisis de datos[1]. Estas tecnologías permiten la monitorización continua de variables agronómicas —como la humedad del suelo, la temperatura, los nutrientes y el estado fisiológico de las plantas— mediante sensores ya sean en configuración única o mediante la creación de redes de sensores distribuidos. Este enfoque pretende ayudar a los agricultores facilitando la toma de decisiones tratando de hacerla más informada y eficiente.

El IoT aplicado a la agricultura se basa en la interconexión de dispositivos bien sean heterogéneos u homogéneos capaces de capturar datos en tiempo real desde el entorno agrícola. Dichos datos, tradicionalmente bien son almacenados para un análisis posterior o son enviados a infraestructuras en la nube para su procesamiento. Estos enfoques presentan limitaciones ya sea en términos de latencia, consumo de ancho de banda o dependencia de la conectividad, especialmente en entornos rurales o remotos. En este sentido, la computación en el borde (edge computing) surge como una solución que permite procesar los datos cerca de su origen, reduciendo la latencia y a menudo el consumo, mejorando la eficiencia operativa [2].

La integración de elementos IoT con computación en el borde habilita sistemas de monitorización inteligente capaces de realizar análisis en tiempo real y ejecutar acciones autónomas, como la optimización del riego, la detección temprana de enfermedades o el control de las condiciones ambientales [3]. Este enfoque distribuido no solo mejora la capacidad de respuesta ante eventos críticos, sino que también reduce la carga sobre infraestructuras centralizadas y aumenta la resiliencia del sistema.

A pesar de sus ventajas, la implementación de sistemas de monitorización basados en IoT y edge computing presenta desafíos importantes, incluyendo la interoperabilidad entre dispositivos, la gestión energética de los nodos sensores, la seguridad de los datos y la escalabilidad de las soluciones. Asimismo, existen limitaciones relacionadas con la heterogeneidad de los entornos agrícolas y la necesidad de modelos adaptativos que puedan operar en condiciones dinámicas y con recursos computacionales restringidos. En este contexto, el presente artículo aborda el diseño de una plataforma de monitorización de plantas y cultivos basados en dispositivos IoT que permita el entrenamiento de redes neuronales así como la aplicación de técnicas de computación en el borde. Se pretende contribuir al desarrollo de soluciones inteligentes que impulsen la eficiencia, sostenibilidad y resiliencia del sector agrícola.

II. Métodos y materiales

La plataforma propuesta ha sido diseñada para ser usada tanto como nodo autónomo de sensado y procesado como plataforma de recolección de datos que puedan ser posteriormente analizados o utilizados para entrenar modelos de inteligencia artificial.

[1]Dpto. de Arquitectura de Computadores, ETSII-EPS. I3US. SCORE. Universidad de Sevilla, e-mail: sdiaz5@us.es.
*Estos autores han contribuido por igual a este trabajo.

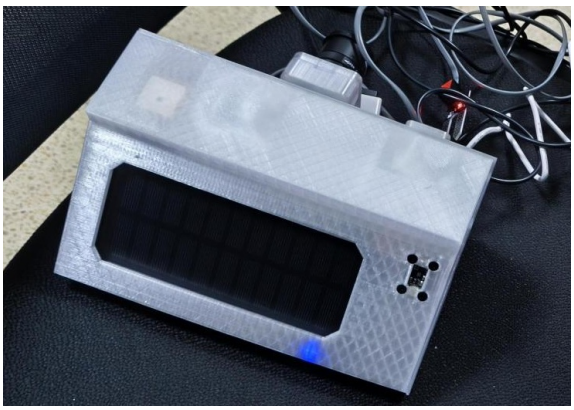


Fig. 1: Montaje final del prototipo.

A. Planta monitorizada

La especie seleccionada para la recogida de datos ha sido la planta del aguacate. Esta se considera un buen modelo para la monitorización técnica mediante sensores ya que a diferencia de otros frutales más resistentes o menos expresivos, el aguacatero actua como un sensor natural muy sensible.

Sus raíces son extremadamente sensibles ya que carecen de pelos radiculares eficientes, lo que las hace vulnerables tanto a la sequía como al encharcamiento. Las hojas del aguacate cierran sus estomas rápidamente ante un déficit de presión de vapor (VPD) elevado, deteniendo la fotosíntesis incluso si hay agua en el suelo. Además, presenta flujos de crecimiento muy marcados que requieren diferentes niveles de humedad y nutrientes según la etapa [4].

Elegir el aguacatero frente a otras plantas (como los cítricos o el olivo) ofrece las siguientes ventajas metodológicas para la recolección de datos:

- Alta reactividad a los Datos de Humedad. Mientras que un olivo puede soportar variaciones drásticas de humedad sin mostrar cambios inmediatos, el aguacatero reacciona en tiempo real.
- Correlación Directa entre Microclima y plagas ya que el aguacate es propenso a hongos de raíz.
- Dinámica de Crecimiento del Tronco mostrando cambios notables en el diámetro del tronco (contracción y expansión diaria) según su estado de hidratación.

B. Plaforma Hardware

Para el hardware, la plataforma presentada ha sido diseñada tratando desde un principio de cubrir todas las necesidades de la agricultura de precisión mediante una selección heterogénea de sensores a la vez que se ha tratado de mantener un precio de fabricación contenido para facilitar el acceso a la misma. Adicionalmente, se ha tenido en cuenta la posibilidad de una futura integración de sensores avanzados como son los dendrómetros, los medidores de flujo de savia o de turgencia, sensores ampliamente utilizados en la literatura[5][6].

Como base para el sistema se ha elegido una Raspberry Pi 5 que incorpora un procesador Broadcom BCM2712 Arm Cortex A76 de 4 núcleos a 2.4GHz, incluye conectividad Wi-Fi y Bluetooth, puertos USB 2.0 y 3.0, una interfaz PCIe 2.0 para la conexión de periféricos de alta velocidad, así como dos interfaces MIPI (Mobile Industry Processor Interface) para dar soporte a cámaras fotográficas. La Raspberry Pi presenta una serie de ventajas como la plena compatibilidad con python lo que permite el uso de cualquier librería o herramientas desarrolladas para este lenguaje. Además, al hacer uso de un sistema Linux permite la ejecución de una amplia variedad de software y drivers para periféricos USB. También, posee una superior potencia de cómputo en comparación con un microcontrolador lo que posibilita el uso de un mayor número de sensores, sensores con mayor ancho de banda o la ejecución en el borde de modelos de IA. Por contra, tiene un tamaño, precio y consumos superiores al de un microcontrolador y no posee un modo de bajo consumo por lo que debe este debe gestionarse mediante hardware adicional.

La plataforma integra un sensor digital de presión BMP280 conectado al bus I2C de la Raspberry pi. El BMP280 es un sensor de presión absoluta de alto rendimiento diseñado para dispositivos portátiles proporcionando una precisión relativa de $\pm 1m$. Este sensor proporciona medidas de presión y temperatura con una resolución de hasta $0{,}16Pa$ y $0{,}01^oC$ respectivamente a una frecuencia máxima de $157Hz$. Adicionalmente, posee un filtro IIR (Infinite Impulse Response) integrado con varias configuraciones preestablecidas. El objetivo de este sensor es obtener medidas de presión y temperatura ambientales que permitan determinar las condiciones del entorno de la planta.

Al mismo bus I2C, se encuentra conectado un módulo MinimU-9 v5 de Pololu. Este módulo ofrece 9 grados de libertad incorporando giroscopio, acelerómetro y magnetómetro en un único PCB. Las resoluciones de cada uno son ajustables individualmente. Aunque, en el prototipo producido este módulo se encuentra integrado dentro de la carcasa para detectar posibles movimientos, caídas o sustracciones. Otra posibilidad es sacar el módulo fuera de la carcasa para anexarlo a una hoja o rama para estudiar su movimiento a lo largo del tiempo.

Otro sensor conectado al bus I2C es el LTR390. Se trata de un módulo diseñado para medir el índice ultravioleta (UV) que incide en su sensor. Se encuentra localizado a un lado de la carcasa, cerca de la batería solar. Dispone de una resolución, ganancia y frecuencia de muestreo programable. Este sensor puede ser utilizado tanto para comprobar la cantidad de radiación UV que recibe tanto la planta como el panel solar integrado. Esto permite evaluar el estrés lumínico de la planta así como los posibles daños ocasionados por este. Adicionalmente, también puede ser usado para prever la efectividad de carga futura de la batería mediante la luz incidente en el sensor. La tabla II muestra la correspondencia entre el estrés lumínico y el índice UV.

El último módulo conectado al bus I2C, es el módulo ADS1115 que incluye un convertidor analógi-

Tabla I: Aguacate vs. Otros Cultivos.

Característica	**Aguacate**	**Cítricos**	**Olivo**
Respuesta al riego	Instantánea (Alta)	Moderada	Lenta
Sensibilidad a Tª	Muy Alta	Media	Baja
Predicción de plagas	Muy vinculada a humedad/Tª	Vinculada a ciclos estacionales	Vinculada a ciclos estacionales
Valor de los datos	Crítico para la supervivencia	Útil para optimización	Útil para ahorro de agua

co digital (ADC por sus siglas en inglés) con 4 canales de 16 bits de resolución y que cuenta con un amplificador de ganancia programable (PGA por sus siglas en inglés) integrado. Este módulo se ha incluido con el objeto de hacer de interfaz entre la raspberry pi y los sensores que provean una salida analógica. En esta primera versión de la plataforma únicamente se ha hecho uso de un canal de los 4 disponibles por lo que siguen quedando 3 canales disponibles para una futura expansión.

Conectado a uno de los canales del ADS1115 se encuentra el sensor de humedad de suelo YL-69. Este módulo es de tipo resistivo y se utiliza para medir la humedad del sustrato donde se encuentra enraizada la planta. En muchas versiones, este módulo se encuentra de forma conjunta con un amplificador operacional y un potenciómetro variable que permite ajustar manualmente la ganancia del módulo.

El último módulo de sensado que se encuentra incluido en esta primera versión de la plataforma es el DS18B20, un sensor de temperatura digital que se encuentra conectado directamente a un puerto GPIO (General Purpose Input/Output) de la Raspeberry Pi a través del protocolo 1-Wire. Este módulo es capaz de medir temperaturas entre -10^oC y $+85^oC$ con una resolución de $\pm 0{,}5^oC$. Este módulo se utiliza para medir la temperatura del sustrato donde esta la planta. En conjunción el módulo YL-69 permite tener información completa sobre el estado del suelo mientras que el módulo BMP280 ofrece la misma información sobre el ambiente.

Además de los módulos de sensado anteriormente descritos, se ha incluido un módulo GPS NEO GY-GPS6Mv2 con antena cerámica. Este módulo que se comunica mendiante comunicación serie UART, puede ser utilizado tanto para ubicar el módulo sobre el terreno como para evitar posibles substracciones mediante la obtención de la localización geográfica actualizada. Adicionalmente, la señal GPS puede utilizarse para sincronizar el RTC de la Raspberry Pi en caso de falta prolongada de conectividad Wi-Fi. Todo lo anterior permite ubicar los datos tomados y sincronizarlos lo que es necesario cuando se quiere cubrir una mayor extensión de terreno usando múltiples nodos[7].

El último módulo que se puede encontrar es el LoRa Ra-01 SX1278. Este módulo está conectado a la Raspberry Pi a traves de una conexión serie SPI (Serial Peripheral Interface). El SX1278 es un transceptor de radiofrecuencia de largo alcance y bajo consumo operando en la banda LPD433, libre de licencia. La tecnología LoRa es ampliamente utilizada en entornos IoT[8], [9], [10] gracias a su bajo consumo y amplio alcance incluso en condiciones adversas. Este módulo puede ser utilizado para la comunicación de datos entre nodos o a un nodo central para reducir el consumo.

Tabla II: Estrés lumínico según índice UV.

Riesgo	**Índice UV**
Bajo	0-2
Moderado	3-5
Alto	6-7
Muy alto	8-10
Extremadamente alto	11+

Los módulos junto a la Raspberry Pi y una batería recargable mediante celdas solares se han integrado dentro de una carcasa impresa en 3D en PETG (Tereftalato de Polietileno modificado con Glicol) para que pueda ser utilizada en exteriores (Fig 1). Se han dejado conexiones externas a un puerto USB y HDMI para poder interactuar con el prototipo de forma sencilla durante las pruebas a desarrollar. Además, dispone de una tuerca roscada M6 en la parte inferior para poder montar el dispositivo sobre un trípode.

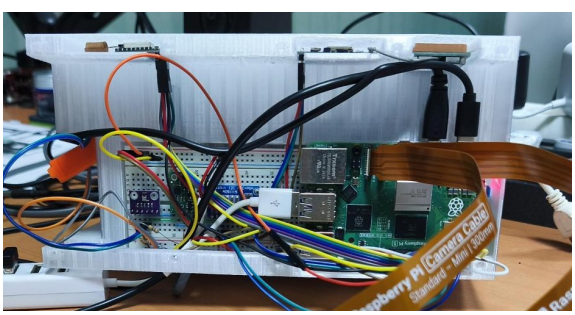


Fig. 2: Distribución interior de los módulos.

Externamente, se han acoplado a la carcasa dos cámaras (Fig 3) orientables. La primera de ellas es una Raspberry Pi HQ Camera de 12 Megapíxeles (MP) de resolución y compatibilidad con óptica externa lo que amplía su rango de usos. Esta cámara hace uso de un sensor Sony IMX477 y se conecta a la Raspberry a través de un cable FPC (Flexible Printed Circuit) a una de las interfaces MIPI disponibles en la Raspberry pi. Esta cámara puede utilizarse para monitorizar el estado de una hoja concreta con el fin de detectar e identificar posibles enfermedades o plagas.

La segunda cámara es una Raspberry Pi Camera NoIR. Esta es una cámara de 8 MP que hace uso de un sensor Sony IMX 219 con foco fijo a la que se le ha eliminado el filtro de bloqueo infrarojo (IR) lo que permite una mejor visión en entornos de baja luminosidad. Su conexión, como la anterior se hace a través de un cable FPC a una interfaz MIPI. Como contrapartida, la eliminación del filtro de bloqueo IR hace que los colores se vean adulterados bajo condi-

ciones normales de luz. Esta segunda cámara puede ser utilizada para monitorizar una hoja o planta en entornos con baja luminosidad o como sistema de vigilancia adicional.

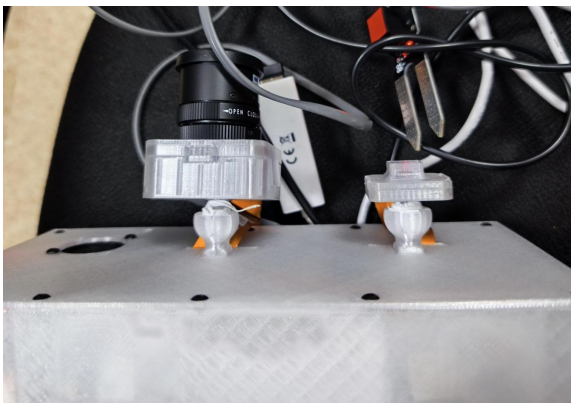


Fig. 3: Detalle cámaras.

C. Software

Respecto al software se ha optado por su desarrollo en python ya que este ofrece algunas ventajas como puede ser su amplia adopción en disciplinas tradicionalmente no relacionadas con la informática pero relacionadas con este desarrollo como la ingeniería agrícola[11] o la biología[12]. Esto permite que usuarios de estas disciplinas puedan manejarlo o personalizarlo con mayor facilidad. Además, python permite una fácil y rápida inclusión de otros sensores gracias a que existen numerosos proveedores que también ofrecen librerías ya desarrolladas para este lenguaje. Por último, software comúnmente utilizado para investigación en los campos anteriormente mencionados como por ejemplo OpenSourceOV[13] ha sido desarrollado en python lo que facilitará su futura integración con estos.

Para este primer prototipo el software desarrollado únicamente permite la recolección de datos. Esto deja el análisis de los datos obtenidos y la toma de decisiones para versiones posteriores de la plataforma. Del mismo modo, tampoco se ha desarrollado aún la comunicación entre nodos.

En el modo de recogida de datos, el dispositivo recoge los datos de los sensores de forma periódica y los almacena junto a su timestamp en la tarjeta SD en un fichero .csv separado para cada sensor además de un fichero general con la información integrada de todos los sensores.

III. Resultados experimentales

Para esta primera prueba se ha instalado la plataforma en un invernadero perteneciente al Instituto de Recursos Naturales y Agrobiología de Sevilla (IRNAS) recolectando datos de una planta de aguacate. Ya que el invernadero es un entorno cerrado y controlado se ha decidido no hacer uso de los módulos GPS e IMU. Además, dado que no entra luz directa en ningún momento, no es posible la carga de la batería mediante la celda solar por lo que el módulo ha permanecido conectado permanentemente a la red eléctrica.



Fig. 4: Dispositivo instalado en invernadero.

La recogida de datos se ha prolongado durante aproximadamente una semana. Para esta prueba, la recogida de datos por los sensores se ha programado con una periodicidad de 10 segundos mientras que la periodicidad de la toma de imágenes por las cámaras se estableció en 1 hora ya que el tamaño de las mismas puede resultar un problema para periodos largos de funcionamiento y no se consideró necesaria una frecuencia de muestreo mayor.

Durante esos 7 días, se han recogido aproximadamente 63,000 entradas por cada sensor y 200 imágenes por cámara. Estudiando los ficheros .csv, se pudo observar que los datos relativos a la humedad recogidos por los sensores YL-69 (Fig 6) y BMP280 presentaban ligeras variaciones que a efectos cuantitativos pueden considerarse constantes. Este comportamiento se debe por un lado al carácter cerrado del invernadero, lo que propicia que la humedad ambiental sufra pocas variaciones y por otro a que, los datos recogidos se corresponden con un ciclo de riego por lo que la humedad en suelo medida por el YL-69 se mantiene en el 100 % de forma consistente..

Respecto a la temperatura, los sensores recogen una mayor variabilidad sobre todo la medida en ambiente donde es posible ver las diferencias térmicas entre el día y la noche. Aún así, la oscilación es baja manteniendose siempre entre los 23^oC y los 32^oC para la temperatura ambiente y entre los 19^oC y los 22^oC para el suelo (Fig 7). Esta falta de variabilidad se debe a que el invernadero se encuentra climatizado por lo que las variaciones de temperatura son leves.

Por último, el sensor LTR390 encargado de valorar el estrés lumínico de la planta, ha medido un índice UV de 0 permanentemente a lo largo de toda la



(a) 16-03-2026 (b) 19-03-2026 (c) 24-03-2026

Fig. 5: Evolución temporal de una hoja durante la captura de datos.

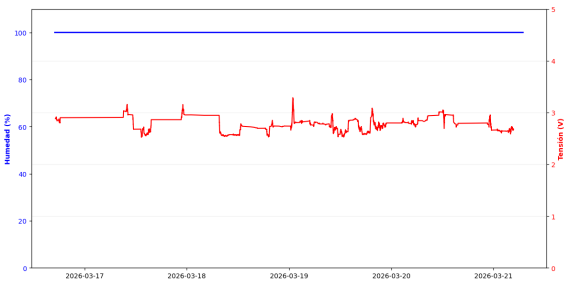


Fig. 6: Evolución temporal de la humedad.

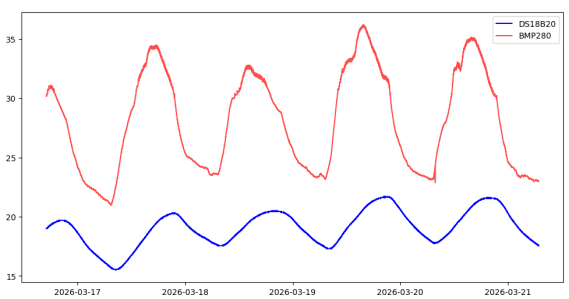


Fig. 7: Evolución temporal de la temperatura.

duración del experimento. Esta lectura es coherente con el entorno donde se han realizado las pruebas ya que, como se ha comentado con anterioridad, el invernadero no recibe nunca luz natural directa lo que elimina la radiación UV recibida.

IV. Conclusiones

El dispositivo ha funcionado satisfactoriamente de manera ininterrumpida durante un periodo ligeramente superior a 7 días generando una cantidad de datos elevada a los que se ha dado formato y posteriormente almacenado. La ejecución de las pruebas en un entorno cerrado con una variabilidad limitada han ayudado a verificar el correcto funcionamiento del sistema ya que a pesar de la falta de variabilidad en los datos, estos han demostrado ser relevantes, correctos y coherentes con las características del entorno en el que han sido tomados.

Todo lo anterior demuestra la capacidad del sistema para trabajar correctamente durante periodos de tiempo prolongados y sin interrupciones significativas así como la viabilidad del sistema para generar sets de datos relevantes y fiables para su posterior análisis o entrenamiento de una IA para la toma de decisiones.

En pruebas futuras se pretende probar la funcionalidad completa del sistema para recoger datos de forma autónoma durante periodos prolongados en situaciones controladas más realistas donde exista mayor variabilidad y el número de sensores involucrados sea mayo. Esta futura prueba ayudará a verificar el funcionamiento de los sensores que no han podido probarse en esta primera prueba controlada debido a las características del entorno, así como de la autonomía y robustez del dispositivo. En futuras versiones del software se pretende implementar la comunicación de bajo consumo vía LoRa, desarrollar una interfaz de usuario más amigable que pueda ser utilizada fácilmente por personal no especializado y añadir compatibilidad con un mayor número de sensores externos cuyos datos puedan resultar relevantes para el estudio de diferentes plantas. Además, los datos generados durante las distintas pruebas pretenden ser usados para el entrenamiento de redes neuronales pulsantes que ayuden a identificar riesgos y carencias así como apoyar la toma de decisiones o ejecutarlas de forma autónoma. Queda también para una implentación posterior la ejecución de estas redes en el propio dispositivo en tiempo real y con un bajo consumo.

Agradecimientos

El Ministerio de Transformación Digital y Función Pública de España ha financiado parcialmente esta investigación mediante la subvención USECHIP (TSI-069100-2023-001) del programa PERTE «Cátedra Chip», financiado por la Unión Europea – Next

Generation EU, por NEKOR (PID2023-149071NB-C54/AEI/10.13039/501100011033) y por la acción complementaria PAIDI 2020 (QUAL21 008 USE) de la Junta de Andalucía. Agradecemos al IRNAS, espcialmente a Virginia Hernández Santana, por el apoyo en las pruebas.

Referencias

[1] Kushagra Sharma and Shiv Kumar Shivandu, "Integrating artificial intelligence and internet of things (iot) for enhanced crop monitoring and management in precision agriculture," *Sensors International*, vol. 5, pp. 100292, 2024.

[2] Ananda Mohon Ghosh and Katarina Grolinger, "Edge-cloud computing for internet of things data analytics: Embedding intelligence in the edge with deep learning," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 3, pp. 2191–2200, March 2021.

[3] Tarek Alahmad, Miklós Neményi, and Anikó Nyéki, "Applying iot sensors and big data to improve precision crop production: A review," *Agronomy*, vol. 13, no. 10, 2023.

[4] Nektarios N Kourgialas and Zoi Dokou, "Water management and salinity adaptation approaches of avocado trees: A review for hot-summer mediterranean climate," *Agricultural Water Management*, vol. 252, pp. 106923, 2021.

[5] Katherine A. McCulloh, Klaus Winter, Frederick C. Meinzer, Milton Garcia, Jorge Aranda, and Barbara Lachenbruch, "A comparison of daily water use estimates derived from constant-heat sap-flow probe values and gravimetric measurements in pot-grown saplings," *Tree Physiology*, vol. 27, no. 9, pp. 1355–1360, 09 2007.

[6] C.M. Rodriguez-Dominguez, W. Ehrenberger, C. Sann, S. Rüger, V. Sukhorukov, M.J. Martín-Palomo, A. Diaz-Espejo, M.V. Cuevas, J.M. Torres-Ruiz, A. Perez-Martin, U. Zimmermann, and J.E. Fernández, "Concomitant measurements of stem sap flow and leaf turgor pressure in olive trees using the leaf patch clamp pressure probe," *Agricultural Water Management*, vol. 114, pp. 50–58, 2012, For a better use and distribution of water.

[7] Joaquín Gutiérrez, Juan Francisco Villa-Medina, Alejandra Nieto-Garibay, and Miguel Ángel Porta-Gándara, "Automated irrigation system using a wireless sensor network and gprs module," *IEEE Transactions on Instrumentation and Measurement*, vol. 63, no. 1, pp. 166–176, 2014.

[8] Mohammed Jouhari, Nasir Saeed, Mohamed-Slim Alouini, and El Mehdi Amhoud, "A survey on scalable lorawan for massive iot: Recent advances, potentials, and challenges," *IEEE Communications Surveys Tutorials*, vol. 25, no. 3, pp. 1841–1876, 2023.

[9] Maram Fahaad Almufareh, Mamoona Humayun, Zulfiqar Ahmad, and Asfandyar Khan, "An intelligent lorawan-based iot device for monitoring and control solutions in smart farming through anomaly detection integrated with unsupervised machine learning," *IEEE Access*, vol. 12, pp. 119072–119086, 2024.

[10] Jehangir Arshad, Musharraf Aziz, Asma A. Al-Huqail, Muhammad Hussnain uz Zaman, Muhammad Husnain, Ateeq Ur Rehman, and Muhammad Shafiq, "Implementation of a lorawan based smart agriculture decision support system for optimum crop yield," *Sustainability*, vol. 14, no. 2, 2022.

[11] Muhammad Azhar Iqbal, *Python for Agriculturists*, Springer, 2025.

[12] Brad Chapman and Jeffrey Chang, "Biopython: Python tools for computational biology," *SIGBIO Newsl.*, vol. 20, no. 2, pp. 15–19, Aug. 2000.

[13] Chris Lucani and Tim Brodribb, "Opensourceov," 2024.

Arquitecturas, lenguajes y sistemas tolerantes a fallos (2/2)

Accelerating Zero-Knowledge Proof Kernels on CGRA Architectures

Cristian Campos Ferrer, Ángeles G. Navarro y Sonia González-Navarro[1]

Abstract— **Wearable biomedical devices increasingly require on-device privacy guarantees through Zero-Knowledge Proofs (ZKPs), yet the underlying cryptographic kernels, based on modular exponentiation over prime fields, far exceed what low-power microcontrollers can deliver. Coarse-Grained Reconfigurable Arrays (CGRAs) offer near-ASIC efficiency with the reprogrammability needed to track evolving ZKP standards, but programming them requires cycle-by-cycle scheduling across a spatial processing-element mesh. We present an efficient spatial-temporal mapping of the S-box layer ($x^7 \bmod p$, BabyBear) of the ZKP-friendly hash function Poseidon2 onto the OpenEdgeCGRA architecture, a 4×4 toroidal PE mesh. The mapping leverages the mesh topology to perform Barrett modular reduction entirely through inter-PE communication, reducing the memory-access stalls that limit naive approaches. Through subroutine reuse of the Barrett reduction phase, fall-through chaining, and instruction merging across PEs, the S-box kernel achieves 1.26× speedup w.r.t. a hand-optimized flat schedule and 6.58× over an automatic compiler. We also identify that for long ZKP kernels multi-batch configuration reload is the dominant bottleneck, and show that increasing the Configuration Register File from 32 to 512 entries (+4.5 % FPGA area) enables deploying the full Poseidon2 kernel on the CGRA, achieving 1.33× speedup and 33.7 % energy savings over an ultra-low power RISC-V CPU core.**

Keywords— **OpenEdgeCGRA, Zero-Knowledge Proofs, Poseidon2.**

I. Introduction

WEARABLE biomedical devices—continuous glucose monitors, cardiac rhythm analyzers, inertial fall predictors—generate vast amounts of physiological data transmitted wirelessly to cloud platforms [1]. Regulatory frameworks (HIPAA, GDPR) impose strict requirements on data provenance, yet current device architectures lack mechanisms for the device itself to *prove* that a measurement is authentic without revealing raw signals. Zero-Knowledge Proofs (ZKPs) [2], [3] resolve this tension: a prover (the sensor) convinces a verifier (a server) that a statement holds—e.g., that a glucose trajectory was computed correctly—without disclosing the underlying data. Elliptic Curve Cryptography (ECC) [4] underpins the most compact constructions, keeping proof sizes small for bandwidth-constrained links.

Deploying ZKP on hardware demands *reconfigurability*. Different proof systems (Groth16 [2], PLONK [3], STARKs [5]) use different prime fields (31-bit BabyBear, 64-bit Goldilocks, 256-bit BN254), S-box exponents ($d \in \{3, 5, 7\}$), and elliptic curves. Plonky3 [6] introduced BabyBear only in 2024, replacing the Goldilocks field of Plonky2. An ASIC designed for one configuration becomes obsolete when the next proof system changes parameters. Coarse-Grained Reconfigurable Arrays (CGRAs) occupy the right design point: unlike FPGAs, they reconfigure at the instruction level with microsecond latency; unlike ASICs, they remain fully reprogrammable [7]. In our research, we target the X-HEEP framework [8], an extensible RISC-V based SoC designed for ultra low power edge computing, capable of interfacing with custom peripherals like accelerators. In particular, applications of the X-HEEP framework, such as HEEPocrates and HEEPsilon feature a CGRA integrated via the Extendible Accelerator Interface (XAIF): the OpenEdgeCGRA [9], a 4×4 PE mesh with a 32-bit datapath, which we consider in this paper.

The computational bottleneck in ZKP is the hash function. We focus our research on the Poseidon2 framework [10], which implement a hash that minimizes finite-field multiplications per permutation making it highly suitable for our data privacy use case (a biomedical wereable). In any case, its main computational kernel is the S-box layer—$x^d \bmod p$—, which accounts for over 90 % of computation on a CGRA with 32-bit datapath. For 31-bit BabyBear ($p = 2^{31} - 2^{27} + 1$, $d = 7$), each S-box requires two modular squarings and two modular multiplications, each involving Barrett reduction [11] with 62-bit intermediates decomposed across 32-bit PEs. Yet programming the CGRA is the real challenge: the developer must specify, for each of the $N \times M$ PEs and each clock cycle, the exact instruction, registers, and neighbor links. Moreover, a single misplaced operand produces silent errors detectable only through full simulation. In particular, the S-box layer spans 2,030 lines and took two weeks of expert effort. Automatic compilers [12], [13] avoid this burden, but cannot express cross-PE carry chains, toroidal routing, or branchless conditional selects that are essential for modular arithmetic.

In this paper we present CASTM (CGRA API & Spatial Temporal Mapper), which occupies the middle ground: the programmer retains explicit control over *which PE* executes *which instruction* at *which cycle*, but through parametric functions, loops, labeled subroutines, and architecture-aware operators instead of raw CSV encodings. A four-stage deterministic pipeline produces simulator-native CSV with structured error diagnostics. More precisely, we make the following contributions: **(C1)** We introduce CASTM, the first API targeting spatial-temporal CGRA programming for cryptographic

[1]Dpto. de Arquitectura de Computadores, Universidad de Málaga, e-mail: {cricamfe,angeles,sonia}@ac.uma.es

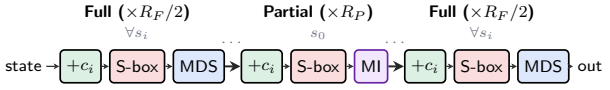


Fig. 1: Poseidon2 permutation structure. The S-box ($x^d \bmod p$) dominates cost.

workloads (Section III). **(C2)** We map the S-box layer ($x^7 \bmod p$, BabyBear) onto OpenEdgeCGRA and validate the resulting flow against independent C/C++ references (Section III-C). **(C3)** We show CASTM reduces source code by approximately 6.0× vs. manual CSV while achieving 20.5 % fewer clock cycles on the S-box kernel (Section IV-A). **(C4)** We identify multi-batch configuration reload as the dominant bottleneck and evaluate increasing Configuration Register File –CRF– capacity to enable full Poseidon2 deployment on CGRA (Section IV-B). **(C5)** We show that throughput and energy favor different deployments of Poseidon2 on the CPU and the CGRA (Section IV-C).

II. Background and Related Work

A. Poseidon2 S-box Exponentiation

Poseidon2 [10] is a ZKP-friendly hash function engineered to minimize finite-field multiplications per permutation, since each multiplication translates directly to proof-generation cost. Fig. 1 shows its round structure: it operates on a state $s = (s_0, s_1, s_2, \ldots, s_{t-1})$ of t field elements through $R_F/2$ full rounds, then a R_P partial round for cell s_0, and next $R_F/2$ full rounds again on s. Each round applies a S-box layer ($x \mapsto x^d \bmod p$), a linear *M*aximum Distance Separable layer (MDS, a $t \times t$ matrix used to provide maximum diffusion), and a round-constant addition ($+c_i$, a standard feature to ensure each round behaves differently); in partial rounds, the round constant and S-box apply only to s_0, followed by the internal matrix MI. For a typical parameterization ($t=16$, $R_F=8$, $R_P=13$), Poseidon2 invokes $8 \times 16 + 13 = 141$ S-box evaluations, making the S-box layer over 90 % of total computation on a 32-bit CGRA.

Algorithm 1 shows the pseudocode for the S-box layer, which is based on the modular Barrett multiplication [11] (`MULBARRET()`). We will go through this code in more detail in Section III-C

Algorithm 1 S-box $x^7 \bmod p$

Input: $x \in [0, p)$, and $\mu = \lfloor 2^{2n}/p \rfloor$, $n = \lceil log_2(p-1) \rceil$
Output: $x^7 \bmod p$
1: $x^2 \leftarrow$ MulBarrett(x, x)
2: $x^3 \leftarrow$ MulBarrett(x_2, x)
3: $x^4 \leftarrow$ MulBarrett(x_2, x_2)
4: $x^7 \leftarrow$ MulBarrett(x_4, x_3)
5: **function** MulBarrett(a, b)
6: $prod \leftarrow$ Multiply(a, b) ▷ Step1: Base Product
7: $\hat{q} \leftarrow$ QuotientEst$(prod, \mu)$ ▷ Step2: Quotient Est.
8: $r \leftarrow prod - \hat{q} \cdot p$ ▷ Step3: Mod. Substract.
9: **return** Select$(r < p,\ r,\ r - p)$ ▷ Step4: Correction
10: **end function**

B. OpenEdgeCGRA Architecture

Coarse Grained Reconfigurable Arrays (CGRAs) are primarily composed of an array of interconnected PEs (or Reconfigurable Cells –RCs–) that can be configured dynamically to implement a variety of computational tasks. The PEs are connected through a fixed interconnect, executing in lockstep under a configuration bitstream loaded before kernel execution [7]. Each PE contains a word-width ALU, a small register file, and neighbor connections. Unlike FPGAs (bit-level reconfiguration, high overhead), CGRAs reconfigure at the instruction level with microsecond latency.

OpenEdgeCGRA [9] provides: *(i)* a 4×4 toroidal mesh of PEs (i.e 16 PEs), where column 3 wraps to column 0 and row 3 to row 0, giving maximum distance of 4 hops between any two PEs; *(ii)* per-PE resources including a 32-bit ALU (1 clock cycle), 32×32 multiplier (3 clock cycles), four registers (R0–R3), output register ROUT visible to neighbors, and branchless select instructions (BSFA/BZFA); *(iii)* shared scratchpad SRAM with 2-cycle latency (one access/column/cycle) organized as 4 interleaved banks; and *(iv)* tight coupling with a RISC-V host core (X-HEEP [8]). At SoC level, we consider two host-to-memory organizations. The simpler *OneToM* (One-to-Many) path forces host and accelerator traffic to share a common route to memory, whereas the richer *N:M* crossbar sustains more parallel access to the interleaved SRAM banks. As a result, *OneToM* incurs more serialization and contention, while *N:M* reduces memory-induced stalls.

Configuration and execution model. A kernel is encoded as a sequence of 32-bit *configuration words*—one per PE per cycle—stored in a Configuration Register File (CRF) with D entries per column—32 entries in our baseline CGRA. Before dispatch, the host CPU writes configuration words into an SRAM backing store called the *Context Memory* (CMEM) via memory-mapped I/O (MMIO). During execution, each column's program counter advances through CRF entries in lockstep, dispatching one instruction per PE. When a kernel exceeds the CRF capacity, the host must reload the CMEM with the next batch of instructions before the CGRA can resume—a transparent but costly operation whose implications are analyzed in Section IV-B.

C. CGRA Programming: Limitations for ZKP Workloads

Manual scheduling requires placing instructions among NB bundles of PEs slots, satisfying data dependencies, multi-cycle latencies, memory-port arbitration, and register pressure—producing CSV files of ∼4,750 lines for our main kernel, with errors detectable only through simulation.

Automatic compilers (Compigra [12], Wang et al. [13]) accept C code and use heuristic placement, working well for regular DSP kernels but encountering structural limitations for cryptographic modular arithmetic (our use case): (1) multi-precision limb

decomposition is invisible at the C level; (2) toroidal routing patterns have no C-level representation; (3) architectural primitives like BSFA (*Branch/Select on Flag A*, a branchless correction needed in our main kernel) are inaccessible; and (4) heuristic scheduling forfeits cycle-count predictability needed for real-time ZKP functions.

D. Related Work

Table I positions our contribution relative to existing ZKP hardware accelerators. PipeZK [14] proposes an ASIC pipeline for MSM (*Multi-Scalar Multiplication*) and NTT (*Number Theoretic Transform*), which are the two most time-consuming computational primitives for generating zk-SNARKs (a type of Zero-Knowledge Proofs), but it does not address ZK-friendly hashing. AMAZE [15] implements MiMC (a ZK-friendly hash function) on FPGA with fixed RTL, achieving 13× over CPU. HashEmAll [16] provides FPGA-optimized cores for Rescue-Prime, Griffin, and Reinforced Concrete (up to 23× over CPU on BN254), yet it targets different hash families than Poseidon2. TRIDENT [17] accelerates the original Poseidon [18] on FPGA for BLS12-381 (Filecoin).

Although FPGAs are inherently reprogrammable platforms, the accelerators listed above implement their arithmetic pipelines as fixed RTL designs optimized for a specific prime field and hash function. Adapting them to a different field (e.g., from BLS12-381 to BabyBear) or a different hash (e.g., from MiMC to Poseidon2) would require modifying the HDL source, re-running synthesis and place-and-route, and generating a new bitstream—a process that takes minutes to hours and demands hardware design expertise. A CGRA operates at a different abstraction level: the hardware (ALUs, registers, interconnect) remains fixed, and switching to a new kernel amounts to loading a new instruction configuration via DMA, completing in microseconds with no hardware re-synthesis. This distinction is particularly relevant for wearable ZKP devices, where the proof system may evolve (new field, new exponent, new hash) and the device must adapt through a software update rather than FPGA re-flashing.

Separately, on the CGRA programming side, efficient CGRA compilers must support both spatially configured and time-multiplexed tasks. Compigra [12] and Wang et al. [13] provide automatic C-to-CGRA compilation tools that represent the state-of-the-art and consider some of the mentioned spatial-temporal challenges. However, they target general-purpose kernels and lack mechanisms for multi-precision spatial arithmetic, toroidal routing, or branchless correction—features essential for ZKP workloads.

To our knowledge, no prior work has mapped Poseidon2 hash function onto a CGRA, designed an API for spatial-temporal cryptographic programming, or analyzed Configuration Register File capacity pressure for ZKP-class workloads.

Tabla I: Comparison with related ZKP hardware accelerators. The "Reprog." column reflects the cost of switching to a different hash or field.

Work	Platf.	Hash	Field	Reprog.
PipeZK [14]	ASIC	MSM+NTT	BN254	Fabr.
AMAZE [15]	FPGA	MiMC	BN254	RTL
HashEmAll [16]	FPGA	Resc./Griff.	BN254	RTL
TRIDENT [17]	FPGA	Poseidon	BLS12	RTL
This work	**CGRA**	**Poseidon2**	**BabyB.**	**DMA**

III. The CASTM API

CASTM[1], can be seen as a deterministic C-like Domain Specific Language (DSL) whose design principle is *explicit spatial-temporal control with structured abstraction*: the programmer specifies what runs on which PE in which cycle, but through high-level constructs instead of raw encodings.

A. API Design

A CASTM program declares a *target profile*, a *build configuration*, and one or more *kernel* definitions. Table II organizes the constructs into three layers: spatial-temporal statements (instruction placement), structured abstractions (functions, loops, labels), and architecture-aware operators for modular arithmetic idioms.

Tabla II: CASTM API constructs by abstraction layer.

Layer	Construct	Semantics
Spatial-temporal statem.	`bundle {...}`	Group inst. in a bundle
	`@r,c: instr;`	Single-PE at (row, col)
	`at @r..r,c..c:`	Range across PE sub-grid
	`at all: instr;`	Broadcast to all PEs
	`R0 = R1 + R2;`	C-like expr. → ALU op
	`LWI/SWI R0, 40;`	Mem load/store (13-bit)
	`BSFA R3,R0,R2;`	Branchless select
Struct. abstr.	`function f(a,b)`	Parametric macro
	`for c in range(N)`	For loop
	`goto label/R3;`	Direct/indirect jump
Arch.-aware oper.	`extract_bytes`	Byte extraction
	`carry_chain`	Multi-limb carry prop.
	`broadcast`	Wave to all PEs
	`normalize`	Limb norm. + carry
	`reduce`	Collective accumul.

The *spatial-temporal statements* layer provides direct control over instruction placement: every statement maps one-to-one to a CSV entry. The *structured abstractions* layer reduces repetition without scheduling opacity: `function` expands the function body at the call site by default (parametric macro expansion) but can be compiled to shared subroutines under the `jump-reuse` expansion mode, which emits each function body once and links calls through a register-based return convention. The *architecture-aware operators* layer encapsulates multi-instruction CGRA idioms for modular arithmetic, each expanding to a fixed, documented sequence.

To illustrate the contrast with manual CSV, consider broadcasting a memory load to all 16 PEs. In CSV, this requires 16 identical entries per cycle (each encoding opcode, mux selects, register indices, and immediate as 7 numeric fields): `29,0,0,0,1,0,0` repeated 16 times. In CASTM, the same operation

[1]Available at `https://github.com/PhDCristian/CASTM`

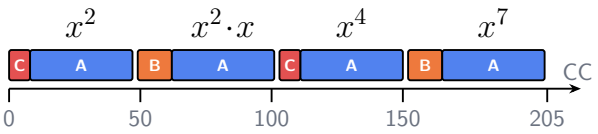


Fig. 2: Temporal breakdown of the S-box kernel (205 CC). Barrett reduction (`subrA`) consumes 76 % of execution.

is `bundle { at all: LWI R0, 0; }` –a single line that the compiler expands to 16 entries, eliminating an entire class of copy-paste errors–.

B. Compilation Pipeline

The compiler performs a four-stage deterministic pipeline: **Parse** (tokenize, syntax validate) → **Analyze** (semantic checks, function expansion, constant resolution) → **Lower** (operator expansion, loop, flat IR generation) → **Emit** (per-PE CSV encoding). Each stage is a pure transformation with no heuristic decisions. The compiler rejects ambiguous scheduling (e.g., two instructions on the same PE in the same cycle) with diagnostics reporting source location, PE coordinate, and cycle number.

CASTM combines three properties no prior CGRA tool provides simultaneously: *deterministic compilation with explicit placement* (cycle count predictable at authoring time), *architecture-aware operators* and *transparent expansion* (every construct has a documented, predictable expansion to CSV).

C. S-box Kernel Walkthrough

Fig. 2 illustrates how our S-box kernel decomposes $x^7 \bmod p$ into four (modular) phases: 1) x^2 (`subrC` –squaring–); 2) $x^3 = x^2 \cdot x$ (`subrB` –general multiplication–); 3) $x^4 = (x^2)^2$ (`subrC`); and 4) $x^7 = x^4 \cdot x^3$ (`subrB`). Each phase performs a Barrett multiplication [11] that consists of a base multiplication (`subrB`/`subrC`) followed by a Barrett reduction (`subrA`).

Fig. 3 now shows how the 4-steps Barrett multiplication introduced in Algorithm 1 is decomposed into four algorithmic stages mapped to a specific CGRA hardware capability. Step 1 represents the base product (`subrB`/`subrC`) while steps 2-4 correspond to the Barrett reduction (`subrA`). Step 1 is implemented as a parallel convolution: 32-bit operands decomposed into four 8-bit limbs, producing 16 partial products on the 4×4 PE mesh in our CGRA. Step 2 represents the quotient estimation and is mapped as a toroidal accumulation via nearest-neighbor routing; Step 3, the modular subtraction, avoids expensive division via subtraction and AND masking; then step 4 performs the final modular correction via branchless conditional `BSFA` (a single-cycle register select based on the ALU sign flag).

```
1  kernel "SBox" {
2   // Main: set return addr, jump to subrC
3   bundle { @0,1: goto subrC;
4            @3,0: R3 = ZERO + ret1; }
5   ret1: //▷ triple merge
6   bundle { @0,0: SWI R3, 720;
7            @0,1: goto subrB;
8            @3,0: R3 = ZERO + ret3; }
9   ...
10  subrC: // squaring accumulation (8 bndl)
11  extract_bytes_col(R0, R1); //▷ 4 PEs extract bytes
12  extract_bytes_row(R0, R0);
13  bundle { at all: R2 = R0 * R1; } //▷ 16 products
14  // Toroidal accumulation (4 bndl, RCR/RCT/RCB)
15  bundle { @3,0: goto subrA; }
16
17  subrA: // Barrett reduction (39 bndl)
18  bundle { @0,0: R1 = RCR;
19           @0,1: R0 = RCB;
20           @0,2: R1 = RCB; }
21  // Quotient estimation: 12 PEs
22  bundle { at @0..2,0..3: R2 = R0 * R1; }
23  // ... carry prop., normalization
24  // Branchless correction:
25  bundle { @0,0: R2 = R0 - RCL; }
26  bundle { @0,0: BSFA R3, R0, R2, SELF;}//▷ if r≥p: r-p
27  bundle { @3,0: goto R3; } //▷ return
28 }
```

Listing 1: S-box kernel for $x^7 \bmod p$ (key excerpts). `RCR/RCL/RCT/RCB` denote *Read Cell Right, Left, Top, Bottom*.

Listing 1 shows key excerpts of the S-box kernel. The kernel structure reveals several design decisions enabled by CASTM:

Jump-reuse subroutines. All calls to subroutines are performed via register-indirect jumps. The Barrett reduction phase (`subrA`, 39 bundles) is shared across all four modular operations through a simple calling convention: the caller stores a return-address label in R3 (`R3 = ZERO + retN`, line 4, line 8), and the subroutine returns with `goto R3` (line 27). This avoids duplicating 39 bundles × 4 calls = 156 wasted bundles, reducing total kernel size from ∼700 to 73 bundles (see Section IV-A). In CASTM, this pattern is expressed naturally with labels and `goto`; in manual CSV, the developer must compute absolute addresses for every jump target.

Triple instruction merging. At label `ret1`, three independent operations execute on three different PEs in the same cycle (lines 6-8): `SWI` stores the intermediate result, `goto` jumps to the next subroutine, and `SADD` prepares the return address. The `bundle{}` statement makes this spatial parallelism explicit and verifiable; in CSV, it is easy to accidentally place two instructions on the same PE, causing a silent overwrite.

Systolic convolution. The `subrC` uses the 4×4 mesh as a spatial multiplier (lines 10-14): `extract_bytes` operators decompose operands into byte-size limbs distributed across columns and rows, then `at all: R2 = R0 * R1` computes all 16 partial products simultaneously. Toroidal accumulation converges results in 4 cycles via neighbor reads (RCR, RCT, RCB). All 16 PEs compute their partial product simultaneously; for squaring, the symmetric pairs are folded via in-place doubling during the subsequent accumulation phase, reducing it from 13 to 8 bundles compared to `subrB` that accumulates all the partial results explicitly.

Branchless correction. The final BSFA instruction (line 26) selects between r and $r-p$ based on the sign flag, completing modular reduction in a single

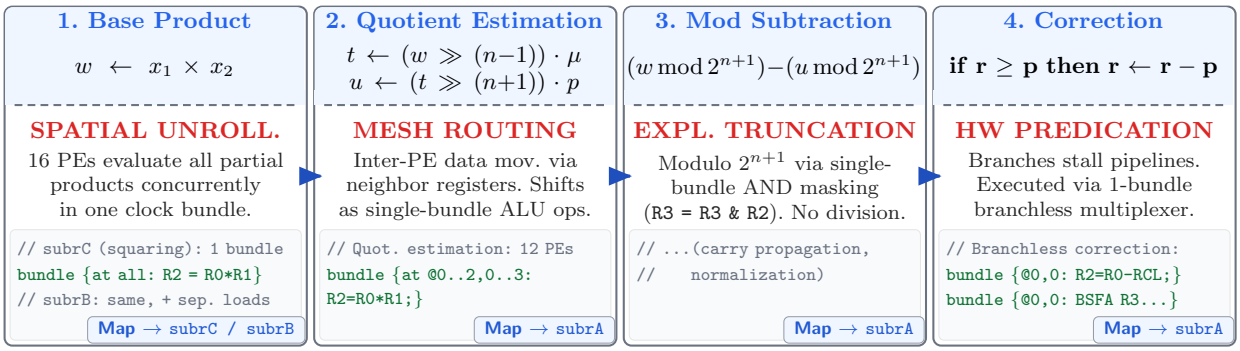


Fig. 3: Barrett multiplication mapped onto the CGRA. Each box shows the mathematical operation (top), hardware strategy (middle), and CASTM code excerpt (bottom).

cycle without branching—avoiding control divergence across the PE mesh.

The jump-reuse feature imposes a residency constraint: the full 73-bundles of the kernel must remain simultaneously resident in the Configuration Register File –CRF– so that register-indirect control flow can execute without fragmentation. When the kernel no longer fits, it must be partitioned into multiple dispatchable segments, sacrificing subroutine sharing and introducing reload overhead. Section IV-B quantifies this S-box-specific CRF-pressure trade-off and shows why it becomes a first-order architectural constraint for the full Poseidon2 evaluation.

D. Spatial Mapping Methodology

The S-box mapping above suggests a set of spatial-scheduling patterns that may be useful beyond this single kernel. Those patterns are abstracted from the concrete artifact discussed in Fig. 2, Fig. 3, and Listing 1.

D.1 Operand Decomposition across the PE Mesh

When an arithmetic operation produces intermediates wider than the PE datapath, one option is to serialize the work limb by limb on a single processing element. Our implementation instead distributes the limbs spatially across the mesh. For the 31-bit BabyBear field used here, each operand is decomposed into four 8-bit limbs aligned with rows or columns of the 4×4 array, exactly as summarized in Fig. 3 and instantiated through the `extract_bytes` operators in Listing 1. A single multiply bundle then emits the 16 products $a_i \cdot b_j$, and the anti-diagonal structure of those products determines how significance levels are accumulated (see Fig. 4(a)).

This decomposition is demonstrated directly for the BabyBear S-box studied in this paper. A related strategy should also be possible for other field widths, but the limb size, mesh dimensions, and routing budget would need to be re-tuned to the target arithmetic. For example, a 64-bit field could use wider limbs on the same mesh or more limbs on a larger array; we do not evaluate those variants here. For squaring ($a = b$), the symmetry $a_i \cdot a_j = a_j \cdot a_i$ is exploited in the accumulation schedule, which is why the current squaring path uses 8 bundles instead of the 13 bundles required by the general multiplication path.

D.2 Toroidal Carry Propagation

After partial products are formed, the current kernel moves carries and partial sums through neighbor registers rather than through a scalar loop. In `subrA` this is reflected by the `RCL/RCR/RCT/RCB` accesses highlighted in Listing 1. Also, normalization is expressed as a deterministic sequence of shifts, masks, additions, and transfers through the toroidal interconnect. The toroidal interconnect guaranties wrap-around connectivity, which can shorten some routes relative to a non-wrapping mesh, but the exact propagation distance still depends on placement and routing. The methodological point is therefore narrower: once limb positions are fixed, carry motion can be programmed as a repeatable spatial wavefront with architecture-visible data movement. This is illustrated in Fig. 4(b).

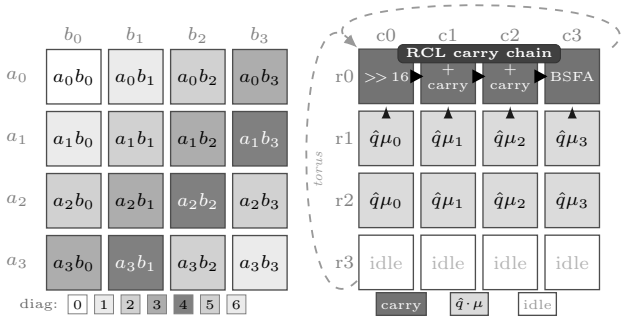
D.3 Subroutine Sharing on a Stack-less Architecture

OpenEdgeCGRA does not provide a hardware call stack, so shared control flow must be constructed explicitly. In this kernel, calls are direct label jumps, the return address is stored in `R3` at a fixed PE, and the callee returns with `goto R3`, as shown in Listing 1. Fig. 2 makes clear why this matters: the same reduction body is reused across the four modular operations in the exponentiation chain. The full kernel layout also exploits contiguous placement to support asymmetric chaining: `subrC` transfers control to `subrA` through an explicit jump, whereas `subrB` reaches `subrA` by fall-through. This convention is sufficient to share one Barrett reduction body across the four modular operations. We do not claim that the same calling discipline is optimal for every CGRA or every modular kernel; the point here is that CASTM can express and verify it directly on the target studied in this work.

D.4 Multi-PE Bundle Merging

A bundle activates all PEs in the same cycle, so idle PE slots are potential opportunities for control and datapath packing. The current kernel exploits this at subroutine boundaries; Listing 1 shows the representative `ret1` case, where a store, a jump, and a return-address update occupy disjoint PEs and therefore fit into one bundle. The benefit is measurable compaction of control work into cycles that would otherwise

contain idle spatial capacity; it is not correct to describe the overhead as disappearing. More generally, bundle-merging opportunities appear most naturally when control actions are localized to a small subset of PEs and can be scheduled beside arithmetic work.



(a) Partial-product diagonals (b) Barrett carry chain

Fig. 4: Spatial mapping of Barrett reduction onto the 4×4 CGRA. (a) PE(r, c) computes $a_r \cdot b_c$; shading marks partial-product diagonals (k=r+c), not communication hops. (b) Row 0 implements the RCL carry chain, while dashed torus links bound quotient/carry travel to at most 4 hops.

IV. Experimental Results

All CGRA cycle counts in Section IV-A are obtained from the ESL-CGRA simulator [19] (deterministic mode). Section IV-C reports steady-state RTL measurements from Verilator cycle-accurate simulation of the HEEPsilon SoC (ultra-low power CV32E20 host, RV32IMC, single-issue in-order). CPU-only time is derived at 250 MHz, whereas CGRA deployments on the OpenEdgeCGRA 4×4, 32-bit datapath use 100 MHz. CPU energy follows a fixed CV32E20 reference model of 2.5 mW applied to active-cycle counts; CGRA energy comes from the ESL simulator replay of the deployed kernels. The full Poseidon2 RTL data additionally includes a canonical-state replay with exact agreement to the reference.

A. S-box Kernel Comparison

We evaluate three approaches to mapping the S-box kernel onto the CGRA: (1) a hand-optimized schedule (Manual), (2) the Compigra compiler [12], and (3) CASTM, our approach. Table III summarizes the ESL-measured results; Fig. 5 presents a visual comparison along with the estimated energy per invocation [19].

Tabla III: S-box kernel: comparison of compilation approaches under ESL-CGRA simulation. *NB*: number of compiled bundles; *CC*: clock cycles including stalls; Disp.: number of CGRA dispatches; Idle (%): idle-PE ratio, summarized as a bundle-wise geometric mean.

Approach	*NB*	*CC*	Disp.	Idle (%)
Manual	406	678	1	64.3 %
Compigra	2,564	3,544	244	89.9 %
CASTM	**73**	**539**	1	**41.7 %**

Energy per invocation combines the ESL-reported execution energy with the fixed dead-time cost incurred at each CGRA dispatch. This normalization

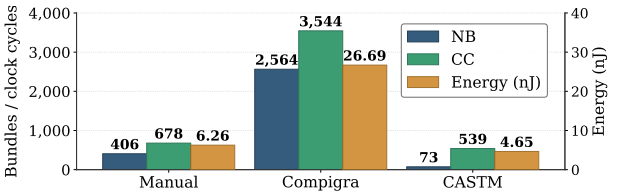


Fig. 5: Comparison of the S-box kernel mappings produced by Manual, Compigra, and CASTM on the CGRA. Blue and green bars use the left axis and report the number of compiled bundles (NB) and clock cycles (CC), respectively; orange bars use the right axis and report energy (nJoules) per invocation.

is necessary because the three mappings differ not only in cycle count but also in execution granularity: without accounting for dispatch overhead, a fragmented schedule would understate the system-level cost of repeatedly launching and draining the accelerator. Idle (%) in Table III is reported as a bundle-wise geometric mean over idle-PE per-bundle ratios.

Compigra cannot map S-box directly. The automatic compiler's scheduling heuristic fails on the S-box kernel because Barrett reduction requires cross-PE carry chains and branchless conditional selects that exceed its mapping capabilities. The workaround is to manually decompose $x^7 \bmod p$ into sub-kernels and iteratively subdivide when compilation fails—a trial-and-error process that negates the "automatic" premise. The result is 244 dispatches with 89.9 % Idle and 0.0 % of SMUL utilization: the hardware multiplier is never used, and the CGRA spends most cycles idle while data is shuffled between dispatches.

CASTM achieves the lowest cycle count. The 73 compiled bundles (vs. 406 for the Manual schedule) reflect jump-reuse subroutines that share Barrett reduction across the four modular multiplications (x^2, x^3, x^4, x^7). The resulting 539 CC (vs. 678 CC for Manual) represents a 20.5 % cycle reduction attributable to better PE utilization: 41.7 % Idle (vs. 64.3 %) and active use of the hardware multiplier (4.1 % SMUL). Under the energy model, CASTM also remains the most efficient mapping at 4.65 nJ, below Manual at 6.26 nJ and far below Compigra at 26.69 nJ. The gap is architecturally coherent with the schedules themselves: Compigra expands the kernel into 244 dispatches, so its higher energy reflects not only a longer execution (3,544 CC) but also repeated launch and quiescent overheads that are largely avoided by the single-dispatch in the CASTM schedule.

Productivity. The Manual approach requires 2,030 lines; CASTM requires 340 lines—an approximately 6.0× reduction—. This gap is structural: CASTM's parameterized functions and loop express the kernel once, whereas the hand-written schedule must spell out each PE-cycle placement explicitly, making coding slower and substantially more error-prone.

Code complexity. Table IV quantifies complexity using Halstead active volume metric, $V_{\text{act}} = N \log_2 \eta$, where N counts active (non-NOP) tokens [20]. We do not report Halstead effort because, for low-vocabulary ISA-level code, it can

produce anomalous scaling [21]. Owing to explicit per-PE-per-cycle encoding, the Manual schedule expands 64.5× the `C++` baseline. By absorbing spatial placement and NOP scheduling into the compiler, CASTM reduces this to 6.4×, corresponding to a 10.10× compression factor relative to the Manual schedule. Ideally, Compigra could reduce further this complexity to 2.4× the `C++` baseline, although at the cost of 6.58× more cycles than CASTM due to inefficient PE utilization as previously explained.

Tabla IV: Halstead metrics for the S-box kernel.

Approach	LOC	η	N	V_{act}	Ratio
C++ (Ref.)	25	39	104	550	1.0×
Compigra	47	53	227	1,300	2.4×
Manual CSV	2,030	48	6,348	35,453	64.5×
CASTM (**Ours**)	**340**	**55**	**607**	**3,509**	**6.4**×

LOC: source lines; η: unique vocabulary; N: active tokens (NOP excl.); $V_{\text{act}} = N \log_2 \eta$: active volume [20]; Ratio: V_{act} normalized to C++ reference. The lower the better.

B. S-box Configuration Register Pressure

The S-box kernel also allow us to conduct a controlled sensitivity study for configuration residency. Two implementations bracket the design space: the *Manual* kernel (406 bundles, flat layout) and the *CASTM* kernel (73 bundles, jump-reuse with shared subroutines). Because the Manual kernel uses only sequential control flow, it can be split into independent sub-kernels at any CRF depth. The CASTM kernel cannot: its register-indirect returns (`JUMP R3`) require all 73 bundles to be simultaneously resident in the CRF.

When a kernel of NB bundles exceeds the CRF depth D, it must be partitioned into $K = \lceil NB/D \rceil$ sub-kernels. In HEEPsilon's default CMEM organization, each sub-kernel fills an entire memory bank, so every transition between sub-kernels forces the host to reload all configuration data. The overhead decomposes as:

$$T_{\text{ovh}} = \underbrace{\textstyle\sum_i (s_i+1)}_{\text{CRF load}} + \underbrace{4(K-1)}_{\text{controller}} + \underbrace{(K-1)(16D+16)}_{\text{CMEM reload}}$$

where the sum runs over the $K-1$ transitions (s_i is the size of each reloaded sub-kernel). The first two terms—streaming instructions into the CRF and controller handshake cycles—are small. The CMEM reload term dominates: it transfers the full bank contents via MMIO, costing approximately $16NB$ cycles for a large K, nearly *constant* regardless of D.

At the default $D=32$, the Manual kernel loses 91 % of its cycles due to reload overhead ($K = 13$); doubling or quadrupling D barely helps because the total data volume stays proportional to NB. Overhead only vanishes when the entire kernel fits ($K = 1$): this happens for $D=512$ in Manual, and $D=128$ in CASTM. The 5.6× bundle compression from Manual to CASTM (406→73) now translates into a 4× reduction in CRF requirement, at the cost of 0.9 % additional area for $D = 128$ vs. 4.5 % for $D = 512$ (i.e. +49,152 flip-flops vs. +245,760 flip-flops on Artix-7 XC7A200T [8], respectively). However, this comes with a rigidity constraint: for any $D < 73$, the CASTM kernel cannot execute at all.

C. Poseidon2: RTL Validation

The S-box CRF-pressure analysis above showed that, for long ZKP kernels, configuration fragmentation and reload overhead dominate before arithmetic throughput can become visible. The full Poseidon2 kernel (366 bundles) requires $D=512$ to fit without partitioning. Under that condition, we analyze three deployment options when running a batch of Poseidon2 hashes: a *CPU-only* reference that executes all the hashes on the CPU, a *CGRA-only* version that offloads all the hashes to the CGRA, and a *CPU+CGRA* implementation, which evenly launches the hashes concurrently on the CPU and the CGRA. Both *CGRA-only* and *CPU+CGRA* keep the full Poseidon2 kernel on the accelerator. This evaluation is of interest for the case of Merkle tree computations [16] when efficient and secure verification of a dataset based on Poseidon2 hashing is required.

Tabla V: Metrics for Poseidon2 kernel. T: Time (μs); Th: Throughput (hashes per sec.); Sp: Speedup; E: Energy (μJ); ES: Energy Savings (%).

Config.	T_{CPU}	T_{CGRA}	T_{tot}	Th	Sp	E_{CPU}	E_{CGRA}	E_{tot}	ES
CPU-only	733.4	—	733.4	1,363	1.00	4.584	—	4.584	0
CGRA-only	7.4	543.5	550.8	1,816	1.33	0.047	2.993	**3.039**	**33.7**
CPU+CGRA	358.3	290.8	**358.3**	**2,791**	**2.05**	2.240	1.596	3.836	16.3

The throughput ranking is straightforward. T_{CPU} and T_{CGRA} report the time per-hash (full Poseidon2) for which the host CPU and the accelerator are active, with T_{tot} being the total time per-hash. From this last metric we see that *CPU+CGRA* is the fastest because both devices work in parallel reaching 358.3 μs/hash, the highest throughput, 2,791 hashes/sec., and up to 2.05× speedup over the *CPU-only*. Interestingly, there is room for improvement for this hybrid deployment because as we see from the T_{CPU} and T_{CGRA} times, the workload is currently not balanced between devices. *CGRA-only* also outperforms *CPU-only* at 550.8 μs/hash and 1.33× speedup.

Energy favors a different organization. E_{CPU} and E_{CGRA} represent the energy per-hash consumed by the CPU and CGRA, respectively, while E_{tot} is the total energy and ES is the energy savings with respect to *CPU-only*. Now, *CGRA-only* reduces the total energy to 3.039 μJ/hash, a 33.7 % reduction relative to *CPU-only*, whereas *CPU+CGRA* reaches 3.836 μJ/hash, or 16.3 % reduction. In the hybrid deployment, the E_{CGRA} component is the minimum, but the less efficient E_{CPU} contribution increases because the CPU computes part of the hashes, which ultimately degrades the total energy.

From the T_{CPU} (E_{CPU}) metric in the *CGRA-only* case, we can infer the overhead due to kernel dispatch (input parameters setting + kernel launching

+ output reading). As we see, this overhead is very low, around 1.3 % (1.5 %) total time (total energy).

In our study, we have found that memory stalls are the most significant source of overhead for this CGRA kernel, representing 23.5 % active cycles. These memory access stalls are mainly due to memory bank conflicts that are even more relevant if other memory subsystem design choices are considered. For instance, replacing the more flexible host-to-memory crossbar *N:M* that we have considered in this analysis, with the simpler shared path *OneToM*, degrades the *CGRA-only* performance from 56,416 CC to 68,643 CC and reduces speedup from 1.33× to 1.06×.

D. Lessons Learned

Our results provide some interesting architectural lessons. Firstly, CRF configuration capacity is a primary tuning knob: before fragmentation is removed, system behavior is dominated by reload traffic rather than arithmetic execution. Thus, kernels should be designed to fit the CRF as a single unit. Once this requisite is satisfied, throughput and energy do not necessarily favor the same organization of kernels: a hybrid deployment as *CPU+CGRA* provides the highest throughput, whereas *CGRA-only* minimizes total energy consumption. The next optimization step lies less in creating finer-grain offload variants than in reducing the cost of delivering data to an already efficient accelerator by minimizing memory bank conflicts. Also for hybrid deployments, load-balancing scheduling strategies must be considered.

V. Conclusions

In this paper, we have presented CASTM, the first API targeting the spatial-temporal programming model of CGRA architectures for cryptographic workloads. Specifically, for the BabyBear S-box layer on the OpenEdgeCGRA architecture, CASTM achieves a 20.5 % cycle reduction over hand-optimized CSV (539 CC vs. 678 CC), while reducing source code by approximately 6.0×.

RTL validation of the full Poseidon2 hash function on the ultra-low power HEEPsilon SoC gives us some system lessons. The Configuration Register File capacity is the enabling condition: once the kernel fits in the CRF the running times are no longer masked by reload fragmentation. Under that condition, offloading the full kernel to the CGRA (*CGRA-only*) minimizes the total energy, whereas a hybrid deployment that computes hashes on the CPU and the accelerator (*CPU+CGRA*) offers the highest throughput.

In summary, we have shown that it is possible to enable privacy-preserving verification on wearable biomedical devices through Zero-Knowledge Proofs, on hardware that is both energy-efficient and with reprogramming capability to track the rapid evolution of ZKP systems.

Future work. Two directions follow: (1) dual-port SRAM or DMA burst modes to reduce the memory stalls that represent now the main source of overhead; and (2) evaluation on alternative ZKP fields (Goldilocks, BN254) and post-quantum workloads to test the generality of CASTM's API.

Acknowledgments

This work was supported by the Ministry of Science, Innovation and Universities, Government of Spain (Grant Numbers TED2021-131527B-I00, PID2022-136575OB-I00 and TSI-069100-2023-0013).

References

[1] Duarte Dias and João Paulo Silva Cunha, "Wearable health devices—vital sign monitoring, systems and technologies," *Sensors*, vol. 18, no. 8, pp. 2414, 2018.

[2] Jens Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology – EUROCRYPT 2016*. 2016, pp. 305–326, Springer.

[3] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru, "PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," in *IACR Cryptology ePrint Archive*, 2019.

[4] Zhe Liu et al., "Efficient implementation of NIST-compliant elliptic curve cryptography for 8-bit AVR-based sensor nodes," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 7, pp. 1385–1397, 2016.

[5] Eli Ben-Sasson et al., "Scalable, transparent, and post-quantum secure computational integrity," in *IACR Cryptology ePrint Archive*, 2018.

[6] Polygon Zero Team, "Plonky3: A high-performance zk proving system," Technical report and project documentation, 2024.

[7] Artur Podobas et al., "A survey on coarse-grained reconfigurable architectures from a performance perspective," in *IEEE Access*, 2020, vol. 8, pp. 146719–146743.

[8] OpenHW Group and University of Bologna, "X-heep documentation," `https://x-heep.readthedocs.io/en/latest/index.html`, 2026.

[9] Rubén Rodríguez Álvarez et al., "An open-hardware coarse-grained reconfigurable array for edge computing," in *Proc. of the 20th ACM International Conference on Computing Frontiers*, New York, NY, USA, 2023, pp. 391–392, ACM.

[10] Lorenzo Grassi et al., "Poseidon2: A faster version of the poseidon hash function," in *Progress in Cryptology – AFRICACRYPT 2023*. 2023, pp. 177–203, Springer.

[11] Paul Barrett, "Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor," in *Advances in Cryptology — CRYPTO '86*. 1987, vol. 263 of *Lecture Notes in Comp. Sc.*, pp. 311–323, Springer.

[12] Embedded Systems Laboratory, EPFL, "Compigra: Open-source c/c++-to-cgra compilation toolchain," `https://github.com/esl-epfl/Compigra`, 2026.

[13] Yuxuan Wang et al., "An mlir-based compilation framework for control flow management on coarse grained reconfigurable arrays," 2025.

[14] Ye Zhang et al., "PipeZK: Accelerating zero-knowledge proof with a pipelined architecture," in *48th ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2021, pp. 416–428.

[15] Anees Ahmed et al., "AMAZE: Accelerated MiMC hardware architecture for zero-knowledge applications on the edge," in *Proc. of the 43rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24)*. 2024, pp. 1–9, ACM.

[16] Nojan Sheybani et al., "Gotta hash 'em all! accelerating hash functions for zero-knowledge proof applications," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2025.

[17] DatenLord, "TRIDENT: A hardware-implemented Poseidon hasher," `https://github.com/datenlord/TRIDENT`, 2022, ZPrize 2022 entry; Xilinx Varium C1100.

[18] Lorenzo Grassi et al., "Poseidon: A new hash function for zero-knowledge proof systems," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 519–535.

[19] Maxime Henri Aspros et al., "A flexible framework for early power and timing comparison of time-multiplexed cgra kernel executions," in *Proc. of the 22nd ACM International Conference on Computing Frontiers*. May 2025, pp. 62–65, ACM.

[20] Maurice H. Halstead, *Elements of Software Science*, Elsevier North-Holland, NY, 1977.

[21] Bishnu Prasad Sinha and Thambipillai Srikanthan, "A software complexity measure for reconfigurable computing," in *IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, 2011, pp. 1–4.

Evaluación de la fiabilidad y protección frente a errores en Vision Transformers

Lester Frías-Domínguez[1], José M. Badía[2], Germán León[2], Adrián Amor-Martín [3] y José A. Belloch[1]

Resumen— **El uso de modelos *Vision Transformer* (ViT) en entornos críticos requiere una evaluación exhaustiva de su fiabilidad, especialmente ante errores transitorios (*soft errors*). A pesar de su popularidad, la sensibilidad de sus distintas capas frente a estos fallos no se conoce aún en detalle. Este trabajo analiza la fiabilidad de la arquitectura ViT-Base mediante una metodología de inyección de fallos. Utilizando *kernels* CUDA personalizados, se simulan *bit-flips* en la representación IEEE 754 de los tensores del modelo, permitiendo cuantificar cómo se degrada la precisión al aumentar la tasa de error.**

Los resultados permiten identificar un "mapa de vulnerabilidad" que revela una sensibilidad crítica en las salidas de las capas de normalización y en las etapas totalmente conectadas. Para paliar estos efectos, se proponen y evalúan diversas técnicas de protección por software: desde el entrenamiento consciente de fallos (*Fault-Aware Training*), aplicado de forma global o selectiva, hasta mecanismos de mitigación en tiempo de ejecución como el acotado de rangos y el filtrado de valores no numéricos. El estudio demuestra que estas soluciones puramente algorítmicas mejoran significativamente la tolerancia a fallos del modelo.

Palabras clave— **Vision Transformers, tolerancia a fallos, efectos de radiación, errores transitorios, PyTorchFI, fiabilidad de redes neuronales, sistemas críticos, mitigación de errores.**

I. Introducción

DURANTE la última década, la Inteligencia Artificial (IA) ha pasado de ser un campo predominantemente teórico a convertirse en una tecnología desplegada en aplicaciones reales. Estos sistemas, especialmente los basados en redes neuronales profundas (DNNs), son hoy fundamentales para tareas complejas como el reconocimiento de imágenes o la toma autónoma de decisiones [1], [2], [3]. Esta capacidad sin precedentes tiene, sin embargo, un coste importante: requiere una gran potencia de cálculo, lo que hace imprescindible recurrir a aceleradores hardware especializados, como GPUs, TPUs o FPGAs, para alcanzar prestaciones en tiempo real en entornos con restricciones energéticas [4], [5], [6].

A medida que las DNNs abandonan el laboratorio y se integran en dominios críticos, como los sistemas avanzados de ayuda a la conducción (ADAS), el sector aeroespacial o la robótica remota, el objetivo deja de ser únicamente maximizar el rendimiento y pasa a ser garantizar la *confiabilidad* (*dependability*) [7], [8]. En estos contextos, un fallo transitorio deja de ser un simple error numérico y puede traducirse en consecuencias graves. Por ello, es necesario asegurar que estos sistemas mantengan un comportamiento fiable incluso en presencia de fallos [9], [10].

Entre las arquitecturas de DNNs, los *Vision Transformers* (ViT) se han consolidado como una alternativa muy competitiva frente a las redes convolucionales tradicionales (CNNs) en visión por computador [11]. Al tratar una imagen como una secuencia de bloques y modelar relaciones globales mediante mecanismos de atención, los ViT ofrecen ventajas claras en capacidad de representación. Sin embargo, esa misma estructura introduce operaciones y patrones de propagación de errores distintos a los estudiados previamente en CNNs. En consecuencia, sigue existiendo una necesidad clara de caracterizar cómo afectan los errores transitorios a esta familia de modelos y qué mecanismos de protección son realmente eficaces.

En este trabajo analizamos la arquitectura ViT-Base bajo inyección de fallos por *bit-flips* aleatorios en la representación IEEE 754 `binary32` de los tensores intermedios. Nuestro objetivo es doble: por un lado, identificar los puntos de mayor vulnerabilidad del modelo; por otro, evaluar estrategias prácticas de mitigación basadas exclusivamente en software. Más concretamente, las principales contribuciones del artículo son las siguientes:

- Se propone una metodología de inyección de fallos a nivel software sobre GPU, basada en extensiones de PyTorchFI y *kernels* CUDA personalizados, capaz de inyectar múltiples inversiones aleatorias de bits con control preciso sobre capas, posiciones y BER.
- Se presenta un mapa detallado de vulnerabilidad por componentes para ViT-Base, identificando las salidas de `LayerNorm` y `Linear` como los puntos más críticos frente a errores transitorios.
- Se evalúa el impacto del entrenamiento consciente de fallos (FAT) y se demuestra que, en el escenario considerado de errores aleatorios por *bit-flips*, esta estrategia no mejora de forma apreciable la resiliencia del modelo.
- Se valida experimentalmente una mitigación en inferencia basada en filtrado de valores anómalos (`NaN`, `Inf` y valores fuera de rango), mostrando mejoras sustanciales de fiabilidad.

El resto del artículo se organiza como sigue. La Sección II resume los conceptos fundamentales sobre Transformers, tolerancia a fallos y efectos de radiación. La Sección III revisa el estado del arte. La

[1]Dpto. de Tecnología Electrónica, Universidad Carlos III de Madrid, correo-e: `lfrias@ing.uc3m.es`.

[2]Dpto. de Ingeniería y Ciencia de Computadores, Universitat Jaume I de Castellón, correos-e: `{badia,leon}@uji.es`.

[3]Dpto. de Teoría de la Señal y Comunicaciones, Universidad Carlos III de Madrid, correo-e: `aamor@ing.uc3m.es`.

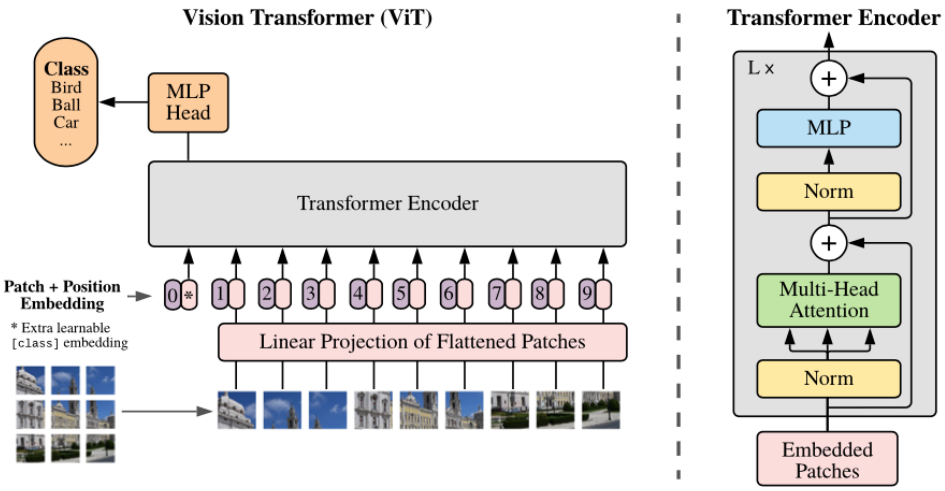


Fig. 1: Visión general de la arquitectura ViT [12].

Sección IV describe la metodología experimental y el marco de inyección de fallos. La Sección V presenta los resultados y su discusión. Finalmente, la Sección VI recoge las conclusiones y las líneas de trabajo futuro.

II. Fundamentos

A. Transformers y Vision Transformers

La arquitectura Transformer, introducida en [13], constituye el bloque básico de los ViT. Su principal innovación fue el uso de mecanismos de atención multi-cabeza para procesar secuencias de datos en paralelo, eliminando la dependencia de estructuras secuenciales como las redes recurrentes o, en ciertos contextos, las convoluciones. En el núcleo de esta arquitectura, el mecanismo de atención aprende relaciones entre los elementos de la secuencia (*tokens*) ponderando su relevancia relativa. Aunque la arquitectura Transformer completa incorpora un codificador y un decodificador, los ViT emplean esencialmente la parte codificadora.

Los ViT adaptan esta idea, originalmente concebida para Procesamiento del Lenguaje Natural (NLP), al análisis de imágenes [12]. La imagen se divide en bloques no solapados de tamaño fijo, que se tratan como *tokens* visuales. Cada bloque se linealiza, se proyecta en un espacio de *embeddings* y se combina con codificaciones posicionales para preservar la información espacial. Una vez que esta secuencia entra en el codificador Transformer, el modelo puede capturar relaciones globales entre regiones de la imagen, constituyendo una alternativa potente a las CNNs cuando se entrena con conjuntos de datos suficientemente grandes [14].

Las Figuras 1 y 2 ilustran la estructura general de ViT. El proceso comienza con un módulo de *Embedding* implementado con una capa lineal o totalmente conectada (`Linear`), que realiza la transformación afín $y = Wx + b$ para mapear los bloques a vectores de *embedding*. Después se añaden las codificaciones posicionales, operación esencial para recuperar la información espacial perdida al generar los bloques.

El núcleo de ViT está formado por una pila de capas *Transformer Encoder*. Cada capa incluye un módulo de atención multi-cabeza (MHA) y un bloque MLP. Antes de ambos se aplica una capa de normalización (`LayerNorm`) que estabiliza el aprendizaje manteniendo una media y una varianza consistentes. Dentro del bloque de atención, capas lineales generan las matrices *Query* (Q), *Key* (K) y *Value* (V). A continuación, una multiplicación matricial calcula las puntuaciones de atención QK^T, que se escalan típicamente por $\sqrt{d_k}$, y una función `Softmax` transforma dichas puntuaciones en una distribución de probabilidad. Una segunda multiplicación matricial combina esos pesos con los valores. Finalmente, las distintas cabezas se concatenan y pasan por una nueva proyección lineal.

El bloque MLP contiene dos capas `Linear` separadas por una activación `GELU`. Esta activación introduce la no linealidad necesaria para aprender patrones complejos. Además, se emplean capas `Dropout` con fines de regularización y conexiones residuales basadas en operaciones de suma, que facilitan el flujo del gradiente. El proceso concluye con la salida del *class token*, que atraviesa una cabeza MLP para producir los *logits* de clasificación.

B. Tolerancia a fallos en redes neuronales

A menudo se asume que la redundancia inherente de las DNNs absorbe perturbaciones hardware, pero numerosos trabajos muestran que dicha tolerancia no es uniforme [5]. La susceptibilidad depende de múltiples factores: topología de la red, tipo de capa, representación numérica y posición exacta del bit corrompido [16]. Además, los aceleradores hardware, apoyados en jerarquías profundas de memoria y en la reutilización intensiva de datos, pueden amplificar el efecto de fallos localizados.

Desde el punto de vista algorítmico, las redes sí poseen cierta capacidad de enmascaramiento. Activaciones como ReLU pueden amortiguar errores al anular valores negativos; operaciones como *max-pooling* o normalización también ayudan a estabilizar valores corrompidos. Sin embargo, esta resiliencia natu-

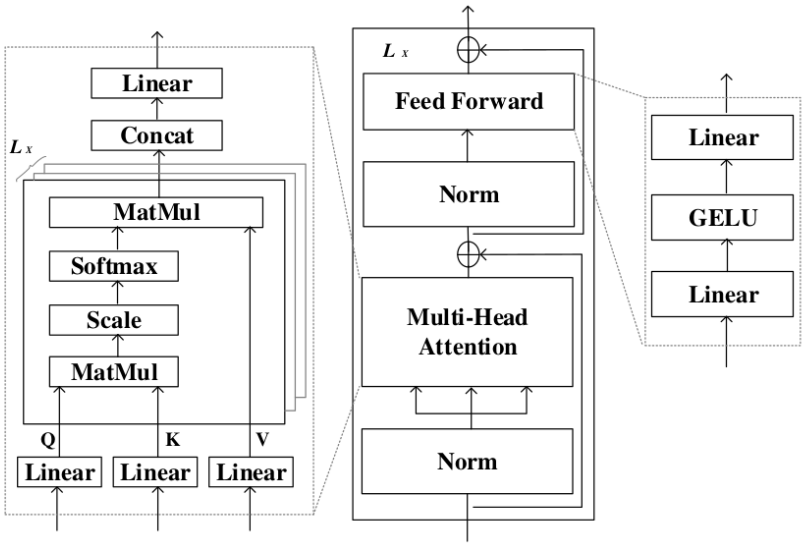


Fig. 2: Vista detallada del codificador Transformer de ViT [15].

ral suele ser incompleta [17]. La sensibilidad también está ligada a la función que desempeña cada componente: los errores en etapas tempranas de extracción de características suelen propagarse con más intensidad que los que ocurren en etapas finales.

En el caso concreto de los ViT, los estudios disponibles sugieren que las operaciones lineales de tipo GEMM o capas totalmente conectadas pueden ser relativamente robustas en algunos escenarios, mientras que las operaciones no lineales como `GELU` o `Softmax` resultan más frágiles y requieren mecanismos de protección específicos [15]. Aun así, la caracterización detallada por capas y por tipo de salida sigue siendo un problema abierto.

C. Efectos de radiación

Los errores transitorios inducidos por radiación constituyen una de las principales fuentes de fallos en dispositivos comerciales, incluso a nivel del suelo [18]. Cuando una partícula de alta energía impacta sobre el sustrato de silicio puede provocar un *Single Event Upset* (SEU) en memoria o un *Single Event Transient* (SET) en lógica. Si dicho transitorio queda retenido (*latched*), el efecto puede propagarse hasta la salida final del sistema.

Los fallos suelen clasificarse en tres grupos: errores enmascarados (sin efecto observable), corrupciones silenciosas de datos o *Silent Data Corruption* (SDC), y errores detectados no recuperables o *Detected Unrecoverable Errors* (DUE). Las SDC son especialmente peligrosas en redes neuronales, ya que pueden generar clasificaciones incorrectas sin ninguna señal de alarma. La continua miniaturización tecnológica agrava este riesgo, porque el elevado paralelismo de los aceleradores modernos permite que un único evento afecte simultáneamente a múltiples hilos de ejecución [19]. Los estudios mediante radiación muestran tasas elevadas de SDC en DNNs y sugieren que la simple protección ECC en memoria no siempre es suficiente, ya que muchas corrupciones críticas se originan fuera de las regiones protegidas [9], [20].

III. Trabajo relacionado

El rápido crecimiento de las DNNs ha impulsado una intensa actividad investigadora en torno a su fiabilidad. Diversas revisiones clasifican actualmente el problema de la tolerancia a fallos en aceleradores como GPUs, FPGAs y circuitos específicos de aplicación [17], [21], [5], [16], [19], [10]. La complejidad de la IA moderna exige estrategias de evaluación sofisticadas; de hecho, en [22] se distinguen seis enfoques principales: formal, basado en aplicaciones, basado en software, estructural/arquitectónico, físico e híbrido. Una evaluación rigurosa requiere combinar análisis teóricos, campañas de inyección de fallos y radiación con haces de partículas [19].

La vulnerabilidad de una DNN está fuertemente condicionada tanto por la arquitectura como por la naturaleza de los datos. Estudios seminales demostraron que la propagación de errores depende del tipo de dato, de sus valores, de los patrones de reutilización y del tipo de capa [17]. La precisión numérica también es determinante: algunos trabajos indican que los reales de 16 bits puede conducir a menores tasas FIT (*Failures in Time)* que los reales de 32 bits, en parte porque la vulnerabilidad suele crecer con el rango dinámico del formato [23], [24], [17]. Aunque gran parte de la literatura clásica se centró en CNNs, en los últimos años la atención se ha desplazado hacia los Transformers. Por ejemplo, irradiación con neutrones de ViT muestran tasas FIT significativamente superiores a las de modelos como ResNet50 [25]. Otros trabajos estudian cómo afectan distintas optimizaciones y representaciones numéricas tanto a modelos convolucionales como Transformer [20]. También se ha observado que un mayor número de cabezas de atención y de bloques residuales puede correlacionarse con mayores tasas de SDC, mientras que arquitecturas híbridas como los *Compact Convolutional Transformers* (CCT) presentan mejor fiabilidad [26].

En cuanto a mitigación, las propuestas abarcan desde soluciones puramente hardware hasta técnicas

software avanzadas. La *Algorithm-Based Fault Tolerance* (ABFT) ha evolucionado desde esquemas básicos de protección matricial hasta implementaciones conscientes de la arquitectura, capaces de corregir un elevado porcentaje de SDC críticas con un sobrecoste moderado [9], [27], [28]. Del mismo modo, las técnicas de detección de errores basadas en algoritmos (ABED), por ejemplo mediante *checksums*, han demostrado una detección completa en convoluciones de CNN con bajo coste temporal [29]. Otras estrategias incluyen duplicación selectiva de capas y detectores de síntomas (SEDs) que identifican rangos de activación anómalos [30], [17]. En Transformers, el recorte de valores y la protección basada en rangos también han mostrado resultados prometedores: restringir el rango de determinadas capas o funciones no lineales puede suprimir salidas anómalas y recuperar gran parte de la precisión perdida [31], [15]. Asimismo, se han propuesto nuevas funciones de activación como ReLUMax, que calculan dinámicamente umbrales de recorte sin añadir latencia [32].

Finalmente, las estrategias integradas hardware-software reconocen que muchas SDC críticas se originan en la ruta de datos de la GPU, en unidades de control o en lógica, y no solo en memoria [9]. Aunque el mecasnismo ECC reduce la tasa de SDC, su eficacia está limitada por huecos de cobertura y por su foco principal en errores de un solo bit, pudiendo incluso incrementar las DUE en ciertos casos [9], [22]. Otra alternativa prometedora es el *Fault-Aware Training*, que introduce ruido o fallos durante el entrenamiento para mejorar la fiabilidad de la red [19], [9]. No obstante, su eficacia depende críticamente de que las perturbaciones introducidas en entrenamiento representen bien los fallos reales. Dentro de este contexto, nuestro trabajo aporta una caracterización detallada de ViT-Base mediante inyección de fallos de alta fidelidad, evalúa FAT sobre Transformers y demuestra la efectividad práctica del filtrado de valores anómalos.

IV. Metodología de inyección de fallos

A. Diseño experimental

Se diseñó un marco experimental para analizar la vulnerabilidad de ViT frente a *bit-flips* aleatorios y para evaluar posibles estrategias de endurecimiento. El proceso se divide en dos fases bien diferenciadas.

La primera es la fase de entrenamiento, en la que un modelo no entrenado se convierte en un modelo ajustado con sus pesos finales y su precisión asociada. Durante esta fase, el entrenamiento puede realizarse sin inyección de fallos para construir una base de comparación, o bien con inyección de fallos sobre una o varias capas usando una BER seleccionada, con el objetivo de generar un modelo potencialmente más robusto. Aunque podrían modificarse múltiples parámetros, en la práctica se varían principalmente la BER y las capas afectadas, lo que facilita la comparación entre modelos.

La segunda fase consiste en analizar el comportamiento de los modelos bajo inyección durante la inferencia. A partir de un modelo ya entrenado, se ejecuta inferencia sobre el conjunto de prueba mientras se introducen fallos con una BER predeterminada. Repetir este proceso para BER crecientes permite determinar el umbral a partir del cual la precisión empieza a deteriorarse y estudiar su evolución. La comparación entre distintos modelos entrenados permite evaluar si se ha conseguido una mayor resiliencia. Como métrica principal se utiliza la precisión Top-1 sobre CIFAR-10, complementada con el recuento de valores anómalos (`NaN`, `Inf` y valores excesivamente grandes) para caracterizar el modo en que se manifiestan los fallos.

B. Metodología de prueba e inyección

Para la campaña de inyección de fallos se adaptó y amplió de forma significativa PyTorchFI [33], una herramienta de ejecución que permite perturbar modelos de PyTorch mediante *hooks* software sobre las salidas de las capas. En este trabajo se empleó como modelo de fallo el *bit-flip*, por considerarse representativa de errores transitorios como los SEU. Mientras que la versión base de PyTorchFI soporta *bit-flips* simples, nuestra investigación requería un mecanismo más fino y eficiente. Para ello se desarrollaron extensiones específicas, en particular *kernels* CUDA optimizados que manipulan directamente la representación IEEE 754 `binary32` en GPU. Esta implementación proporciona control preciso sobre la capa objetivo, el elemento del lote (*batch*) y la posición exacta del bit alterado. Además, se añadió la capacidad de inyectar múltiples *bit-flips* aleatorios distribuidos por toda la red, incluyendo selección aleatoria de capa y de posición intra-capa.

También se modificó el mecanismo de *bit-flip*. La función original realizaba cuantización, conversión a entero, complemento a dos y una serie de transformaciones inversas que introducían redondeos y podían causar fallos críticos cuando el valor del tensor excedía ciertos rangos. Para evitarlo, se implementó una nueva función que altera directamente la representación binaria del valor en coma flotante dentro de un *kernel* CUDA, eliminando transferencias innecesarias CPU–GPU y mejorando tanto la eficiencia como la precisión.

Durante el entrenamiento con fallos, las posiciones afectadas se vuelven a aleatorizar en cada lote. La inyección está activa únicamente durante la propagación hacia delante, mientras que se desactiva durante la retropropagación para no corromper el proceso de aprendizaje. Aun así, un *bit-flip* puede producir valores especiales como `NaN`, `Inf` o magnitudes fuera de rango capaces de detener el entrenamiento. Para garantizar que la campaña pudiera completarse, se aplicó un filtro que sustituye esos valores anómalos por cero. Aunque esta aproximación no replica perfectamente el comportamiento físico real, permite entrenar modelos bajo condiciones de fallo controladas.

En la fase de evaluación de modelos ya entrenados, por el contrario, las posiciones de fallo seleccionadas

permanecen fijas durante toda la época. Esto permite construir casos de prueba deterministas y comparar resultados entre experimentos. Para explorar distribuciones aleatorias distintas basta con cambiar la semilla y repetir el procedimiento.

La intensidad de fallo se controla mediante la BER, definida como

$$\text{BER} = \frac{\text{número de bit flips}}{\text{número total de bits en las capas objetivo}}.$$

Para obtener el número total de bits, se multiplican las dimensiones del tensor de salida por 32, dado que se trabaja con precisión simple. Los módulos de PyTorch objeto de inyección se corresponden con los principales componentes de ViT:

- `Linear`: proyección de entrada, proyecciones de atención (Q, K, V), redes *feed-forward* y cabeza de clasificación.
- `LayerNorm`: capas de normalización previas a atención y bloques *feed-forward*.
- `Dropout`: capas de regularización (activas solo durante entrenamiento).
- `GELU`: funciones de activación del bloque MLP.
- `Softmax`: normalización de pesos de atención.

Este enfoque modular permite tanto campañas de análisis de red completa como estudios dirigidos por tipo de componente.

C. Entorno computacional y configuración experimental

Todos los experimentos se realizaron sobre hardware NVIDIA dentro de contenedores Docker para asegurar reproducibilidad y coherencia en las dependencias software. Se utilizó PyTorch [34] como marco principal de aprendizaje profundo, aprovechando su grafo dinámico de computación y su aceleración nativa sobre GPU, esenciales para una campaña intensiva de inyección de fallos.

Se empleó la configuración ViT-Base como arquitectura de referencia para evaluar la resiliencia frente a errores. La implementación del modelo se basó en una referencia de código abierto ampliamente utilizada [35], similar a la arquitectura descrita en el trabajo original [12]. La configuración concreta utilizada consta de 12 bloques *Transformer Encoder*, dimensión de *embedding* 768 y 12 cabezas de atención en el mecanismo MHA, con aproximadamente 86 millones de parámetros. El `dropout` se fijó a 0.1 durante el entrenamiento.

Se utilizó el conjunto de datos CIFAR-10 [36], con 60000 imágenes en color distribuidas en 10 clases (50000 para entrenamiento y 10000 para prueba). La elección de este conjunto se debe a su equilibrio entre relevancia experimental y coste computacional. Al ser bastante más pequeño que ImageNet [37], reduce sensiblemente los tiempos de entrenamiento y validación, algo importante cuando el foco principal no es alcanzar la máxima precisión posible, sino realizar campañas iterativas amplias para estudiar tolerancia a fallos. Las imágenes se redimensionaron de 32×32 a 224×224 píxeles para ajustarse a los requisitos de entrada de ViT. Después se dividieron en bloques de 16×16, dando lugar a 196 bloques por imagen, que se linearizaron e integraron antes de entrar en el codificador Transformer.

V. Resultados experimentales

Esta sección recoge tres líneas principales de experimentación para evaluar la fiabilidad de ViT bajo inyección de fallos. En primer lugar, se entrenó un modelo base exclusivamente con el subconjunto de entrenamiento de CIFAR-10 durante 100 épocas, utilizando el optimizador Adam, pérdida *Cross-Entropy* y una tasa de aprendizaje inicial de 10^{-4}. El modelo resultante alcanzó una precisión Top-1 del 86,48 % sobre el conjunto de prueba de CIFAR-10. Disponer de un modelo de referencia verificable es esencial para poder atribuir la degradación observada a los fallos inducidos, y no a limitaciones del entrenamiento o de la propia arquitectura.

Posteriormente se analizó la vulnerabilidad del modelo base, inyectando *bit-flips* en distintos puntos durante la inferencia. Al variar sistemáticamente tanto la BER como el punto de inyección, fue posible caracterizar la caída de precisión y localizar las zonas más sensibles de la arquitectura. A partir de esta caracterización se comparó el comportamiento del modelo original con el de modelos entrenados mediante FAT. Finalmente, se analizó la aparición de valores anómalos (`NaN`, `Inf` o magnitudes extremadamente grandes) y el efecto de su filtrado sobre la precisión final y la estabilidad del proceso de inferencia.

A. Evaluación de vulnerabilidad bajo inversiones aleatorias de bit

Los primeros experimentos se centraron en analizar la vulnerabilidad del modelo base. Se ejecutaron múltiples inferencias sobre el conjunto de prueba de CIFAR-10 sometidas a inyección de fallos, variando la BER en un rango amplio, desde 10^{-11} hasta 10^{-6}. Inicialmente, los fallos se distribuyeron de forma uniforme sobre las salidas de todas las capas para capturar la sensibilidad global del modelo. Posteriormente, la campaña se refinó concentrando los fallos en un único tipo de capa cada vez, de forma que pudiera cuantificarse su contribución específica a la degradación global.

A.1 Análisis global con variación de BER

La evaluación global del modelo base, resumida en la Figura 3, muestra tres regímenes bien diferenciados ante el incremento de la intensidad de fallo:

- **Umbral de resiliencia.** Por debajo de una BER de 10^{-9}, la degradación de precisión es despreciable ($\leq 0,6\,\%$), lo que sugiere la existencia de mecanismos internos de tolerancia derivados de la naturaleza distribuida de la atención y de la redundancia presente en las representaciones de alta dimensión [38], [39], [40].
- **Degradación progresiva.** Entre 10^{-9} y 10^{-7} aparece una pérdida sistemática de precisión,

Tabla I: Hardware experimental empleado.

	Volta	**NASP C03**
CPU	Intel(R) Xeon(R) Gold 6126 a 2.60GHz	Intel(R) Xeon(R) Gold 6418H a 2.10GHz
GPU	NVIDIA Tesla V100 Arquitectura NVIDIA Volta 640 Tensor Cores 5120 CUDA Cores 32 GB de memoria GPU Interfaz PCIe Gen3	NVIDIA H100 NVL Arquitectura NVIDIA Hopper 1056 Tensor Cores 33792 CUDA Cores 94 GB de memoria GPU Interfaz PCIe de doble ranura
Memoria	64 GB DDR4	2 TB DDR5

con degradaciones aproximadas del 10 %, 20 % y 30 % para BER de 2×10^{-8}, 4×10^{-8} y $6,5\times10^{-8}$, respectivamente. Este régimen es especialmente relevante porque permite estimar el impacto esperado de los fallos en escenarios de despliegue.

- **Fallo catastrófico.** Para BER $\geq 10^{-6}$, la precisión cae hasta aproximadamente el 10 %, equivalente a una clasificación aleatoria entre las 10 clases de CIFAR-10. Este punto marca el límite operativo de un ViT sin protección en entornos con radiación.

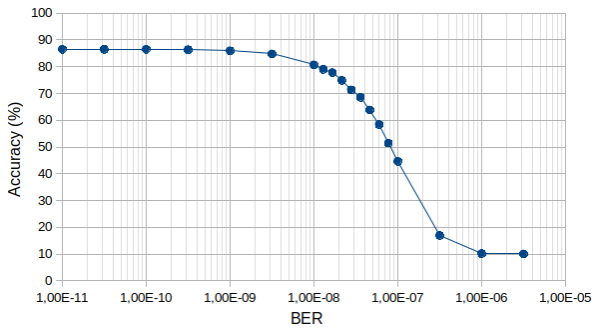


Fig. 3: Precisión del modelo base frente al incremento de BER.

A.2 Vulnerabilidad específica por capa

Para identificar qué salidas de capa eran más vulnerables, se realizaron campañas dirigidas donde los *bit-flips* se concentraban exclusivamente en un tipo de capa cada vez. El número total de inversiones se mantuvo constante, por lo que el valor de BER, de acuerdo con su definición, permanecía invariante; lo que cambiaba era la concentración de los errores dentro de un subconjunto concreto de la red.

El análisis por componentes, mostrado en la Figura 4, revela una clara heterogeneidad en la sensibilidad a fallos. Las salidas de `LayerNorm` y `Linear` son las que provocan el mayor impacto sobre la precisión. En el caso de `LayerNorm`, las salidas corrompidas se sitúan inmediatamente antes de los bloques de atención y de las capas *feed-forward*, por lo que sus perturbaciones se propagan y pueden amplificarse en capas posteriores con pesos aprendidos elevados. Las capas `Linear`, por su parte, participan en funciones críticas del modelo: proyección de entrada, generación de Q, K y V, transformaciones *feed-forward* y clasificación final. Su vulnerabilidad deriva de su posición funcional dentro de la arquitectura; un error en su salida puede modificar directamente la predicción o verse amplificado por las operaciones posteriores.

Aunque `Dropout` permanece inactiva en inferencia, su posición permite que valores ya corrompidos alcancen capas posteriores `Linear` y `LayerNorm`; por ello, la vulnerabilidad observada en esta categoría refleja más la estructura de la red que la sensibilidad intrínseca del operador. Finalmente, las funciones no lineales `GELU` y `Softmax` muestran una vulnerabilidad relativamente menor en sus salidas que otros componentes, lo que sugiere cierta atenuación natural del error. Sin embargo, dicha atenuación no basta para evitar la degradación de precisión.

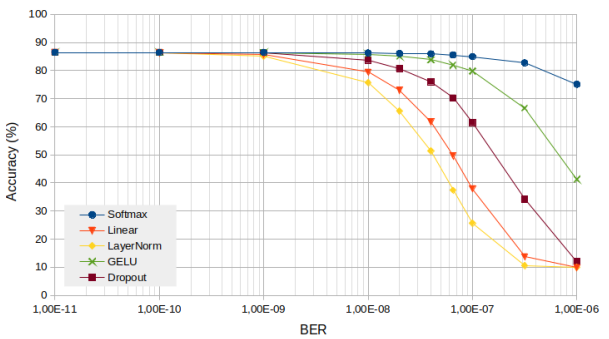


Fig. 4: Precisión del modelo base por tipo de capa frente al incremento de BER.

B. Fault-Aware Training como técnica de mejora de tolerancia

En los experimentos con FAT, el modelo se entrenó mientras se inyectaban fallos activamente con una BER de $6,5\times10^{-8}$, valor elegido porque en la campaña de vulnerabilidad producía una caída aproximada del 30 % en precisión. Se evaluaron dos estrategias: inyectar sobre todos los tipos de capa simultáneamente e inyectar únicamente sobre capas `Linear`, dado que son abundantes, ocupan posiciones funcionales críticas y sus salidas se mostraron especialmente vulnerables. Las posiciones de fallo se aleatorizaban en cada lote para impedir que el modelo simplemente memorizase un patrón fijo de corrupción.

La hipótesis inicial era que FAT debería traducirse en curvas con mayor precisión para una misma BER. Sin embargo, los resultados mostrados en la Figura 5 contradicen esa expectativa: los modelos entrenados con inyección de fallos, tanto global como selectiva sobre `Linear`, presentan curvas de degradación prácticamente paralelas a la del modelo base cuando se someten a inyección durante la inferencia.

Además, el entrenamiento con fallos reduce la precisión final del modelo limpio: los modelos con FAT alcanzan aproximadamente 85 % y 83,5 %, frente al 86,48 % del modelo base. Este descenso indica que la inyección de fallos interfiere con la convergencia normal del entrenamiento y ese coste no se compensa con una mejora apreciable en fiabilidad. En BER elevadas tampoco se observa una ganancia real de fiabilidad, lo que sugiere una limitación fundamental de FAT en escenarios con *bit-flips* aleatorios. Al variar aleatoriamente las posiciones de fallo de lote a lote, el optimizador no consigue identificar un patrón estable de perturbación y termina interpretando el fenómeno como ruido no estructurado de alta varianza. En consecuencia, el modelo no desarrolla mecanismos específicos de adaptación o enmascaramiento útiles para una tolerancia generalizada.

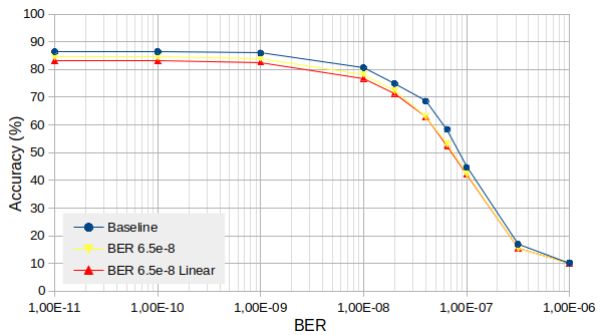


Fig. 5: Precisión del modelo base y de los modelos con FAT frente al incremento de BER.

C. *Técnicas de mitigación en inferencia*

Durante la evaluación de vulnerabilidad no solo se registró la precisión, sino también el número de valores especiales generados tras la inyección: `NaN`, infinitos o valores fuera de rango. Se observó que una causa principal del colapso rápido de precisión a BER altas es precisamente la generación y propagación de estos valores anómalos. Dichos artefactos aparecen cuando los *bit-flips* afecta al signo o al exponente de la representación IEEE 754 `binary32`, creando estados numéricos inestables que se propagan catastróficamente a través de operaciones matriciales.

El análisis cuantitativo confirmó una correlación directa y pronunciada entre el aumento de BER y la frecuencia de anomalías numéricas durante la inferencia. La Figura 6 ilustra esta relación y justifica la necesidad de mitigaciones en tiempo de ejecución.

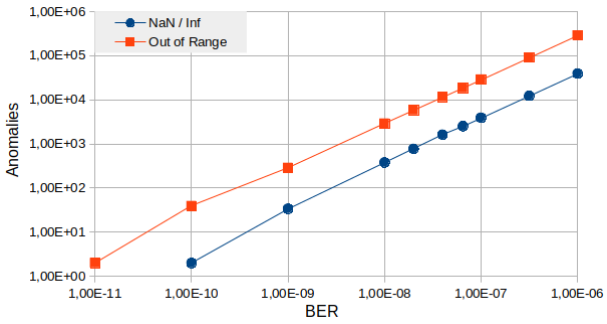


Fig. 6: Anomalías numéricas frente al incremento de BER.

Se implementaron mecanismos de filtrado para tratar valores `NaN`, infinitos y magnitudes excesivamente grandes producidas por la inyección. Los valores que superan un umbral prefijado, así como `NaN` e `Inf`, se detectan y sustituyen. Se estudiaron dos formas de definir el umbral: un valor fijo y un valor variable dependiente del tipo de capa. El umbral fijo inicial se estableció en 10^4, elegido empíricamente a partir del rango típico observado en las activaciones de ViT. En condiciones normales de inferencia, la mayoría de activaciones se mantiene claramente por debajo de ese valor; por tanto, magnitudes superiores a 10^4 son raras y es razonable asociarlas a anomalías numéricas inducidas por fallos.

Como alternativa, se emplearon umbrales específicos por capa obtenidos a partir de ejecuciones preliminares de inferencia: para cada tipo de capa se tomó el valor máximo observado y se incrementó un 5 %, salvo en `Softmax`, donde se mantuvo 1 como valor máximo. Los valores anómalos se sustituyeron o bien por cero o bien por el propio valor umbral. En total se analizaron tres variantes:

- Umbral fijo (10^4) con sustitución por cero.
- Umbral variable por capa con sustitución por cero.
- Umbral variable por capa con saturación al valor umbral.

Los resultados, mostrados en la Figura 7, revelan un potencial muy elevado de estas técnicas de posprocesado para mejorar la tolerancia a radiación en ViT. El filtrado con umbral fijo y sustitución por cero proporciona una mejora sustancial de precisión en todo el rango de BER analizado y amplía el rango operativo antes de que se produzca degradación severa. La variante con umbrales específicos por capa ofrece el mejor comportamiento, superando ligeramente al umbral fijo. Esta mejora refleja la heterogeneidad de distribuciones de valores entre capas y la conveniencia de estrategias adaptadas a cada componente. Además, la sustitución por cero supera sistemáticamente a la saturación en el umbral, lo que indica que eliminar completamente los valores corrompidos es preferible a forzarlos a permanecer en un valor extremo de frontera.

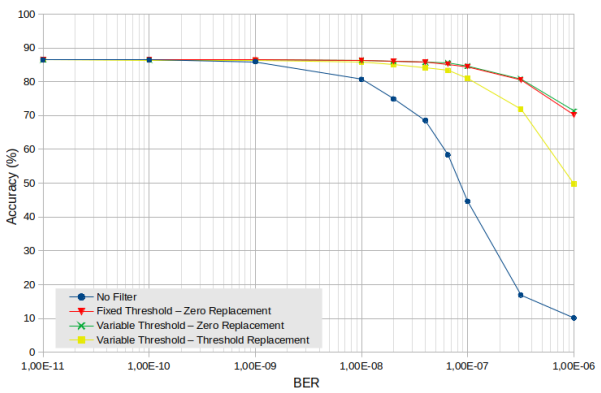


Fig. 7: Precisión del modelo base filtrado frente al incremento de BER.

VI. Conclusiones

El análisis exhaustivo de inyección de fallos realizado sobre ViT-Base, simulando fallos transitorios como *bit-flips* en la representación IEEE 754 `binary32`, permite extraer varias conclusiones relevantes. En primer lugar, la fiabilidad del modelo depende fuertemente de su estabilidad numérica. El comportamiento permanece esencialmente estable solo hasta una BER relativamente baja, del orden de 1×10^{-9}; a partir de ese punto la precisión cae rápidamente, hasta alcanzar un comportamiento equivalente a clasificación aleatoria para BER de 1×10^{-6}. Esto define con claridad un límite funcional gobernado por la integridad numérica del cómputo.

En segundo lugar, el mapa de vulnerabilidad identifica con nitidez una debilidad arquitectónica: las salidas de las capas de normalización (`LayerNorm`) y de las capas totalmente conectadas (`Linear`) constituyen la principal fuente de amplificación del error. En contraste, aunque algunas operaciones no lineales muestran cierta atenuación natural, esta no es suficiente para proteger el modelo de forma efectiva.

En tercer lugar, los intentos de mejorar la resiliencia mediante FAT no produjeron los beneficios esperados. Incluso cuando el entrenamiento se focalizó sobre las capas más vulnerables, no se observó una mejora real en la fiabilidad frente a fallos. Este resultado sugiere una limitación de fondo: el optimizador no puede extraer patrones anómalos significativos a partir de ruido aleatorio por *bit-flips*, lo que impide desarrollar defensas eficaces basadas únicamente en gradiente.

Por último, la estrategia más práctica y efectiva para incrementar la fiabilidad del modelo fue la incorporación de un filtro numérico en tiempo de ejecución. El uso de umbrales adaptativos por capa combinados con sustitución por cero estabiliza significativamente la inferencia al impedir la propagación de valores especiales inducidos por fallo, como `NaN`, `Inf` o valores fuera de rango. Aunque esta técnica proporciona una mejora importante, la vulnerabilidad residual indica que la tolerancia completa a radiación requerirá probablemente una estrategia múltiple que combine protección algorítmica, arquitectónica y hardware.

Como trabajo futuro, resulta natural explorar endurecimiento arquitectónico selectivo, variantes de Transformer menos dependientes de operaciones vulnerables y estrategias híbridas de protección. También será importante complementar este estudio con campañas experimentales de radiación que permitan validar las conclusiones obtenidas mediante inyección de fallos software.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia e Innovación de España a través de los proyectos PID2023-146569NB-C22 y PID2022-137048OA-C43, financiados por MCIN/AEI/10.13039/501100011033 y por el FEDER, así como por los proyectos TED2021-131401B-C21 y TED2021-131401A-C22. Asimismo, esta investigación ha recibido financiación del programa de actividades de I+D con referencia TEC-2024/COM-360 y acrónimo DISCO6G-CM, concedido por la Comunidad de Madrid a través de la Dirección General de Investigación e Innovación Tecnológica mediante la Orden 2402/2024.

Referencias

[1] William Grant Hatcher and Wei Yu, "A survey of deep learning: Platforms, applications and emerging research trends," *IEEE Access*, vol. 6, pp. 24411–24432, 2018.

[2] Jose A. Belloch, José M. Badía, Francisco D. Igual, Alberto Gonzalez, and Enrique S. Quintana-Ortí, "Optimized fundamental signal processing operations for energy minimization on heterogeneous mobile devices," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 5, pp. 1614–1627, 2018.

[3] Jose A. Belloch, German Ramos, Jose M. Badia, and Maximo Cobos, "An efficient implementation of parallel parametric hrtf models for binaural sound synthesis in mobile multimedia," *IEEE Access*, vol. 8, pp. 49562–49573, 2020.

[4] Sparsh Mittal, "A survey on optimized implementation of deep learning models on the nvidia jetson platform," *Journal of Systems Architecture*, vol. 97, pp. 428–442, Jan. 2019.

[5] Sparsh Mittal, "A survey on modeling and improving reliability of DNN algorithms and accelerators," *Journal of Systems Architecture*, vol. 104, Art. no. 101689, Mar. 2020.

[6] Jon Perez-Cerrolaza, Jaume Abella, Leonidas Kosmidis, Alejandro J Calderon, Francisco Cazorla, and Jose Luis Flores, "GPU devices for safety-critical systems: A survey," *ACM Computing Surveys*, vol. 55, no. 7, Art. no. 147, Dec. 2022.

[7] ISO, "ISO 26262-9:2018 Road vehicles — Functional safety — Part 9: Automotive safety integrity level (ASIL)," `https://www.iso.org/standard/68391.html`, 2018.

[8] Xiaojie Wang, Beibei Wang, Yu Wu, Zhaolong Ning, Song Guo, and Fei Richard Yu, "A survey on trustworthy edge intelligence: From security and reliability to transparency and sustainability," *IEEE Communications Surveys & Tutorials*, Aug. 2024, to be published.

[9] Fernando Fernandes Dos Santos and Paolo Rech, "Challenges in assessing and improving deep neural networks reliability," *IEEE Design & Test*, 2024.

[10] Mohammad Hasan Ahmadilivani, Mahdi Taheri, Jaan Raik, Masoud Daneshtalab, and Maksim Jenihhin, "A systematic literature review on hardware reliability assessment methods for deep neural networks," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–39, 2024.

[11] Yakoub Bazi, Laila Bashmal, Mohamad M Al Rahhal, Reham Al Dayil, and Naif Al Ajlan, "Vision transformers for remote sensing image classification," *Remote Sensing*, vol. 13, no. 3, pp. 516, 2021.

[12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al., "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[14] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah, "Transformers in vision: A survey," *ACM computing surveys (CSUR)*, vol. 54, no. 10s, pp. 1–41, 2022.

[15] Xinghua Xue, Cheng Liu, Ying Wang, Bing Yang, Tao Luo, Lei Zhang, Huawei Li, and Xiaowei Li, "Reliability analysis of vision transformers," *arXiv preprint arXiv:2302.10468*, 2023.

[16] Younis Ibrahim, Haibin Wang, Junyang Liu, Jinghe Wei, Li Chen, Paolo Rech, Khalid Adam, and Gang Guo, "Soft errors in DNN accelerators: A comprehensive review," *Microelectronics Reliability*, vol. 115, Art. no. 113969, Dec. 2020.

[17] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and

Stephen W Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[18] Robert C Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and materials reliability*, vol. 5, no. 3, pp. 305–316, 2005.

[19] Paolo Rech, "Artificial Neural Networks for Space and Safety-Critical Applications: Reliability Issues and Potential Solutions," *IEEE Transactions on Nuclear Science*, vol. 71, no. 4, pp. 377–404, Apr. 2024, DOI: 10.1109/TNS.2024.3349956.

[20] Jose M Badia, Ignacio Martin-Salinas, German Leon, Adrian Amor-Martin, Lester Frias-Dominguez, Jose A Belloch, Mario Garcia-Valderas, Almudena Lindoso, Carlo Cazzaniga, and Luis Entrena, "Reliability of vision transformers and cnns on edge ai systems under neutron radiation," *IEEE Transactions on Nuclear Science*, 2025.

[21] Cesar Torres-Huitzil and Bernard Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17322–17341, Aug. 2017.

[22] Dennis Gnad, Martin Gotthard, Jonas Krautter, Angeliki Kritikakou, Vincent Meyers, Paolo Rech, Josie E Rodriguez Condia, Annachiara Ruospo, Ernesto Sanchez, Fernando Fernandes Dos Santos, et al., "Reliability and security of ai hardware," in *2024 IEEE European Test Symposium (ETS)*. IEEE, 2024, pp. 1–10.

[23] Fernando Fernandes dos Santos, Caio Lunardi, Daniel Oliveira, Fabiano Libano, and Paolo Rech, "Reliability evaluation of mixed-precision architectures," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 238–249.

[24] Fernando Fernandes dos Santos, Philippe Navaux, Luigi Carro, and Paolo Rech, "Impact of reduced precision in the reliability of deep neural networks for object detection," in *2019 IEEE European Test Symposium (ETS)*. IEEE, 2019, pp. 1–6.

[25] Fernando Fernandes dos Santos, Paolo Rech, Angeliki Kritikakou, and Olivier Sentieys, "Neutron-induced error rate of vision transformer models on GPUs," in *RADECS-RADiation and its Effects on Components and Systems Conference, hal-04124814*, Toulouse, France, 25–29 Sep., 2023.

[26] Pablo R. Bodmann, Christopher Frost, and Paolo Rech, "Vision Transformer Reliability Evaluation on the Coral Edge TPU," in *NSREC*, Ottawa, Canada, 22–26 Jul., 2024.

[27] Aurick Qiao, Bryon Aragam, Bingjing Zhang, and Eric Xing, "Fault tolerance in iterative-convergent machine learning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 5220–5230.

[28] Fernando Fernandes dos Santos, Pedro Foletto Pimenta, Caio Lunardi, Lucas Draghetti, Luigi Carro, David Kaeli, and Paolo Rech, "Analyzing and increasing the reliability of convolutional neural networks on gpus," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, November 2019.

[29] Siva Kumar Sastry Hari, Michael B Sullivan, Timothy Tsai, and Stephen W Keckler, "Making convolutions resilient via algorithm-based error detection techniques," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2546–2558, 2022.

[30] Fernando Fernandes dos Santos, Luigi Carro, and Paolo Rech, "Kernel and layer vulnerability factor to evaluate object detection reliability in GPUs," *IET Computers & Digital Techniques*, vol. 13, no. 3, pp. 178–186, Dec. 2019.

[31] Junchao Chen, Giuseppe Esposito, Fernando Fernandes dos Santos, Juan-David Guerrero-Balaguera, Angeliki Kritikakou, Milos Krstic, Robert Limas Sierra, Josie E Rodriguez Condia, Matteo Sonza Reorda, Marcello Traiola, et al., "Reliability assessment of large dnn models: Trading off performance and accuracy," in *2024 IFIP/IEEE 32nd International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2024, pp. 1–10.

[32] Leonardo Iurada, Niccolò Cavagnero, Fernando Fernandes Dos Santos, Giuseppe Averta, Paolo Rech, and Tatiana Tommasi, "Transient fault tolerant semantic segmentation for autonomous driving," *arXiv preprint arXiv:2408.16952*, 2024.

[33] PyTorchFI Developers, "PyTorchFI: A Fault Injection Framework for PyTorch," Project Website, 2024, Available at: `https://www.pytorchfi.dev`.

[34] PyTorch Developers, "PyTorch," Website and open source project, 2024, Available at: `https://pytorch.org/projects/pytorch/`.

[35] lucidrains, "vit-pytorch: Implementation of Vision Transformer in PyTorch," GitHub Repository, 2024, Available at: `https://github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit.py`.

[36] Alex Krizhevsky, Geoffrey Hinton, et al., "Learning multiple layers of features from tiny images," 2009.

[37] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2009, pp. 248–255.

[38] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov, "Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned," *arXiv preprint arXiv:1905.09418*, 2019.

[39] Paul Michel, Omer Levy, and Graham Neubig, "Are sixteen heads really better than one?," *Advances in neural information processing systems*, vol. 32, 2019.

[40] Vedant Nanda, Till Speicher, John Dickerson, Krishna Gummadi, Soheil Feizi, and Adrian Weller, "Diffused redundancy in pre-trained representations," *Advances in Neural Information Processing Systems*, vol. 36, pp. 4055–4079, 2023.

Agentes LLM periféricos para la gestión autónoma del mantenimiento en infraestructura IoT crítica

Ángela Gijón[1], Félix Jesús Villanueva[1], Fernando Rincón[1], Cristina Bolaños[1], Xavier del Toro[1], Juan Carlos López[1]

Resumen— **El mantenimiento de infraestructuras IoT (*Internet of Things*) críticas de seguridad, como el alumbrado de emergencia, requiere la interpretación de telemetría técnica compleja y la generación de prescripciones accionables para el personal de campo, una tarea que tradicionalmente exige criterio experto y es difícilmente escalable. Este trabajo presenta ELIA (*Emergency Luminaires Intelligent Agent*), un sistema multiagente que combina un motor de reglas determinista con modelos de lenguaje de gran escala ejecutados íntegramente en local, siguiendo el paradigma neuro-simbólico para garantizar la auditabilidad de las decisiones críticas sin depender de infraestructura *cloud*. Se describe la arquitectura del sistema, basada en dos agentes implementados con LangGraph, uno de los cuales integra Recuperación Aumentada por Generación (RAG, *Retrieval-Augmented Generation*) sobre el historial de incidencias, y se evalúan comparativamente 15 modelos de lenguaje de gran escala (LLM, *Large Language Models*) cuantizados mediante un marco de evaluación en tres dimensiones que combina métricas de rendimiento, evaluación experta con rúbrica ponderada y *LLM-as-a-Judge*. Los resultados experimentales demuestran que el paradigma *local-first* es viable para este dominio: el modelo `qwen2.5:14b` alcanza cobertura completa sobre 38 incidencias posibles con una latencia media de 4.45 s y una puntuación de fidelidad de 7.90 sobre 10, constituyendo el punto de operación óptimo entre eficiencia y calidad de razonamiento. Tras una comparativa de los resultados obtenidos en los distintos modelos se constata además que el tamaño del modelo no predice el rendimiento en dominio específico y que las arquitecturas de razonamiento explícito presentan incompatibilidades con los requisitos de salida estructurada de los *pipelines* agénticos.**

Palabras clave— **LLM, Edge computing, IoT, Agentes, RAG, Mantenimiento predictivo**

I. Introducción

El alumbrado de emergencia constituye un elemento crítico de seguridad en edificios públicos. En España, su instalación y mantenimiento están regulados por el Código Técnico de la Edificación (CTE), el Reglamento Electrotécnico de Baja Tensión (REBT, ITC-BT-28) y las normas armonizadas UNE-EN 50172 y UNE-EN 62034 [1–4], que exigen pruebas funcionales mensuales, ensayos de autonomía anuales y el registro sistemático de resultados por dispositivo. El incumplimiento no es solo una infracción administrativa: una luminaria que falla en el momento de una evacuación puede tener consecuencias fatales. Sin embargo, la inspección física unidad por unidad es costosa, propensa a errores de registro y difícilmente escalable. Los sistemas autotest estandarizados (UNE-EN 62034) resuelven parcialmente este problema al automatizar las pruebas y generar *logs* de estado por dispositivo, pero la interpretación de dichos *logs*, especialmente cuando múltiples luminarias presentan anomalías simultáneas o recurrentes, sigue requiriendo criterio técnico experto para determinar causas raíz, priorizar actuaciones y documentar evidencias de cumplimiento normativo.

La proliferación de los modelos de lenguaje de gran escala (LLMs, Large Language Models) ha abierto nuevas posibilidades para la automatización del razonamiento técnico en tareas de mantenimiento industrial [5], pero su despliegue mediante APIs (*Application Programming Interfaces*) externas presenta tres barreras estructurales en entornos críticos: (1) **latencia de red**, incompatible con sistemas de seguridad donde el diagnóstico debe ser inmediato; (2) **coste por inferencia**, que escala linealmente con el volumen de telemetría procesada; y (3) **fuga de datos**, dado que los *logs* industriales contienen información sensible sobre la infraestructura física del edificio que no debe abandonar el perímetro de la instalación [6]. La disponibilidad creciente de modelos ejecutables en local mediante herramientas como Ollama[1] sugiere una oportunidad: desplegar *pipelines* de diagnóstico inteligente directamente en el *hardware* de campo donde se genera la telemetría, eliminando dependencias externas y garantizando la privacidad. La hipótesis que motiva este trabajo sostiene que es viable ejecutar un pipeline de razonamiento LLM íntegramente en local para el diagnóstico de infraestructura crítica sin sacrificar la calidad del análisis, identificando el punto de operación óptimo en el compromiso entre latencia, recursos computacionales y precisión diagnóstica en entornos industriales reales [7].

En este contexto, presentamos **ELIA** (*Emergency Luminaires Intelligent Agent*), un sistema de detección y análisis de incidencias en luminarias de emergencia que opera completamente en local (paradigma *local-first*). La arquitectura combina un motor de reglas determinista, responsable de la clasificación de severidad y asignación de responsables mediante lógica booleana inspeccionable, con un LLM local para el razonamiento sobre causas raíz y la generación de prescripciones técnicas, mediante *pipelines* de ejecución condicional con LangGraph[2], integran-

[1]Escuela Superior de Informática (Ciudad Real), Universidad de Castilla-La Mancha

[1] `https://ollama.com/`
[2] `https://github.com/langchain-ai/langgraph`

do Recuperación Aumentada por Generación (RAG, *Retrieval-Augmented Generation*) sobre el historial de incidencias para el razonamiento contextual en recurrencias [8]. Esta separación entre lógica determinista y razonamiento probabilístico garantiza que los atributos críticos de severidad y responsable sean siempre reproducibles, mientras el LLM se emplea exclusivamente donde aporta valor diferencial.

Las contribuciones principales son: (1) la arquitectura de ELIA, desplegable íntegramente en *hardware* local con separación determinista/LLM conforme a la normativa citada; (2) un banco de pruebas con siete casos de uso y la evaluación comparativa de quince modelos LLM mediante un marco tri-nivel: rendimiento computacional, rúbrica experta ponderada y *LLM-as-a-Judge*; y (3) el análisis del equilibrio entre calidad, latencia y recursos, identificando el perfil óptimo para entornos industriales con recursos limitados.

El resto del artículo se organiza como sigue. La Sección II revisa el estado del arte en monitorización de infraestructura crítica, diagnóstico de fallos mediante LLMs, despliegue en *edge*, *pipelines* agénticos con RAG y metodologías de evaluación de modelos de lenguaje. La Sección III describe la metodología y arquitectura del sistema. La Sección IV presenta el entorno experimental y los resultados del *benchmark*. La Sección V recoge las conclusiones y líneas de trabajo futuro.

II. Estado del arte

La monitorización automática de infraestructuras en edificios ha experimentado un avance significativo con la proliferación de dispositivos IoT (*Internet of Things*) capaces de generar telemetría continua sobre el estado de sus componentes [9]. En el ámbito de los sistemas de seguridad, esta capacidad se materializa en nodos de borde (*edge nodes*) que reportan periódicamente su estado mediante *logs* estructurados (típicamente NDJSON (*Newline Delimited JavaScript Object Notation*) o tramas binarias), permitiendo detectar anomalías sin necesidad de inspección física. Sin embargo, la literatura sobre detección y diagnóstico automático de fallos (FDD, del inglés *Fault Detection and Diagnosis*) en edificios pone de manifiesto una brecha persistente: mientras que los enfoques basados en reglas y aprendizaje automático han demostrado eficacia para detectar la presencia de un fallo, la identificación de su causa raíz, la correlación semántica de anomalías concurrentes en múltiples dispositivos y la generación de prescripciones accionables para el personal técnico siguen siendo retos abiertos [10]. Esta limitación es especialmente crítica en sistemas de seguridad, como el alumbrado de emergencia, donde un diagnóstico incompleto puede derivar en incumplimiento normativo o en una actuación incorrecta ante una emergencia real. Desde la perspectiva del mantenimiento industrial, esta brecha impacta directamente en dos métricas clave: el MTTR (*Mean Time To Repair*), que se incrementa sin prescripciones accionables que guíen al personal técnico, y el MTTF (*Mean Time To Failure*), cuya reducción se ve comprometida por la incapacidad de detectar patrones recurrentes y anticipar fallos sistémicos. Reducir ambas métricas en infraestructuras críticas de seguridad es el objetivo operativo que motiva el diseño de ELIA.

Frente a las limitaciones de los sistemas FDD tradicionales para razonar sobre causas raíz, los modelos de lenguaje de gran escala han emergido como una alternativa prometedora. A diferencia de los clasificadores estadísticos, que asocian patrones de datos a categorías predefinidas, un LLM puede procesar telemetría no estructurada y generar una explicación causal sobre el origen del fallo y las acciones correctivas necesarias. Esta capacidad de razonamiento contextual ha motivado su aplicación en entornos operativos reales: trabajos recientes en el ámbito de los sistemas de computación demuestran que los LLMs pueden analizar logs de incidencias complejas y proponer la causa raíz con una precisión comparable a la de expertos humanos [11]. En el ámbito del mantenimiento industrial de equipamiento físico, se ha demostrado que arquitecturas como la propuesta por Qaid et al. [12] superan las barreras de interpretabilidad de los modelos de aprendizaje profundo convencionales al traducir señales técnicas en prescripciones accionables para el personal de campo. Sin embargo, la dependencia de la práctica totalidad de estos sistemas de APIs en la nube los hace inviables para las infraestructuras críticas descritas anteriormente.

La arquitectura predominante en el diagnóstico mediante LLM se basa en un paradigma de inferencia externa (*Cloud-based inference*). No obstante, este modelo centralizado introduce limitaciones estructurales críticas en entornos industriales. En términos de latencia, la inferencia remota está sujeta a la variabilidad de la red y a colas de espera en el servidor que elevan el tiempo de respuesta a varios segundos, frente a la latencia determinista y el procesamiento inmediato alcanzable en local [13]. En cuanto al coste operativo, análisis empíricos demuestran que el despliegue local de modelos de código abierto puede resultar entre diez y veinte veces más económico por millón de *tokens* que los servicios *cloud*, con un coste de inferencia predecible y escalable en función del *hardware* disponible [14]. Finalmente, la soberanía del dato constituye una barrera infranqueable: los *logs* de infraestructura crítica contienen detalles sobre la topología física de las instalaciones cuya transmisión al exterior compromete la seguridad y el cumplimiento normativo [6]. Estos tres factores posicionan a la inferencia local como el único paradigma viable para sistemas de seguridad crítica.

La respuesta a las limitaciones del paradigma *cloud* ha impulsado una línea de investigación centrada en hacer viable la inferencia LLM directamente sobre *hardware* de borde. La técnica clave en este ámbito es la cuantización post-entrenamiento (PTQ, *Post-Training Quantization*), que reduce la precisión numérica de los pesos del modelo, habitualmente de representaciones de 16 bits en coma flotante a for-

matos de 4 u 8 bits enteros, permitiendo ejecutar modelos de miles de millones de parámetros en dispositivos con memoria reducida sin reentrenamiento [15]. Herramientas como Ollama, construidas sobre el motor de inferencia `llama.cpp`, abstraen esta complejidad técnica y facilitan el despliegue de modelos pre-cuantizados en local. Estudios experimentales recientes han evaluado la eficiencia de estos modelos en *hardware edge* de bajo coste, demostrando que es posible alcanzar un punto de operación equilibrado entre velocidad de inferencia, precisión de respuesta y consumo energético, incluso en dispositivos con recursos muy limitados [16]. Este ecosistema de herramientas y modelos cuantizados constituye la base tecnológica sobre la que se asienta ELIA.

Establecida la viabilidad de la inferencia local, el siguiente reto consiste en dotar al sistema de memoria contextual sobre el histórico de incidencias. Los *pipelines* RAG responden a esta necesidad: en lugar de depender exclusivamente del conocimiento paramétrico del modelo, el sistema recupera en tiempo de inferencia los registros más relevantes de una base de datos vectorial y los incorpora al contexto del LLM, reduciendo alucinaciones y mejorando la precisión en dominios especializados. Sin embargo, los pipelines RAG lineales, donde los datos fluyen en una única dirección desde el recuperador hasta el generador, resultan frágiles ante consultas complejas o contextos ruidosos. Los sistemas RAG agénticos superan esta limitación al modelar el flujo de trabajo como un grafo de estados con bifurcaciones condicionales: el agente puede reformular la consulta, reintentar la recuperación o autocorregir la respuesta en función de su evaluación de la relevancia y coherencia del contexto recuperado [8]. LangGraph implementa este paradigma mediante grafos de estado dirigidos donde los nodos representan unidades de cómputo (recuperación, razonamiento, generación) y las aristas codifican la lógica de transición. Esta combinación ha demostrado mejorar significativamente la precisión en tareas críticas de ingeniería, como el Análisis de Modos de Fallo y Efectos (FMEA, *Failure Mode and Effects Analysis*), frente a modelos sin apoyo contextual [17].

Los *benchmarks* de propósito general como HELM [18], GAIA [19] o AgentBench [20] establecen marcos de referencia para evaluar capacidades de razonamiento y ejecución en tareas heterogéneas, pero no capturan la calidad del diagnóstico técnico en dominios especializados donde el anclaje a la telemetría real y la accionabilidad de las prescripciones son más relevantes que la precisión en tareas de propósito general. La evaluación humana mediante expertos del dominio permite valorar estas dimensiones cualitativas de forma fiable, pero resulta costosa y difícilmente escalable a comparativas entre múltiples modelos. El paradigma *LLM-as-a-Judge* emerge como solución complementaria: un modelo de mayor capacidad actúa como evaluador automático aplicando una rúbrica estructurada, logrando una correlación superior al 80 % con el juicio humano experto a una fracción del coste. Sin embargo, este enfoque presenta limitaciones documentadas: tendencia a favorecer respuestas más extensas, sesgo de posición y autopreferencia hacia el estilo del modelo juez [21]. Estas limitaciones complementarias entre la evaluación humana y la automática motivan el diseño de un marco tri-nivel que combina métricas de rendimiento, evaluación experta ponderada y *LLM-as-a-Judge* como mecanismos mutuamente validantes, tal y como se describe en la Sección III.

La revisión de la literatura permite identificar con claridad el hueco que motiva este trabajo. Los sistemas de diagnóstico basados en LLM han demostrado su utilidad en entornos de operaciones IT (*Information Technology*) y mantenimiento industrial [5, 7], y trabajos recientes han comenzado a explorar el papel de la IA (Inteligencia Artificial) generativa en la protección de infraestructuras críticas [22]. Sin embargo, estos enfoques operan de forma prácticamente universal sobre infraestructura *cloud* y se centran en amenazas de ciberseguridad, sin abordar el diagnóstico de fallos físicos en sistemas de seguridad con requisitos normativos explícitos ni la operación en entornos desconectados. Por otro lado, la investigación en inferencia LLM sobre *hardware edge* ha avanzado significativamente en cuantización y eficiencia, pero no ha abordado su aplicación a *pipelines* agénticos de diagnóstico sobre telemetría industrial estructurada. Finalmente, los trabajos sobre FDD en edificios se centran en la detección de anomalías mediante aprendizaje automático clásico, sin incorporar razonamiento causal ni prescripción accionable [10]. ELIA aborda simultáneamente estas tres brechas: propone un *pipeline* de razonamiento LLM ejecutado íntegramente en local sobre *hardware edge*, específicamente diseñado para el dominio del alumbrado de emergencia conforme a la normativa citada, con una arquitectura que separa explícitamente la lógica determinista del razonamiento probabilístico y un marco de evaluación tri-nivel que valida tanto la eficiencia técnica como la calidad del diagnóstico.

III. Metodología

A. Visión general del sistema

ELIA es un sistema multiagente de detección y análisis de incidencias en luminarias de emergencia que combina procesamiento determinista basado en reglas de dominio con razonamiento semántico mediante modelos de lenguaje ejecutados localmente. El sistema se articula en dos agentes independientes con responsabilidades complementarias: el Agente de Campo, orientado a la detección en tiempo real, y el Agente de Análisis, orientado al diagnóstico estratégico periódico. Ambos agentes se implementan como grafos de ejecución con LangGraph y comparten una base de datos SQLite[3] como capa de persistencia.

La entrada al sistema proviene de un gateway embebido basado en Raspberry Pi 4 Model B[4], que

[3] `https://sqlite.org/`
[4] `https://www.raspberrypi.com/products/`

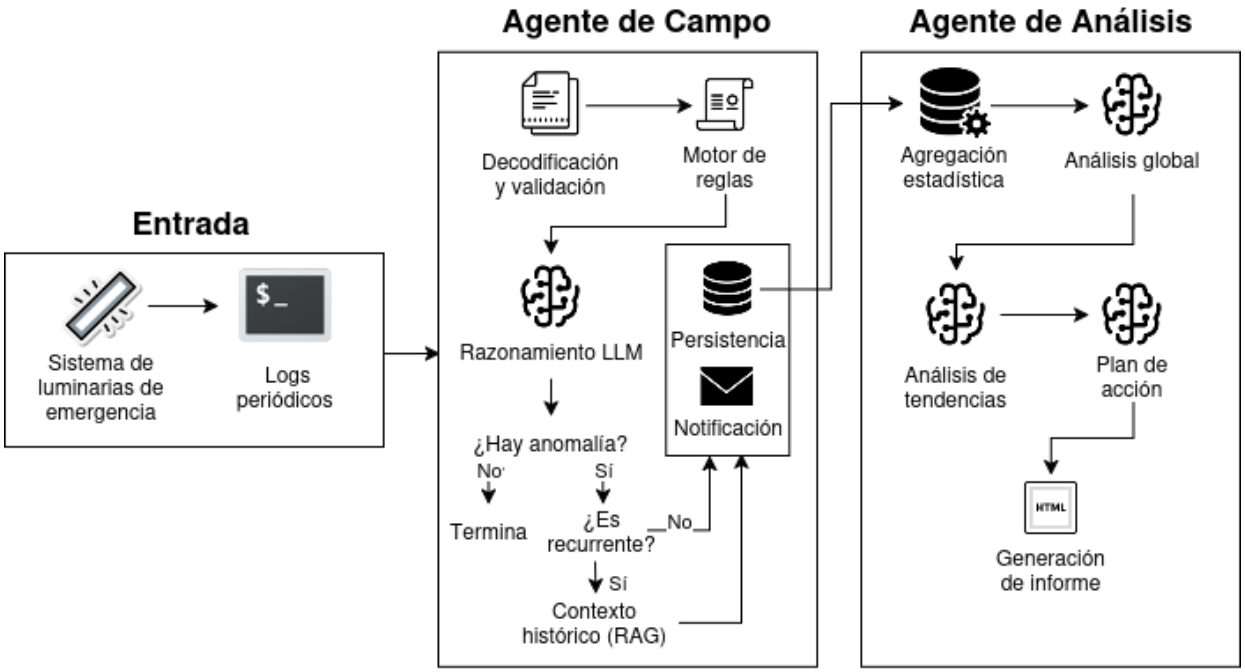


Fig. 1: Arquitectura del sistema: flujo de telemetría desde el *gateway* embebido hasta los agentes, con los grafos de ejecución del Agente de Campo (izquierda) y del Agente de Análisis (derecha), incluyendo la capa de persistencia compartida.

actúa como concentrador de telemetría: recopila periódicamente los registros de estado de cada luminaria mediante BLE (*Bluetooth Low Energy*) y los serializa en formato NDJSON. Cada registro contiene los campos de estado del dispositivo monitorizado codificados en binario, incluyendo estado operativo, nivel y tensión de batería, calidad de enlace, temperatura y códigos de error, junto con metadatos de red como latencia y *timestamp*.

Dado que las restricciones de cómputo del *gateway* impiden ejecutar modelos de lenguaje de capacidad suficiente directamente en el dispositivo, el razonamiento se delega a un servidor local con GPU (*Graphics Processing Unit*), *hardware* cada vez más habitual en los sistemas de gestión de edificios (BMS, *Building Management Systems*) modernos. Esta separación entre la adquisición de datos en el nodo embebido y el procesamiento inteligente en el servidor local es una decisión de diseño central del sistema, que permite aprovechar la capacidad de razonamiento de modelos de hasta 70B (*Billion*, miles de millones) parámetros manteniendo la privacidad de los datos de telemetría industrial y eliminando el coste variable por inferencia. La arquitectura completa se muestra en la Figura 1, donde se ilustra el flujo de telemetría desde la fuente de datos hasta los agentes, así como los grafos de ejecución de cada uno de ellos.

B. *Capa determinista: procesamiento de logs y motor de reglas*

Antes de invocar al modelo de lenguaje, los registros NDJSON son sometidos a una etapa de procesamiento determinista que transforma los datos de telemetría en crudo en un contexto estructurado y verificable. Cada registro NDJSON contiene 27 campos que codifican el estado del dispositivo monitorizado: 10 requieren decodificación explícita al estar codificados mediante *bitmask* o enumeración: `error` codifica hasta ocho fallos *hardware* simultáneos, `outputState` el estado de las salidas de potencia y emergencia, y `configuration` las funcionalidades activas del dispositivo, entre otros. Los 17 restantes son numéricos de lectura directa, incluyendo tensión de batería, temperatura, calidad de enlace y latencia de red. La base de conocimiento del sistema actúa como diccionario de referencia para esta decodificación, documentando para cada campo su nombre técnico, descripción, tipo de dato y valores aceptados. Esta etapa valida rangos, detecta valores fuera de especificación y agrupa los registros por dispositivo produciendo una serie temporal, que constituye la entrada al motor de reglas.

El motor de reglas evalúa cada serie temporal en tres niveles de abstracción creciente, todos ellos definidos declarativamente en un fichero de reglas de dominio en formato JSON (*JavaScript Object Notation*):

- **Reglas individuales** (16): detectan fallos atómicos sobre un único campo de telemetría: fallos de alimentación, autonomía o pantalla TFT (*Thin Film Transistor*) mediante el campo `error` (registro de fallos *hardware* codificado mediante *bitmask*), o condiciones térmicas críticas mediante `battery_temp` (temperatura de la batería en décimas de grado), entre otros.
- **Grupos causales** (5): agregan múltiples reglas individuales sobre un mismo dispositivo mediante lógica OR/AND para detectar patrones de fallo compuestos. Por ejemplo, `GROUP_BATTERY` se activa ante cualquier combinación de estado IDLE (dispositivo inactivo o en reposo sin reali-

zar su función de emergencia), error de batería o tensión por debajo del umbral mínimo en el mismo dispositivo.
- **Patrones colectivos** (3): operan transversalmente sobre múltiples dispositivos de la instalación simultáneamente, detectando eventos zonales indetectables por análisis individual. A diferencia de los grupos causales, no analizan síntomas de un único dispositivo sino correlaciones entre dispositivos, como cortes de suministro que afectan a una zona completa o concurrencias de pruebas funcionales en varios nodos.

Todas las reglas incluyen *technical_clues*: pistas de diagnóstico del experto de dominio que orientan el razonamiento del LLM sin condicionarlo.

La capa determinista produce dos salidas con propósitos distintos: el contexto estructurado que se entrega al LLM para su razonamiento, formado por los valores de telemetría decodificados, las reglas activadas con sus condiciones técnicas y las *technical_clues* asociadas, y los campos `responsible` (responsable de atender la incidencia) y `severity` (nivel de criticidad) de cada incidencia, que son determinados exclusivamente por las reglas y nunca inferidos por el modelo. Esta separación es una decisión de arquitectura deliberada: el LLM se ocupa del razonamiento semántico: diagnóstico, prescripción de acción y detección de recurrencias, mientras las decisiones de enrutamiento quedan gobernadas por reglas auditables. En infraestructura crítica, una notificación mal enrutada o una severidad subestimada tienen impacto operativo directo, y mantener esta lógica fuera del agente hace al sistema predecible e independiente del modelo utilizado.

Esta separación entre capa determinista y razonamiento semántico sigue el paradigma neuro-simbólico [23, 24], que combina el poder de generalización de las redes neuronales con la transparencia e interpretabilidad de los sistemas basados en reglas. Este enfoque ha demostrado ser especialmente efectivo en dominios críticos donde la auditabilidad y el cumplimiento normativo son requisitos no negociables, como la medicina o la gestión de riesgos industriales, y constituye la base teórica sobre la que se sustenta este diseño.

C. Agente de Campo (Field Agent)

El Agente de Campo implementa la detección de incidencias en tiempo real mediante un *pipeline* de ejecución condicional. La elección de una arquitectura agéntica frente a un *pipeline* secuencial fijo responde a la naturaleza condicional del procesamiento: no toda serie temporal produce una incidencia, no toda incidencia es nueva, y no todo fallo requiere el mismo contexto de razonamiento. Un grafo de ejecución permite cortocircuitar el flujo cuando no es necesario invocar al modelo, evitando llamadas innecesarias al LLM, y bifurcar el procesamiento según el estado de la base de conocimiento, sin necesidad de gestionar esta lógica externamente. Esta flexibilidad de flujo es precisamente la ventaja diferencial de las arquitecturas agénticas sobre los *pipelines* lineales en tareas de diagnóstico con lógica condicional compleja [8].

El *pipeline* de este agente se compone de cuatro etapas. Las dos primeras implementan la capa determinista descrita en la Sección III-B: la *decodificación y validación* de la telemetría y la *evaluación del motor de reglas*. Si ninguna regla se activa, el *pipeline* termina sin invocar al modelo, lo que en instalaciones con alta frecuencia de muestreo supone una reducción significativa del coste computacional.

La etapa de *razonamiento LLM* constituye el núcleo del procesamiento agéntico. El LLM recibe el contexto estructurado producido por el motor de reglas: los síntomas activados con su persistencia, las causas posibles documentadas por el experto mediante *technical_clues*, y el contexto colectivo de la instalación (información sobre el estado del resto de dispositivos y su topología de red, que permite al modelo detectar si el fallo es aislado o forma parte de un patrón zonal), y razona sobre la causa raíz del fallo y la acción correctiva más adecuada. Lo que distingue esta etapa de una simple llamada a un modelo es la gestión del conocimiento acumulado mediante RAG: cuando una anomalía ya fue resuelta anteriormente, se consulta ChromaDB mediante búsqueda por similitud semántica para recuperar la intervención previa más relevante: diagnóstico emitido, acción aplicada y resultado, que se incorporan al *prompt* junto con los síntomas actuales. El modelo recibe así no solo la telemetría presente sino también la evidencia de que una intervención anterior fue insuficiente, lo que le permite escalar la respuesta en lugar de repetirla. Esta capacidad de razonamiento contextualizado sobre el historial operativo es precisamente lo que justifica el uso de un LLM frente a un sistema de reglas puro: la generalización semántica sobre situaciones no previstas explícitamente en las reglas.

La etapa de *persistencia y notificación* materializa la incidencia: los campos `responsible` y `severity` son sobreescritos desde el motor de reglas, descartando cualquier valor que el LLM pudiera haber inferido en su razonamiento, ya que estas decisiones de enrutamiento deben estar siempre gobernadas por lógica auditable y no por el modelo. Adicionalmente, esta etapa persiste el registro en SQLite, lo indexa en ChromaDB para consultas futuras y envía una notificación al responsable correspondiente, agregando en un único mensaje todas las incidencias del mismo responsable generadas en la ejecución.

D. Agente de Análisis (Analytics Agent)

Mientras el agente anterior opera de forma reactiva sobre cada luminaria afectada individualmente, el Agente de Análisis adopta una perspectiva global y periódica: lee el historial de incidencias acumuladas en SQLite durante un periodo de tiempo y produce un diagnóstico estratégico de la instalación completa. La motivación es que una incidencia puntual de batería en una luminaria es un fallo aislado que requiere una intervención concreta. Pero si durante un período prolongado aparecen incidencias del mis-

mo tipo en múltiples luminarias de una zona, o si un tipo de fallo recurre sistemáticamente tras cada intervención, estamos ante un problema estructural del sistema que requiere una respuesta distinta. Esta distinción, entre fallo puntual y patrón sistémico, no es detectable revisando incidencias de forma individual: requiere razonar sobre el historial completo del período, algo que el Agente de Campo, por diseño, no hace.

Por otro lado, el agente realiza un número fijo de llamadas al LLM independientemente del número de dispositivos de la instalación, lo que garantiza su escalabilidad. Esto es posible porque la etapa de *agregación estadística* consolida el historial completo en un único resumen estructurado antes de invocar al modelo, de modo que las tres llamadas al LLM son una propiedad del *pipeline* y no del tamaño de la instalación.

El *pipeline* de este agente se compone de cinco etapas. La etapa de *agregación estadística*, al igual que las dos primeras etapas del Agente de Campo, es completamente determinista: agrega el historial de incidencias por dispositivo y calcula las estadísticas necesarias (total, críticas, pendientes, recurrencias y tiempo medio de resolución) para producir el resumen estructurado que constituye la entrada a las etapas de razonamiento. Las tres etapas siguientes implementan un razonamiento encadenado en el que cada llamada al LLM recibe el *output* de la anterior como contexto adicional. La etapa de *análisis global* analiza el conjunto de dispositivos, detecta correlaciones, identifica el patrón dominante de fallos y emite señales de alarma que no son visibles por análisis individual. La etapa de *análisis de tendencias* razona sobre tendencias temporales y causas sistémicas a partir del análisis anterior. La etapa de *plan de acción* recibe el análisis global y las tendencias para generar el plan priorizado con plazos y responsables concretos. Esta cadena es necesaria porque cada nivel requiere el anterior: no es posible prescribir acciones sistémicas sin haber identificado tendencias, ni tendencias sin haber analizado el patrón global. Finalmente, la etapa de *generación de informe* produce el informe HTML (*HyperText Markup Language*) interactivo y persiste el razonamiento completo en JSON para trazabilidad y evaluación posterior.

Este diseño permite obtener algo cualitativamente distinto al diagnóstico por incidencia: el estado de salud del sistema como un todo. Por ejemplo, si en un período determinado el 60 % de las incidencias corresponden a fallos de batería en luminarias de la misma zona, el agente puede inferir una causa común: un proveedor defectuoso, unas condiciones ambientales adversas, que ningún técnico detectaría revisando las incidencias una a una. Es esta capacidad de síntesis sobre el historial operativo completo la que justifica la existencia de un segundo agente independiente.

E. Despliegue local y privacidad

El sistema utiliza Ollama como servidor de inferencia local, que gestiona la cuantización y servicio de modelos mediante una API REST compatible con el estándar OpenAI. Los *pipelines* de ambos agentes se implementan con LangGraph, que proporciona el mecanismo de ejecución condicional y la gestión del estado entre etapas. La cuantización *Q4_K_M*, aplicada por defecto por Ollama, reduce el tamaño de los modelos aproximadamente un 75 % respecto a su precisión completa con una degradación mínima de calidad, haciendo viable el despliegue de modelos de hasta 70B parámetros en servidores con 48 GB de VRAM [16]. La separación entre la etapa de adquisición: el *gateway* embebido que serializa la telemetría BLE, y la de razonamiento: el servidor local con GPU, responde a las restricciones de cómputo del dispositivo embebido y sigue el patrón habitual en sistemas ciberfísicos industriales [13], permitiendo escalar la capacidad de razonamiento de forma independiente al *hardware* de campo.

F. Marco de evaluación

La evaluación del sistema se estructura en torno a dos preguntas de investigación:

RQ1 ¿Qué modelos son viables para este tipo de razonamiento industrial en entorno local?
RQ2 ¿Cómo se relacionan tamaño, arquitectura y coste con la calidad del diagnóstico?

Para responderlas se diseña un *benchmark* comparativo sobre 15 modelos de lenguaje ejecutados localmente mediante Ollama.

F.1 Selección de modelos

La selección cubre tres dimensiones de variabilidad. En cuanto a *tamaño*, el rango va de 1.5B a 70B parámetros, cubriendo desde dispositivos de cómputo reducido sin GPU dedicada hasta servidores con aceleración *hardware*, lo que permite analizar la viabilidad del sistema en distintos entornos de despliegue *edge*. En cuanto a arquitectura, se incluyen modelos densos estándar, modelos de razonamiento explícito con *chain-of-thought* interno [25] (DeepSeek-R1, QwQ) y un modelo de mezcla de expertos (Mixtral 8x7b, con 47B parámetros totales pero solo 12B activos por inferencia). En cuanto a *procedencia*, se representan seis organizaciones: Meta, Mistral AI, Alibaba, Google, Microsoft y DeepSeek, con estrategias de entrenamiento diferenciadas que permiten aislar el efecto de la arquitectura del efecto del tamaño.

La Tabla I resume los modelos seleccionados. El número de parámetros se expresa en miles de millones (*Billion*, B, $\sim 10^9$) y el tamaño en disco corresponde a lo que ocupa cada modelo cuantizado en *Q4_K_M* descargado mediante Ollama.

Los criterios de selección de modelos priorizan la reproducibilidad (modelos públicamente disponibles en Ollama con documentación verificable) y una comparabilidad controlada que permita validar la

Tabla I: Modelos seleccionados para el *benchmark* comparativo.

Modelo	Organización	Arquitectura	Parámetros	Tamaño en disco
`qwen2.5:1.5b`	Alibaba	qwen2	1.5B	986 MB
`smollm2:1.7b`	HuggingFace	llama	1.7B	1.8 GB
`phi3.5:3.8b`	Microsoft	phi3	3.8B	2.2 GB
`gemma3:4b`	Google	gemma3	4B	3.3 GB
`mistral:7b`	Mistral AI	llama	7B	4.4 GB
`llama3.1:8b`	Meta	llama	8B	4.9 GB
`gemma3:12b`	Google	gemma3	12B	8.1 GB
`qwen2.5:14b`	Alibaba	qwen2	14B	9.0 GB
`phi4:14b`	Microsoft	phi3	14B	9.1 GB
`mistral-small3.2:24b`	Mistral AI	mistral3	24B	15 GB
`qwq:32b`	Alibaba	qwen2	32B	19 GB
`deepseek-r1:32b`	DeepSeek	qwen2	32B	19 GB
`mixtral:8x7b`	Mistral AI	llama	46.7B	26 GB
`llama3.3:70b`	Meta	llama	70B	42 GB
`deepseek-r1:70b`	DeepSeek	llama	70B	42 GB

hipótesis de escalabilidad. Para ello, el diseño experimental incluye: (1) pares de modelos con distinta arquitectura y mismo tamaño (`qwq:32b` frente a `deepseek-r1:32b`) para aislar el efecto del diseño del modelo y su capacidad de razonamiento; y (2) pares de la misma familia con distinto tamaño (`gemma3:4b` frente a `gemma3:12b`, o `deepseek-r1:32b` frente a `deepseek-r1:70b`) para cuantificar el impacto del número de parámetros en la precisión del diagnóstico y la latencia resultante. Este enfoque permite identificar el punto de operación óptimo entre recursos computacionales y calidad de análisis. Todos los modelos se evalúan en modo inferencia directa, sin *fine-tuning* ni adaptación al dominio, empleando únicamente *prompting* estructurado con el contexto generado por el sistema en cada caso de uso.

F.2 Protocolo de evaluación

Cada modelo se evalúa sobre los siete casos de uso del sistema, diseñados para cubrir los tres niveles del motor de reglas (fallos individuales, grupos causales y patrones colectivos) y la gestión de recurrencias, casuística que no depende del motor de reglas sino del razonamiento contextual del Agente de Campo mediante RAG. Para garantizar la comparabilidad, antes de cada modelo se reinicializa la base de datos y se ejecutan los casos en el mismo orden, resolviendo las incidencias necesarias antes del caso de recurrencias para que el sistema las detecte como tales. Toda la salida del LLM: observación, razonamiento y acción prescrita, se persiste en JSON para su análisis posterior.

La evaluación se estructura en tres dimensiones complementarias. La primera es el *rendimiento computacional*: latencia media por decisión, desviación típica, tiempo total de inferencia y coste por decisión, que permite evaluar la viabilidad de cada modelo en un entorno de producción con restricciones de tiempo real.

La segunda es la *fidelidad del razonamiento*, evaluada mediante una rúbrica ponderada aplicada sobre una muestra representativa de incidencias. Se cubren cuatro criterios: Fidelidad técnica (35 %), mide si los valores técnicos citados por el modelo coinciden con la telemetría real; Accionabilidad (35 %), que mide si la acción prescrita es concreta y ejecutable; Tono (15 %), que evalúa el uso de lenguaje técnico apropiado; y Concisión (15 %), que penaliza la verbosidad innecesaria. La ponderación prioriza los dos primeros por ser los criterios con impacto operativo directo en infraestructura crítica.

La tercera es la evaluación *LLM-as-a-Judge*: un modelo de lenguaje independiente actúa como auditor automático, aplicando la misma rúbrica sobre todas las incidencias generadas. Se selecciona `qwen2.5:72b` como modelo juez por su tamaño significativamente superior al de los modelos evaluados, su pertenencia a una familia arquitectónica distinta (evitando el sesgo de autocomplacencia [26]) y su disponibilidad en Ollama, lo que mantiene la privacidad de los datos de evaluación. La comparación entre la evaluación del experto humano y la del juez automático permite además analizar la consistencia entre ambos métodos y detectar divergencias sistemáticas.

G. Entorno experimental y caracterización de datos

Las pruebas de inferencia se han ejecutado sobre dos GPUs NVIDIA GeForce RTX 4090 (48 GB VRAM agregados) bajo Debian GNU/Linux 12, con Ollama v0.5 como servidor de inferencia y CUDA 12.5. Esta configuración permite ejecutar los 15 modelos evaluados, incluidos los de 70B parámetros cuantizados en *Q4_K_M*, dentro de los límites de memoria disponible.

El *dataset* de evaluación se ha construido a partir de un entorno simulado que replica una instalación con 3 plantas y red *mesh* Wirepas[5], con una frecuencia de muestreo de 5 minutos durante un periodo de observación de una hora. Se han definido siete casos de uso agrupados en tres escenarios:

- **Escenario Estándar (Casos 1-4):** Red de 8

[5] `https://wirepas.com/`

Tabla II: Resultados del *benchmark* comparativo: rendimiento computacional y calidad del razonamiento.

Modelo	T[a]	Inc[b]	Rec[c]	Lat[d]	Experto					Juez (qwen2.5:72b)				
					FT	Ac	T	C	Media	FT	Ac	T	C	Media
qwen2.5:1.5b	39.35	21	2	0.72	2.43	2.71	4.29	3.43	2.96	2.5	3.64	5.71	5.0	3.76
smollm2:1.7b	219.56	10	0	3.12	0.0	0.0	0.0	0.0	0.0	1.21	1.00	1.29	1.14	1.14
phi3.5:3.8b	158.58	5	0	6.04	1.14	0.86	1.29	1.14	1.06	1.36	1.21	1.29	1.21	1.27
gemma3:4b	104.13	28	2	2.66	3.79	4.21	4.86	3.57	4.07	3.71	4.71	5.14	4.71	4.43
mistral:7b	85.03	33	6	2.30	6.00	5.86	7.64	6.79	6.32	6.5	6.93	8.79	7.79	7.18
llama3.1:8b	158.61	18	2	4.30	0.93	1.00	1.79	1.57	1.18	2.43	1.86	2.57	2.21	2.22
gemma3:12b	279.17	30	0	6.42	5.71	6.29	6.57	4.64	5.88	5.43	6.21	6.57	5.71	5.92
qwen2.5:14b	250.02	38	6	4.45	7.68	7.21	9.07	8.43	7.90	7.86	7.93	9.07	8.36	8.14
phi4:14b	164.1	30	5	5.40	7.57	7.64	9.14	7.79	7.86	7.07	7.71	8.93	8.07	7.72
mistral-small3.2:24b	118.5	14	0	6.97	3.50	3.86	4.00	3.43	3.69	3.79	3.71	4.00	3.64	3.77
qwq:32b	60.7	0	0	-	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
deepseek-r1:32b	816.57	38	6	23.42	6.50	6.79	8.86	7.64	7.13	6.79	7.29	8.93	7.93	7.45
mixtral:8x7b	149.37	34	4	4.47	6.50	6.07	7.36	6.29	6.45	4.64	6.64	7.57	6.79	6.10
llama3.3:70b	627.31	37	6	17.61	7.21	6.93	8.86	7.93	7.47	7.71	7.79	9.0	7.93	7.96
deepseek-r1:70b	1559.01	38	6	43.50	8.14	8.29	9.21	8.28	8.37	8.93	8.5	9.21	8.36	8.73

[a] Tiempo total de ejecución (s). [b] Incidencias detectadas sobre 38 posibles. [c] Recurrencias detectadas (máx. 6). [d] Latencia media por decisión (s).
FT: Fidelidad técnica (35 %). Ac: Accionabilidad (35 %). T: Tono (15 %). C: Concisión (15 %). Media: Media ponderada.

luminarias de emergencia con 96 registros, para validar diagnósticos individuales y colectivos.

- **Escenario Extendido (Caso 5):** La misma configuración ampliada a 10 nodos incluyendo controladores de entradas digitales (SmartZ GPI), resultando en 132 registros, para validar la integración de sensores externos.
- **Análisis de Recurrencia (Casos 6 y 7):** Escenario crítico que evalúa la memoria operativa del agente. Se introducen datos que contradicen una resolución previa para verificar si el sistema detecta la recurrencia, recupera el historial de la intervención fallida y escala la prescripción hacia una solución de mayor nivel.

La descripción detallada de cada caso de uso, incluyendo la incidencia evaluada y su justificación metodológica, se recoge en la Sección IV.

El uso de *hardware* de alto rendimiento permite establecer un *benchmark* de referencia completo sobre los 15 modelos evaluados, facilitando la identificación del umbral mínimo de capacidad de razonamiento requerido y la selección del modelo adecuado en función de las restricciones de *hardware* del despliegue final.

IV. Resultados experimentales

Esta sección presenta los resultados del *benchmark* comparativo sobre los 15 modelos evaluados, estructurados en torno a las dos preguntas de investigación definidas en la Sección III. Se analiza primero la viabilidad computacional de cada modelo (RQ1) y posteriormente la calidad del razonamiento mediante evaluación experta y *LLM-as-a-Judge* (RQ2), identificando el perfil óptimo para despliegue en entornos *edge* con restricciones de recursos. La Tabla II resume de forma conjunta los resultados de ambas dimensiones para los 15 modelos evaluados.

A. Rendimiento computacional (RQ1)

El análisis de rendimiento debe interpretarse conjuntamente con la cobertura de incidencias: una latencia baja o un tiempo total reducido no implica necesariamente mayor eficiencia si el modelo no genera todas las incidencias posibles. Modelos como `qwen2.5:1.5b` (0.72 s, 21/38 incidencias) o `smollm2:1.7b` (3.12 s, 10/38 incidencias) presentan latencias competitivas precisamente porque no procesan el espacio completo de fallos. En el caso extremo, `qwq:32b` registra el tiempo total más bajo del grupo de modelos con arquitectura de razonamiento (60.7 s) sin generar ninguna incidencia, lo que lo hace inviable con independencia de su aparente eficiencia temporal. Este resultado sugiere que su arquitectura *chain-of-thought* interno es incompatible con los requisitos de salida estructurada del *pipeline* agéntico.

Los modelos que completan la cobertura total (38/38 incidencias y 6/6 recurrencias) son `qwen2.5:14b`, `deepseek-r1:32b` y `deepseek-r1:70b`. De estos, `qwen2.5:14b` es el único que lo hace con una latencia por debajo de 5 s (4.45 s), constituyendo el punto de operación óptimo en términos de cobertura y eficiencia temporal. Los modelos de razonamiento explícito `deepseek-r1:32b` y `deepseek-r1:70b` alcanzan la misma cobertura pero a un coste de 23.42 s y 43.50 s respectivamente, entre 5 y 10 veces superior, lo que los descarta para despliegues con restricciones de tiempo real. Un resultado contraintuitivo es el de `gemma3:12b` frente a `gemma3:4b`: el modelo de mayor tamaño detecta menos incidencias (30 vs 28) sin detectar ninguna recurrencia, con una latencia 2.4 veces superior, invirtiendo la relación esperada entre tamaño y rendimiento dentro de la misma familia arquitectónica.

El análisis de cobertura y latencia responde a RQ1 en términos de viabilidad operativa, pero no dice nada sobre la calidad del diagnóstico generado. La siguiente subsección aborda esta dimensión.

B. Calidad del razonamiento (RQ2)

La evaluación de fidelidad se realiza sobre una muestra de siete incidencias que se corresponden con los casos de uso definidos en la Sección III, seleccionadas para cubrir de forma sistemática el espacio de casos del sistema:

- **Regla individual atómica** (CRITICAL, Caso

de uso 4): fallo en el termistor NTC (*Negative Temperature Coefficient*) sobre un único campo de telemetría. Constituye el caso base de referencia que todos los modelos deberían resolver correctamente.
- **Condición térmica crítica** (CRITICAL, Caso de uso 2): fallo con valores numéricos concretos de temperatura. Evalúa el anclaje técnico del modelo a la telemetría real.
- **Patrón colectivo de alerta** (WARNING, Caso de uso 1): evento zonal que afecta a múltiples luminarias simultáneamente. Requiere razonamiento transversal sobre la instalación completa.
- **Grupo causal** (CRITICAL, Caso de uso 3): fallo compuesto que requiere correlacionar múltiples condiciones simultáneas. Evalúa la capacidad de razonamiento sobre causas compuestas.
- **Fallo eléctrico con severidad reducida** (WARNING, Caso de uso 5): evalúa si el modelo calibra correctamente el tono y la accionabilidad en fallos no críticos, evitando tanto la subestimación como la sobrealarma.
- **Grupo causal con recurrencia** (CRITICAL, Caso de uso 6): la misma luminaria y grupo con resoluciones previas registradas en el historial. Evalúa si el modelo recupera el contexto histórico y escala la prescripción en lugar de repetirla.
- **Patrón colectivo con recurrencia** (WARNING, Caso de uso 7): recurrencia sobre un evento zonal de emergencia previamente resuelto. Combina la complejidad del razonamiento colectivo con la gestión del historial operativo.

Cada incidencia es evaluada de forma independiente por un experto del dominio y por un modelo *LLM-as-a-Judge* (`qwen2.5:72b`), aplicando ambos la misma rúbrica ponderada definida en la Sección III-F.2. La comparación entre ambas evaluaciones permite analizar la consistencia del juez automático con el criterio experto y detectar divergencias sistemáticas.

Los resultados revelan tres grupos de calidad diferenciados, que deben interpretarse en conjunción con la cobertura de incidencias de la Tabla II. Las puntuaciones incluyen penalización por incidencias no generadas (puntuación 0), por lo que reflejan tanto la cobertura como la calidad del razonamiento en un único valor.

Entre los modelos con cobertura total (38/38 incidencias y 6/6 recurrencias), `deepseek-r1:70b` obtiene la puntuación más alta según el experto (8.37), seguido de `qwen2.5:14b` (7.90). Cruzando calidad y latencia, `qwen2.5:14b` emerge como el modelo de mayor eficiencia global: cobertura máxima con solo 4.45 s de latencia, frente a los 43.50 s de `deepseek-r1:70b` para una mejora de apenas 0.47 puntos. `llama3.3:70b`, con cobertura casi completa (37/38 incidencias, 6/6 recurrencias) y una puntuación de 7.47, representa una alternativa válida dentro del grupo de calidad alta, aunque su latencia de 17.61 s lo sitúa en un perfil intermedio entre eficiencia y capacidad de razonamiento.

Un patrón sistemático en todos los modelos evaluables es la brecha entre tono y fidelidad técnica: los modelos producen respuestas con lenguaje técnico apropiado antes de lograr un anclaje preciso a la telemetría real. Esta brecha es especialmente pronunciada en `deepseek-r1:32b` (+2.36 puntos) y en modelos de menor capacidad como `qwen2.5:1.5b` (+1.86 puntos), lo que sugiere que la fluidez técnica del lenguaje no implica necesariamente precisión en los valores referenciados.

Respecto a la consistencia entre experto y juez automático, la diferencia media es inferior a 0.50 puntos en 10 de los 15 modelos, lo que valida el uso de `qwen2.5:72b` como evaluador automático en este dominio. Las divergencias más significativas se producen en `mixtral:8x7b`, donde el juez penaliza la fidelidad técnica 1.86 puntos por debajo del experto, y en `smollm2:1.7b` y `llama3.1:8b`, donde el juez es más permisivo que el experto ante respuestas de baja calidad. Estos resultados sugieren que el juez automático es fiable para comparaciones globales pero infravalora sistemáticamente los fallos de razonamiento en modelos de baja capacidad.

V. Conclusiones

Este trabajo presenta ELIA, un sistema multiagente de diagnóstico de luminarias de emergencia que opera íntegramente en local mediante modelos LLM cuantizados, demostrando la viabilidad del paradigma neuro-simbólico en entornos *edge* sin dependencia de infraestructura *cloud*. El *benchmark* comparativo sobre 15 modelos permite extraer seis conclusiones con respaldo empírico directo:

- **Viabilidad del paradigma *edge* local.** La cuantización *Q4_K_M* permite ejecutar modelos de hasta 70B parámetros en 48 GB de VRAM manteniendo la privacidad de los datos de telemetría y eliminando el coste variable por inferencia, lo que elimina las tres barreras estructurales del paradigma *cloud* identificadas en la introducción.
- **Punto de operación óptimo identificado.** `qwen2.5:14b` alcanza cobertura completa (38/38 incidencias, 6/6 recurrencias) con una puntuación de fidelidad de 7.90 y latencia de 4.45 s, siendo el único modelo que combina cobertura máxima y latencia por debajo de 5 s. Para entornos donde la latencia no sea restrictiva, `deepseek-r1:70b` ofrece la mayor puntuación de fidelidad del *benchmark* (8.37) a costa de una latencia de 43.50 s, constituyendo una alternativa válida cuando la calidad del diagnóstico prime sobre la eficiencia temporal.
- **El tamaño no predice el rendimiento en dominio específico.** `gemma3:12b` rinde peor que `gemma3:4b` en cobertura y latencia, y `phi3.5:3.8b` detecta únicamente 5 de 38 incidencias pese a superar en *benchmarks* generalistas a modelos de mayor tamaño. Los *rankings* generalistas no transfieren al razonamiento industrial técnico.

- **La implementación del razonamiento explícito condiciona fuertemente la viabilidad operativa.** `qwq:32b` resulta incompatible con la generación de salida estructurada, mientras que los modelos `deepseek-r1` operan a latencias entre 5 y 10 veces superiores a modelos densos con cobertura equivalente.
- **La separación neuro-simbólica es esencial para la fiabilidad.** Los campos `responsible` y `severity` son siempre correctos independientemente del modelo utilizado, demostrando que la arquitectura neuro-simbólica protege las decisiones de enrutamiento crítico de los fallos del componente probabilístico.
- **El juez automático es viable para evaluación a escala.** Las puntuaciones medias del juez `qwen2.5:72b` y del experto humano muestran una diferencia inferior a 0.5 puntos en 10 de los 15 modelos evaluados, con una tendencia del juez a ser ligeramente más permisivo en modelos de baja capacidad. Esto valida su uso como evaluador automático escalable para *benchmarks* con mayor número de modelos o incidencias, donde la evaluación manual sería inviable.

Como trabajo futuro se plantea la extensión del sistema a detección proactiva mediante series temporales, la evaluación sobre datos reales de instalaciones en producción, y la exploración de *fine-tuning* sobre modelos pequeños con datos del dominio para mejorar la cobertura en el rango *edge*, donde modelos como `qwen2.5:1.5b` muestran latencias muy competitivas pero cobertura insuficiente.

Agradecimientos

Esta investigación está financiada en parte por el Ministerio de Economía y Competitividad (MINECO) del Gobierno de España, en el marco del proyecto OASIS (PID2023-148285OB-C41), junto con las ayudas PERTE-Chip para cátedras (Cátedra UCLM, TSI-069100-2023-0014) del Ministerio de Transformación Digital y Función Pública, y a través del FEDER y del Gobierno Regional de Castilla-La Mancha en el marco de los proyectos REACH (SBPLY/24/180225/000105) y RuralCARE (SBPLY/24/ 180225/000158) y el programa Horizonte Europa en el marco del proyecto MAYA (GA Nº 101213323).

Referencias

[1] Ministerio de Transportes, Movilidad y Agenda Urbana, "Código técnico de la edificación (CTE)," Tech. Rep., Gobierno de España, 2006, Documento Básico SUA y SI. Disponible en: `https://smediterraneo.com/normativa-cte-del-alumbrado-de-emergencia/`.

[2] Ministerio de Ciencia y Tecnología, "Reglamento electrotécnico para baja tensión (REBT)," Tech. Rep., Gobierno de España, 2002, Real Decreto 842/2002, ITC-BT-28.

[3] AENOR, "UNE-EN 50172:2005. Sistemas de alumbrado de emergencia alimentado por baterías autónomas," 2005.

[4] AENOR, "UNE-EN 62034:2012. Sistemas automáticos de prueba para alumbrado de emergencia alimentado por batería," 2012.

[5] Jongsu Park, Seongwoo Cho, Yoonji Chae, and et al., "LLM-based multi-agent system for autonomous maintenance process of machine tools," *PHM Society Asia-Pacific Conference*, vol. 5, 2025.

[6] Kushal Khatiwada, Jayden Hopper, Joseph Cheatham, and et al., "Large language models in the IoT ecosystem - A survey on security challenges and applications," *arXiv preprint arXiv:2505.17586*, 2025.

[7] Lingzhe Zhang, Tong Jia, Mengxi Jia, and et al., "A survey of AIOps in the era of large language models," *arXiv preprint arXiv:2507.12472*, 2025.

[8] Aditi Singh, Abul Ehtesham, Saket Kumar, and et al., "Agentic retrieval-augmented generation: A survey on agentic RAG," *arXiv preprint arXiv:2501.09136*, 2025.

[9] Denis Leite, Emmanuel Andrade, Diego Rativa, and et al., "Fault detection and diagnosis in industry 4.0: A review on challenges and opportunities," *Sensors*, vol. 25, no. 1, pp. 60, 2025.

[10] Giovanni Cicceri and Fabrizio De Vita, "Fault diagnosis in IoT applications: Advances, challenges, and future directions," *Journal of Sensor and Actuator Networks*, vol. 14, no. 6, pp. 104, 2025.

[11] Yinfang Chen, Huaibing Xie, Minghua Ma, and et al., "Automatic root cause analysis via large language models for cloud incidents," in *Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys)*, 2024.

[12] Hamzah A.A.M. Qaid, Bo Zhang, Dan Li, and et al., "FD-LLM: Large language model for fault diagnosis of machines," *arXiv preprint arXiv:2412.01218*, 2024.

[13] Mayank Arya and Yogesh Simmhan, "Understanding the performance and power of LLM inferencing on edge accelerators," *arXiv preprint arXiv:2506.09554*, 2025.

[14] Guanzhong Pan, Yifei Zhang, Jiayi Liu, and et al., "A cost-benefit analysis of on-premise large language model deployment: Breaking even with commercial LLM services," *arXiv preprint arXiv:2509.18101*, 2025.

[15] Rui Wang, Zhiyong Gao, Liuyang Zhang, and et al., "Empowering large language models to edge intelligence: A survey of edge efficient LLMs and techniques," *Computer Science Review*, vol. 57, 2025.

[16] Erik Johannes Husom, Arda Goknil, Merve Astekin, and et al., "Sustainable LLM inference for edge AI: Evaluating quantized LLMs for energy efficiency, output accuracy, and inference latency," *ACM Transactions on Internet of Things*, vol. 6, no. 4, 2025.

[17] Lukas Bahr, Christoph Wehner, Judith Wewerka, and et al., "Knowledge graph enhanced retrieval-augmented generation for failure mode and effects analysis," *Journal of Industrial Information Integration*, vol. 45, pp. 100807, 2025.

[18] Percy Liang, Rishi Bommasani, Tony Lee, and et al., "Holistic evaluation of language models," *arXiv preprint arXiv:2211.09110*, 2022.

[19] Grégoire Mialon, Clémentine Fourrier, Craig Swift, and et al., "GAIA: A benchmark for general AI assistants," *arXiv preprint arXiv:2311.12983*, 2023.

[20] Xiao Liu, Hao Yu, Hanchen Zhang, and et al., "AgentBench: Evaluating LLMs as agents," *arXiv preprint arXiv:2308.03688*, 2023.

[21] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, and et al., "Judging LLM-as-a-judge with MT-bench and chatbot arena," *arXiv preprint arXiv:2306.05685*, 2023.

[22] Yagmur Yigit, Mohamed Amine Ferrag, Mohamed C Ghanem, and et al., "Generative AI and LLMs for critical infrastructure protection: Evaluation benchmarks, agentic AI, challenges, and opportunities," *Sensors*, vol. 25, no. 6, pp. 1666, 2025.

[23] George Prenosil, Thilo Weitzel, Sandra Bello, and et al., "Neuro-symbolic AI for auditable cognitive information extraction from medical reports," *Communications Medicine*, 2025.

[24] Chaitanya Kolli, "Hybrid Neuro-Symbolic models for ethical AI in risk-sensitive domains," *arXiv preprint arXiv:2511.17644*, 2025.

[25] Jason Wei, Xuezhi Wang, Dale Schuurmans, and et al., "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[26] Haitao Li, Qian Dong, Junjie Chen, and et al., "LLMs-as-judges: A comprehensive survey on LLM-based evaluation methods," *arXiv preprint arXiv:2412.05579*, 2024.

Indice de autores

Abad Fidalgo, Pablo, 21
Abellán Miguel, José Luis, 117, 157, 665, 689
Acacio Sánchez, Manuel Eugenio, 157, 665, 689
Agulló-Domingo, Carlos, 821
Alladi, Shreya, 655
Almeida Rodríguez, Francisco, 493, 619
Alonso García, Esther, 21
Alonso Pascual, Sergio, 789
Alonso, Mario, 299
Amador, María Ángeles, 957
Amenabar, Alex, 401, 485
Amor-Martín, Adrián, 1025
Argüello, Francisco, 31
Ariño-Sales, Juan F., 197
Atukpor, Emanuel, 949

B. Heras, Dora, 31
Badía, José M., 321, 1025
Baltasar, Clara, 205, 547
Bandera, Gerardo, 583
Baños González, Miguel, 991
Barba, Jesús, 237, 921
Baviera Guaita, Alejandro, 359
Becerra Alvarez, Edwin Christian, 983, 1001
Belloch, José A., 1025
Bendjeddou, Abdellatif, 963
Bermejo, Belen, 285
Blanco Alcaine, Hector, 421
Blanco Pérez, Vicente, 493, 619
Bolaños Peño, Cristina, 1035
Bosque Orero, José Luis, 251
Botella, Guillermo, 299
Boubrima, Ahmed, 963
Bourcier, Johann, 367
Briz Velasco, José Luis, 169, 377, 421

Caba, Julián, 237, 921
Calderón-Mateos, Alejandro, 147, 265, 275, 509
Camarmas-Alonso, Diego, 147, 265, 509
Caminero Herráez, María Blanca, 601, 971
Campos Ferrer, Cristian, 1017
Cañada Rostro, Carlos, 683, 799
Cano, José, 689
Capel Tuñón, Manuel I., 631
Carrera Farran, Francesc Xavier, 501
Carretero, Jesús, 89, 99, 107, 127, 147, 211, 219, 509, 591, 831
Carrión, Carmen, 601, 971
Carrizales-Espinoza, Diana, 107, 211
Casanueva Morato, Daniel, 859
Cascado Caballero, Daniel, 859
Cascajo García, Alberto, 99
Castelló, Adrián, 473, 639
Castillo, Maria Isabel, 321
Cebrián-Márquez, Gabriel, 179, 723
Cerezuela Escudero, Elena, 865
Cervellini, Paula, 851
Chaves, Pedro, 427
Civit-Balcels, Antón, 901
Coluzza, Ivan, 391
Coronado Salazar, Elías, 1001
Corral García, Javier, 991
Costero, Luis, 463
Cuenca-Asensi, Sergio, 949
Cuenca, Pedro, 723
Czappa, Fabian, 347

Da Silva Verbel, Vitor, 229, 501, 709
Daksha, Lohit, 821
De Castro, Manuel, 555
De la Calle Herrero, F. Javier, 291
De la Cerda González, Juan Pablo, 1001
De Lacy Orellana-Pizarro, Jaime, 305
De Toro Castro, Antonio, 437, 453
De Torres Sánchez, Alejandra, 69
Del Barrio, Alberto A., 299
Del Toro García, Xavier, 1035
Delicado Martínez, Francisco Manuel, 179
Díaz Pier, Sanda, 347
Díaz Reyes, David, 733
Díaz Romero, Santiago, 1009
Díaz-Honrubia, Antonio Jesús, 205, 547, 723
Díez-Poza, David, 789
Domínguez Morales, Juan P., 845, 859
Donaire, Laura María, 337

E. Singh, David, 99, 591, 831
Echevarría Floranes, Aitor, 251
El Jerari, Aymane, 821
Entrialgo Castaño, Joaquín, 291
Escolar, Soledad, 237, 851, 921
Escribano Romero, Daniel, 39
Escudero-Sahuquillo, Jesús, 51, 77
Esteban Sauce, Asier, 883

Fazio, Maria, 89
Fernández Camello, Alejandro, 761
Fernández Cuevas, Plácido, 845
Fernández Fabeiro, Jorge, 931
Fernández Muñoz, Javier, 99
Fernández Sánchez, Carlos, 447
Fernández-Cuevas, Plácido, 901
Fernandez-Urroz, Unai, 875
Ferrer Beired, Jorge, 753
Filloy Montesino, Nicolás, 931
Flich Cardo, José, 565, 575, 717
Florensa, Didac, 911
Fodor, Zoltan, 421
Frias, Lester, 1025

G. Almudéver, Carmen, 313
G. Lema, Darío, 291
G. Navarro, Ángeles, 1017
Galiano, Vicente, 701
García Carballeira, Félix, 61
García Flores, María B., 683, 799
García Guirado, Antonio, 665
García Herrero, Francisco, 313
García Sánchez, Carlos, 761, 779
García-Blas, Javier, 127, 591
García-Carballeira, Félix, 147, 265, 275, 509
García, Felix O., 991
García, Pedro Javier, 51, 77
Garciarena, Unai, 401, 485, 875
Garrido-Hidalgo, Celia, 957
Gherasim, Iulius, 779
Gijón Flores, Ángela, 1035
Giné de Sola, Francesc, 501
Gomariz Martínez, Pablo, 179
Gómez López, Gabriel, 77
Gómez Luna, Juan, 157
Gómez Requena, María Engracia, 359
Gómez-Hernández, Eduardo José, 13
Gómez-Plana Rodríguez, Manuel, 61
Gómez, María E., 413
González Navarro, María Ángeles, 733
González Ruiz, Vicente, 259, 437, 453, 611
Gonzalez-Compean, J. L., 211
Gonzalez-Escribano, Arturo, 789
González-Montesoro, Darío, 949
González-Navarro, Sonia, 1017
González, Sonia, 385
Gracia Rodríguez, Alex, 377, 421
Gran Tejero, Rubén, 137, 753
Grao González, Daniel, 501
Guerrero Espinosa, Álvaro, 265, 275
Guirado Fernández, Fernando, 229, 709
Gutiérrez, Eladio, 385
Guzelhan, Seyda, 821

Haut Hurtado, Juan M., 683, 773, 799, 815
Hernández González, Nicolás, 493
Hernández Lorite, Rogelio, 893
Hernández Luz, Carles, 565, 575
Hernández, Carlos, 851
Hernandez, Rogelio, 937
Herranz, Victoria, 701
Herrerías León, Andrea, 583
Hidalgo-Izquierdo, Víctor, 601, 971
Hormigo, Francisco Javier, 385, 733
Hristov, Emil, 285

Ibáñez, Mario, 347
Igual, Francisco D., 463
Ilarraza Sarto, Marcos, 753
Imbernón Tudela, Baldomero, 117
Inglés Romero, Juan Francisco, 117

Jimborean, Alexandra, 655, 743
Jiménez-Fernández, Ángel, 845, 859, 901, 1009
Juiz, Carlos, 285

Kaster, Marvin, 347
Khollala, Marouan, 337
Kono García, Dionicio Yukitaro, 983, 1001
Kulasekara, Maithri R., 117
Kurniawan, Handy, 313
Kuzman, Melisa, 851, 921

Landeros Muñoz, Nicolás, 137
Lei, Jie, 473
León, Germán, 1025
Lérida Monsó, Josep Lluís, 229
Linares Barranco, Alejandro, 859, 865, 1009
Llanos, Diego R., 555
López Granado, Otoniel, 529, 537
López Oliva, Vicente, 321
López Redondo, Juana, 69
López-Martínez, Daniel, 555
López, Juan Carlos, 237, 1035
Lu, Han, 347
Lupión Lorente, Marcos, 437, 453, 611

M. Garzón, Ester, 337
Marín Nuño, Anna, 709
Martín Osuna, Juan José, 565
Martínez Álvarez, Antonio, 949
Martínez Fernández, Jaime, 157
Martínez Ortigosa, Pilar, 437, 453, 611
Martínez Rach, Miguel, 529, 537
Martínez Rosas, Gabriela, 631
Martínez-Herrero, Pedro Jesús, 723
Martínez, Héctor, 473, 639
Mas, Lluis, 911
Mateo, Jordi, 911
Medina Jara, Matias Nicolas, 229

Medrano Navalón, Carlos, 77
Menchaca-Martínez, Antonio de Jesús, 367
Mezquita Raya, Pedro, 69
Migallón, Héctor, 529, 537
Mira, José Luis, 851
Molina Picó, Antonio, 649
Montes-Sánchez, Juan M., 845, 901
Morabito, Gabriele, 89
Moreno-Vassart, X., 701
Moreno, Juan José, 259
Mori, Libe, 875, 883
Morón Fernández, Maria José, 807
Mouriño Gallego, José Carlos, 931
Mujica, Gabriel, 937
Muñoz-Muñoz, Dario, 147

Navaridas, Javier, 189, 391, 401, 485, 875, 883
Navarro Lázaro, Javier, 611
Navarro, Adrián, 689
Navarro, Marta, 413

Olcoz, Katzalin, 463
Oliva Costas, Joaquim, 501
Olivares, Teresa, 957, 971
Ondicol, Elena, 331
Ortega Cisneros, Susana, 983
Ortega, Gloria, 337
Ortega, Víctor, 237
Orts, Francisco, 337

Pallardó-Julià, Vicent, 413
Paoletti Ávila, Mercedes E., 683, 773, 799, 815
Paramés Estévez, Santiago, 931
Pascual, Jose A., 189, 391, 401, 485, 875, 883
Pecino Fernández, Manuel Antonio, 39
Pérez Delgado, Inés, 893
Pérez Martín, Simón, 1009
Pérez Pavon, Borja, 251
Pérez Pedrajas, Samuel, 675
Pérez, Borja, 347
Perias, Arthur, 743
Petit, Salvador, 39, 413
Petre Carastoian, Cosmin Octavian, 591
Piso Fernández, Daniel, 7
Plata, Óscar, 385, 583
Pons, Lucia, 305, 413
Portilla Berrueco, Jorge, 893
Possamai-Bastos, Rodrigo, 949
Poves Arjona, Blanca, 937
Pozo Astigarraga, Mikel Eukeni, 77
Prieto Matías, Manuel, 761
Prieto Torralbo, Pablo, 21
Puente Varona, Valentín, 21
Puertas-Martín, Savíns, 69, 259

Quiles, Francisco J., 51, 77
Quintana-Ortí, Enrique S., 639

R. Rull, Elena, 337
Ramírez Treviño, Antonio, 169
Ramos Montes, Marcos, 717
Ramos, Aingeru, 391
Rangel-Cascajosa, Carlos, 845, 901
Rasco Flores, Rebeca, 385
Ravindranath Reddy, Ravikiran, 743
Raygoza Panduro, Juan José, 983, 1001
Reaño, Carlos, 197
Resano, Javier, 675
Rico Gallego, Juan Antonio, 991
Riego del Castillo, Virginia, 331
Rincón, Fernando, 237, 921, 1035
Ríos Navarro, Antonio, 1009
Ripoll Acosta, Alejandro, 575
Rius, Josep, 911
Rivas Becerra, María de Lourdes, 983
Rivas Becerra, María Regina, 983
Rivera Dominguez, Jorge, 1001
Roda-Sanchez, Luis, 957
Rodríguez Herrero, Máximo, 219
Roig Mateu, Concepció, 709
Romero Lorenzo, Alejandro, 971
Romero, Javier, 851, 921
Romero, Luis F., 583
Ros, Alberto, 3, 13, 655, 665, 743
Rossainz López, Mario, 631
Rubira-García, Álvaro, 13

Sacristán, Miguel Angel, 299
Sahuquillo, Julio, 39, 305, 359, 413
Sánchez Calza, Benjamín, 773, 815
Sánchez Gallegos, Genaro, 127
Sánchez González, Lidia, 331
Sánchez Olmedo, Miguel Ángel, 69
Sánchez Rinza, Bárbara, 631
Sánchez-Checa, Paula, 831
Sánchez-Gallegos, Dante D., 89, 107, 211, 219
Sanjuan Estrada, Juan Francisco, 437, 453, 611
Santander-Jiménez, Sergio, 521
Sarrías, Adrián, 529, 537
Segarra Flor, Juan, 169, 377, 421
Seguí Moreno, Juan, 649
Serrano-Cases, Alejandro, 949
Shivdikar, Kaustubh, 821
Singh, Sawan, 743
Siwinska, Patricia, 639
Spaimoc, Radu, 911
Stafford Fernandez, Esteban, 251
Suárez Gracia, Darío, 675, 753
Suárez Labena, Daniel, 619

Tarraga-Moreno, Joaquin, 51

Titos-Gil, Rubén, 13
Toledo Delgado, Pedro, 493, 619
Toledo, F. Javier, 701
Tomás, Andrés E., 639
Tomàs Cuñat, Rosana, 229, 709
Torres Macías, Alitzel Galilea, 169
Torres-Charles, Catherine A., 211
Torres, Enrique, 427
Torres, Yuri, 789
Trejos Hernández, Diógenes Enrique, 859, 865
Trinidad Salvador, Luis Miguel, 807

Usamentiaga Fernández, Rubén, 291

Valenzuela Saz, Diego, 377
Valero, Alejandro, 137, 427, 753
Valle Neila, Jorge, 991
Vega-Rodríguez, Miguel A., 521
Vera-López, Óscar, 821
Verde Jiménez, José Antonio, 265, 275
Vicente Tomás, Germán, 665
Vicente-Díaz, Saturnino, 845, 901
Vigueras, Guillermo, 205, 547
Vilaplana, Jordi, 911
Vilela-Pérez, Nicolás, 31
Villanueva Molina, Félix Jesús, 1035
Villarroya-Gaudó, María, 367
Villarrubia, Jorge, 463
Viñals, Víctor, 427

Wan, Xiangyu, 205, 547
Wolf, Felix, 5

Yáñez Rodríguez, Francisco, 447

Zage, David, 421
Zheng, Haojie, 197