Diseño e Implementación de un Agente en Espacio de Usuario para la Gestión Dinámica de Frecuencia (DVFS) en Sistemas Heterogéneos mediante Modelos Ligeros de Machine Learning

Yeison Adrián Cáceres Torres y Ricardo Andrés Pérez Porras

Trabajo de Grado para Optar al Título de Ingeniero de Sistemas

Director
Gilberto Javier Díaz Toro
Doctor en Ciencias de la Computación

Universidad Industrial de Santander
Facultad de Ingenierías Físico-Mecánicas
Escuela de Ingeniería de Sistemas e Informática
Bucaramanga
2026

Agradecimientos

En primer lugar, agradecemos a Dios por darnos la vida y la salud necesarias para culminar este proyecto de grado, y por permitirnos llegar a esta etapa de nuestras vidas.

Agradecemos a los profesores Gilberto Díaz Toro y Luis Alejandro Torres Niño por su guía, su paciencia y su apoyo a lo largo de este trabajo. Su conocimiento y su experiencia fueron fundamentales para llevarlo a buen término.

A la Universidad Industrial de Santander, por darnos la oportunidad de formarnos como profesionales y por los recursos que hicieron posible este proyecto. A la Universidad de Cartagena, por abrirnos las puertas de su infraestructura de cómputo, donde se realizó buena parte de este trabajo, y por su apoyo a la investigación. Al Grupo de Investigación en Cómputo Avanzado y a Gran Escala (CAGE) y al SC3 por la oportunidad de aprender y compartir conocimientos con otros investigadores y estudiantes.

Yo, Ricardo, agradezco a mi familia y a todas las personas que aportaron su granito de arena en esta etapa de mi vida; en especial, a Herlinda Caicedo y a Luis Sierra, por su apoyo incondicional y su guía en mi crecimiento personal.

Yo, Yeison, agradezco a mi madre, Anyuly; a mi padre, Yeison, y a mi hermano, Santiago, por haber sido siempre mi apoyo y mi sostén durante estos años de formación. A mis amigos, por su compañía y su motivación, y a quienes me acompañaron en estos últimos meses e hicieron más llevadero el proceso. Agradezco también a la UIS por brindarme las condiciones para desarrollar mi vida académica.

Tabla de Contenido

Pág.
Introducción 9
Planteamiento y Justificación del Problema 9
1. Objetivos 12
1.1 Objetivo General 12
1.2 Objetivos Específicos 12
2. Marco de Referencia 13
2.1 Marco Conceptual 13
2.1.1 Potencia CMOS y Fundamento de DVFS 13
2.1.2 Modelo Roofline y Regímenes de Ejecución 14
2.1.3 Telemetría Microarquitectónica con PMU 16
2.1.4 Espacio de Usuario e Interfaces de Observabilidad 17
2.1.5 DVFS, P-states y Mecanismos de Actuación 18
2.1.6 Modelos de Clasificación Ligera e Hiperparámetros 19
2.1.7 Producto Energía-Retardo (EDP) 20
2.2 Estado del Arte 21
3. Metodología 23
3.1 Enfoque y Diseño 23
3.1.1 Plataforma e Instrumento de Medición 24
3.1.2 Modelo Roofline y Etiquetado 27
3.2 Fase 1: Recolección y Caracterización 28
3.2.1 Catálogo, Matriz Experimental y Control de Sesgos 28
3.2.2 Calidad de los Datos, Granularidad y Reproducibilidad 29
3.3 Fase 2: Modelos de Clasificación y Política de Frecuencia 30
3.3.1 Formulación, Variables y Validación 30
3.3.2 Decisión Selectiva, Calibración y Selección de Modelos 32
3.3.3 Invarianza de las Entradas frente a la Acción del Agente 33
3.3.4 Derivación de la Política de Frecuencia 34
3.4 Fase 3: Agente de Control DVFS 34
3.4.1 Arquitectura, Detección de Fase y Parámetros 35
3.4.2 Política, Actuación y Coordinación 36
3.4.3 Aplicaciones Compuestas y Verificación Previa 37
3.5 Fase 4: Validación Experimental 38
3.5.1 Diseño de la Matriz y Escenarios 38
3.5.2 Medición y Análisis 39
4. Resultados 43
4.1 Resultados de CPU 43
4.1.1 Composición de Fase por Kernel y por Frecuencia 44
4.1.2 Techos Roofline y *Ridge* 45
4.1.2.1 Validación de los Techos 46
4.1.3 Clasificador de Fase en CPU 46
4.1.3.1 Variables de Entrada 46

4.1.3.1.1 Invarianza frente a la Acción del Agente 47
4.1.3.2 Selección de Modelo e Hiperparámetros 47
4.1.3.2.1 Búsqueda de Hiperparámetros 48
4.1.3.3 Evaluación del Modelo Final 49
4.1.3.4 Decisión Selectiva y Costo de Inferencia 51
4.1.3.5 Límite de la Exactitud del Clasificador de CPU 52
4.1.4 Política de Frecuencia por Clase en CPU 54
4.1.4.1 Tamaño de Efecto Detectable 56
4.2 Resultados de GPU 57
4.2.1 Etiquetado, Conjunto de Datos y Clasificador de Fase en GPU 57
4.2.2 Candidato Operativo de GPU: Invarianza frente a la Acción del Agente 59
4.2.3 Política de Frecuencia por Clase en GPU 62
4.3 Resultados del Agente de Control 65
4.3.1 Exportación de los Modelos 65
4.3.2 Bloqueo del Reloj de GPU y Premisa Física de la Política 66
4.3.3 Latencias de Conmutación y Sobrecarga 66
4.3.4 Restauración y Coordinación 66
4.3.5 Clasificación de Fase de GPU sobre Aplicaciones Compuestas 67
4.4 Evaluación del Agente frente a la Base 67
4.4.1 Aplicaciones Compuestas y Clasificación de Fase en el Lazo 67
4.4.1.1 Entradas por Ventana frente a Entradas por Corrida 69
4.4.2 Tiempo, Energía, EDP y Sobrecarga del Agente frente a la Base 69
4.4.3 Potencial de la GPU 71
4.4.4 Escenario E: GPU Dominada por Fases de Memoria 73
4.4.4.1 Familias Vistas (E-A) 73
4.4.4.2 Agente frente a F1 Fijo (E-A) 74
4.4.4.3 Familias Inéditas (E-B) 75
4.4.5 Escenario E-C: Fase de Cómputo Sensible al Reloj 76
4.4.6 Aplicaciones Externas: LAMMPS y CloverLeaf 78
4.4.6.1 LAMMPS como Aplicación de Terceros 78
4.4.6.2 CloverLeaf CUDA Fortran como Aplicación Continua 79
4.4.7 Contraste con el Gobernador Nativo powersave 80
5. Discusión 82
5.1 Validez de la Medición 82
5.2 Generalización de los Clasificadores 82
5.3 Alcance de la Ganancia Energética 84
5.4 Contraste con el Estado del Arte 85
6. Conclusiones 86
6.1 Limitaciones 88
6.2 Trabajo Futuro 90
Referencias Bibliográficas 91

Lista de Tablas

Pág.

Tabla 1 Plataforma Experimental 25
Tabla 2 Rejilla Fina de Frecuencia de la Campaña Definitiva, con Pasos Uniformes de 300 MHz en CPU y de 150 MHz en GPU 29
Tabla 3 Brazos del Agente 36
Tabla 4 Parámetros del Agente 36
Tabla 5 Acciones del Agente 37
Tabla 6 Aplicaciones Compuestas con Fronteras y Clases Conocidas por Construcción 38
Tabla 7 Matriz Experimental de la Fase 4 38
Tabla 8 Mapa de los Escenarios de la Fase 4 y Carácter de su Análisis 40
Tabla 9 Amenazas a la Validez de Mayor Peso y Medidas de Control 42
Tabla 10 Configuración Final del Clasificador CPU 49
Tabla 11 Métricas del Clasificador CPU sin Abstención 49
Tabla 12 Métricas del Clasificador GPU sin Abstención 61
Tabla 13 Síntesis de la Política GPU 63
Tabla 14 Composición del Escenario E 73

Lista de Figuras

Pág.

Figura 1 Modelo Roofline Conceptual con el Techo de Ancho de Banda de Memoria y el Techo de Rendimiento de Cómputo 14
Figura 2 Ámbito de Atribución de los Contadores de Núcleo y de *Uncore* Dentro del Paquete del Procesador 17
Figura 3 Jerarquía Mediante la Cual una Política de Rendimiento Solicita Cambios que Posteriormente Son Materializados por el Hardware 19
Figura 4 Recorrido de una Campaña a Través de las Capas de Preparación, Medición y Consolidación del Instrumento 26
Figura 5 Validación Anidada LOFO 32
Figura 6 Arquitectura del Agente: Dos Procesos, una Tabla de Política Compartida y una Señal de Coordinación 35
Figura 7 Porcentaje de Intervalos Elegibles Clasificados *Compute-bound* y *Memory-bound* por Familia Algorítmica 44
Figura 8 *Ridge* Roofline según el Nivel de Frecuencia CPU 45
Figura 9 Comparación de Modelos del Clasificador CPU 48
Figura 10 Matrices de Confusión del Modelo Final 50
Figura 11 Resultado del Clasificador CPU por Familia 50
Figura 12 Calibración y Compromiso de la Abstención en CPU 52
Figura 13 Perfiles de Celdas Confusables entre Clases Opuestas 53
Figura 14 Ganancia del EDP frente a la Base por Nivel y Clase (Cómputo: 19 Kernels; Memoria: 28), con IC95 por *Bootstrap* de Kernels; Eje Simétrico-logarítmico 55
Figura 15 Energía, Tiempo y EDP por Nivel de Frecuencia en CPU 56
Figura 16 Potencia de Paquete y DRAM frente a la Frecuencia 56
Figura 17 Comparación de Modelos Candidatos GPU 59
Figura 18 Invarianza de los Candidatos GPU frente al Reloj 60
Figura 19 Matrices de Confusión del *Random Forest* de GPU 62
Figura 20 Ganancia del EDP GPU por Nivel y Clase, en Forma de *Forest Plot* 64
Figura 21 Energía, Tiempo y EDP por Nivel de Frecuencia en GPU 65
Figura 22 Ciclo de Cada Aplicación Compuesta 68
Figura 23 EDP CPU+GPU Relativo a la Base en la Matriz Inicial 71
Figura 24 Potencial de la GPU en Fases *Memory-bound* 72
Figura 25 Escenario E: EDP CPU+GPU y Energía de GPU Bajo Ambos Gobernadores 76
Figura 26 Agente frente a F1 Fijo en E-A y E-C 78
Figura 27 EDP de LAMMPS por Entrada, Brazo y Gobernador 79
Figura 28 Resultados de CloverLeaf frente a la Base 80

Resumen

Título: Diseño e Implementación de un Agente en Espacio de Usuario para la Gestión Dinámica de Frecuencia (DVFS) en Sistemas Heterogéneos mediante Modelos Ligeros de Machine Learning[*]

Autor: Yeison Adrián Cáceres Torres, Ricardo Andrés Pérez Porras[**]

Palabras Clave: DVFS, eficiencia energética, computación de alto rendimiento, modelo Roofline, telemetría microarquitectónica, aprendizaje automático supervisado

Descripción: El consumo energético limita la computación de alto rendimiento (HPC) en nodos CPU-GPU. El Escalado Dinámico de Voltaje y Frecuencia (DVFS) permite controlarlo, pero los gobernadores nativos de Linux no distinguen si el rendimiento está limitado por cómputo o memoria.

Se diseñó, implementó y evaluó un agente en espacio de usuario que identifica ese régimen mediante contadores PMU en CPU y NVML en GPU, y ajusta la frecuencia para reducir el Producto Energía-Retardo (EDP). Se construyó un conjunto de datos etiquetado con Roofline y se entrenaron clasificadores ligeros, validados dejando fuera cada familia algorítmica, con exactitud balanceada por celda de 0.728 (XGBoost, CPU) y 0.819 (*random forest*, GPU). Se derivaron políticas por régimen y se evaluó el agente frente a los gobernadores *performance* y *powersave*.

En la asignación evaluada, cerca del 84 % de la potencia de paquete y DRAM permaneció al reducir la frecuencia de los seis núcleos delegados; ningún nivel inferior mejoró consistentemente el EDP. La política de CPU fue no actuar y la sobrecarga de su proceso empeoró el EDP CPU+GPU. En kernels *memory-bound* de GPU, fijar 1260 MHz mejoró el EDP en 8.9 %.

En la aplicación compuesta de familias vistas y en CloverLeaf, el proceso de GPU redujo el EDP CPU+GPU en 3.8 y 3.9 % bajo *performance*, y en 2.6 y 3.2 % bajo *powersave*, con cinco bloques favorables ($p = 0.0625$). En la compuesta, 1260 MHz fijos igualaron al agente; en un escenario construido con cómputo sensible al reloj, este superó al reloj fijo en 2.2 a 2.8 % (seis bloques, $p = 0.031$; prueba de mecanismo). En LAMMPS y familias no vistas quedó cerca de la paridad. La ventaja de clasificar por fase requiere validación externa adicional.

[*] Trabajo de Grado.

[**] Facultad de Ingenierías Físico-Mecánicas, Escuela de Ingeniería de Sistemas e Informática. Director: Gilberto Javier Díaz Toro, Doctor en Ciencias de la Computación.

Abstract

Title: Design and Implementation of a User-Space Agent for Dynamic Frequency Management (DVFS) in Heterogeneous Systems Using Lightweight Machine Learning Models*

Author: Yeison Adrián Cáceres Torres, Ricardo Andrés Pérez Porras**

Key Words: DVFS, energy efficiency, high performance computing, Roofline model, microarchitectural telemetry, supervised machine learning

Description: Energy consumption limits high performance computing (HPC) on CPU-GPU nodes. Dynamic Voltage and Frequency Scaling (DVFS) can control it, but native Linux governors do not distinguish whether computation or memory limits performance.

A user-space agent was designed, implemented and evaluated to identify this regime using CPU PMU counters and GPU NVML measurements, and adjust frequency to reduce the Energy-Delay Product (EDP). A Roofline-labeled dataset was built and lightweight classifiers were trained, validated by leaving each algorithmic family out, achieving per-cell balanced accuracy of 0.728 (XGBoost, CPU) and 0.819 (*random forest*, GPU). Policies were derived for each regime and the agent was evaluated against the *performance* and *powersave* governors.

In the evaluated allocation, about 84 % of package and DRAM power remained when the frequency of the six delegated cores was lowered; no lower level consistently improved EDP. The CPU policy was not to act, and its process overhead worsened CPU+GPU EDP. On *memory-bound* GPU kernels, fixing the clock at 1260 MHz improved EDP by 8.9 %.

On the composite application with seen families and on CloverLeaf, the GPU process reduced CPU+GPU EDP by 3.8 and 3.9 % under *performance*, and by 2.6 and 3.2 % under *powersave*, with all five blocks favorable ($p = 0.0625$). On the composite application, a fixed 1260 MHz clock matched the agent; in a constructed scenario with clock-sensitive computation, the agent outperformed the fixed clock by 2.2 to 2.8 % (six blocks, $p = 0.031$; mechanism test). On LAMMPS and unseen families, it stayed close to parity. The advantage of classifying execution phases requires further external validation.

*Degree Work.

**Faculty of Physical-Mechanical Engineering, School of Systems Engineering and Computer Science. Director: Gilberto Javier Díaz Toro, PhD in Computer Science.

Introducción

En los sistemas de cómputo de alto rendimiento (HPC), el consumo energético limita el escalamiento, en particular en nodos heterogéneos que combinan CPU y GPU. El Escalado Dinámico de Voltaje y Frecuencia (DVFS) es el principal mecanismo de control en el hardware: permite ajustar el estado operativo de cada procesador para equilibrar la energía consumida y el tiempo de ejecución. Su efectividad depende, sin embargo, de que la frecuencia elegida corresponda al comportamiento de la aplicación, que cambia a lo largo de la ejecución.

Este problema se ha abordado con los gobernadores de frecuencia del sistema operativo y con heurísticas basadas en métricas microarquitectónicas. Los gobernadores de Linux deciden a partir de la utilización del procesador o de una política fija, sin considerar qué limita el rendimiento de la carga. Las heurísticas usan reglas de bajo costo, pero dependen de umbrales estáticos que generalizan mal cuando las fases de la aplicación cambian con rapidez. Aunque en el estado del arte existen estrategias que actúan por fase (capítulo 2), los gobernadores nativos evaluados aquí no incorporan de forma explícita si una fase está limitada por la capacidad de cómputo del procesador (*compute-bound*) o por la transferencia de datos desde memoria (*memory-bound*), y de esa distinción depende cuánto se alarga la ejecución al reducir la frecuencia. Esta brecha motiva inferir el régimen durante la ejecución y decidir la frecuencia a partir de él.

En este contexto, el presente trabajo presenta el diseño, la implementación y la evaluación de un agente en espacio de usuario basado en modelos ligeros de aprendizaje automático. El agente clasifica la fase de ejecución como *compute-bound* o *memory-bound* a partir de telemetría microarquitectónica y ajusta la frecuencia de CPU y GPU para reducir el Producto Energía-Retardo (EDP). La evaluación lo compara con los gobernadores nativos de Linux que expone la plataforma y cuantifica el ahorro de energía, la degradación del rendimiento y el costo de observar e inferir.

Planteamiento y Justificación del Problema

La operación de los sistemas de alto rendimiento (HPC) presenta una ineficiencia energética, en particular en infraestructuras que integran aceleradores gráficos (GPU) para ejecutar apli-

caciones científicas intensivas. Cuando una aplicación entra en una fase donde el procesamiento depende principalmente de la transferencia de datos desde memoria, mantener la CPU o la GPU a su frecuencia máxima no mejora el rendimiento de manera proporcional. En estas condiciones, los núcleos de cómputo permanecen parcialmente inactivos esperando datos, lo que genera un consumo energético elevado sin un beneficio equivalente en tiempo de ejecución. Esta desalineación entre demanda computacional y frecuencia operativa ha motivado el estudio del escalado de frecuencia como mecanismo para mejorar la eficiencia energética en aplicaciones científicas (Simsek et al., 2024) y en sistemas de gran escala donde la gestión de potencia impacta el comportamiento global del sistema (Costa et al., 2025).

El problema se agrava por la naturaleza multifásica de las aplicaciones HPC, cuyo perfil de ejecución cambia a lo largo del tiempo de ejecución. En particular, estas aplicaciones alternan entre fases *compute-bound* y fases *memory-bound*. De esta distinción depende la respuesta a DVFS: en el régimen *memory-bound*, los incrementos de frecuencia producen mejoras marginales en el tiempo de ejecución, mientras aumentan el consumo energético. En fases *compute-bound*, en cambio, la frecuencia sí impacta el desempeño de forma más directa. Por eso se han propuesto marcos de caracterización para identificar y clasificar estas fases en entornos HPC, y así habilitar decisiones de control más informadas (Antici et al., 2024; Calore et al., 2017).

A partir de este contexto, la gestión eficiente de frecuencia no depende únicamente de la disponibilidad del mecanismo DVFS en el hardware, sino de la capacidad del sistema para decidir, en tiempo de ejecución, cuál configuración resulta más conveniente según el comportamiento de la carga. Aunque se han propuesto mecanismos automatizados para seleccionar frecuencias óptimas de GPU con el fin de mejorar la eficiencia energética sin comprometer significativamente el desempeño (Ali et al., 2023), persiste una brecha en la implementación de estrategias que integren de forma coordinada componentes CPU-GPU y que operen durante la ejecución de la aplicación, considerando además el costo computacional asociado al propio proceso de decisión.

Esta brecha es relevante en entornos académicos y científicos, donde el consumo eléctrico incide directamente en los costos operativos, la disponibilidad efectiva de recursos y la sostenibili-

dad institucional. Un mecanismo capaz de observar el comportamiento de la aplicación, inferir su fase de ejecución y ajustar la frecuencia de operación sin intervenir el código fuente es, por eso, una alternativa con valor técnico y práctico.

En las infraestructuras académicas de cómputo científico, como el clúster universitario en el que se desarrolló este trabajo, un mecanismo que combine telemetría de bajo nivel, modelos ligeros de aprendizaje automático y control en espacio de usuario puede incorporarse sin modificar las aplicaciones de los usuarios. Por eso el problema es también pertinente en términos de aplicabilidad local.

A partir de esta necesidad de optimización energética en sistemas heterogéneos, se formula la siguiente pregunta de investigación que servirá como eje de desarrollo del proyecto:

¿En qué medida el diseño e implementación de un agente en espacio de usuario, guiado por modelos ligeros de aprendizaje automático, permite ajustar la frecuencia de operación (DVFS) en sistemas heterogéneos CPU-GPU para reducir el consumo energético, sin que el costo de la inferencia computacional degrade el rendimiento global de la aplicación, en comparación con los gobernadores nativos de Linux?

1. Objetivos

1.1 Objetivo General

Caracterizar, diseñar, implementar y evaluar un agente en espacio de usuario basado en modelos ligeros de Aprendizaje Automático para el ajuste dinámico de frecuencia (DVFS) en sistemas heterogéneos CPU-GPU, con el propósito de maximizar la eficiencia energética en aplicaciones científicas sin inducir una penalización severa en su rendimiento global.

1.2 Objetivos Específicos

- Caracterizar el comportamiento computacional y el consumo energético de cargas de trabajo representativas (intensivas en cómputo y en memoria) bajo distintos estados de frecuencia, recolectando telemetría de bajo nivel mediante contadores de rendimiento por hardware e interfaces de potencia estándar (Perf y RAPL para CPU y NVML para GPU).

- Entrenar y validar modelos clásicos de Aprendizaje Automático (tales como Árboles de Decisión o Bosques Aleatorios) que sean capaces de clasificar, en tiempo de ejecución y con baja latencia de inferencia, las fases de ejecución de las aplicaciones basándose en la telemetría extraída.

- Desarrollar un servicio de control (*daemon*) en el espacio de usuario que interactúe de forma asíncrona con el sistema operativo, leyendo el estado de los contadores de hardware, ejecutando la inferencia del modelo clasificador y aplicando políticas proactivas de DVFS a través de las interfaces estándar del sistema operativo, en función de la fase de ejecución inferida por el modelo.

- Evaluar el impacto empírico del sistema propuesto mediante el cálculo del Producto Energía-Retardo (EDP), con el fin de determinar estadísticamente si el ahorro energético compensa la sobrecarga computacional (*overhead*) derivado de la inferencia del modelo en comparación con los gobernadores nativos de Linux.

2. Marco de Referencia

2.1 Marco Conceptual

El marco conceptual reúne solo lo que el resto del trabajo utiliza. Parte de la relación entre frecuencia, voltaje, potencia y tiempo de ejecución, y del modelo Roofline, que distingue los regímenes *compute-bound* y *memory-bound* de una carga. Las señales de la PMU y de otros dominios del sistema alimentan un clasificador ligero que infiere ese régimen, y el Producto Energía-Retardo (EDP) permite valorar el efecto de las decisiones de frecuencia que se toman con esa inferencia.

2.1.1 Potencia CMOS y Fundamento de DVFS

La potencia dinámica de un circuito CMOS depende del factor de actividad, la capacitancia efectiva, el voltaje de alimentación y la frecuencia de operación. Esta relación es la base física de DVFS y se expresa como:

$$P_{\text{dyn}} = \alpha \cdot C \cdot V^2 \cdot f \tag{1}$$

donde α representa el factor de actividad, C la capacitancia efectiva conmutada, V el voltaje de alimentación y f la frecuencia de operación (Gonzalez et al., 1997). Para trabajar a frecuencias más altas normalmente se necesita un mayor voltaje. Por eso, cuando se reduce la frecuencia también es posible reducir el voltaje, y como el voltaje aparece al cuadrado en la ecuación, la potencia dinámica baja de forma marcada.

Sin embargo, la reducción de la potencia no asegura un menor consumo de energía, porque la energía también depende del tiempo de ejecución. Si la aplicación tarda más en terminar a una frecuencia menor, parte del ahorro se pierde. Además, la potencia estática se consume durante toda la ejecución. Por eso el efecto energético de cambiar la frecuencia depende de cómo responda la carga, y esto motiva clasificar el régimen de cada carga.

2.1.2 Modelo Roofline y Regímenes de Ejecución

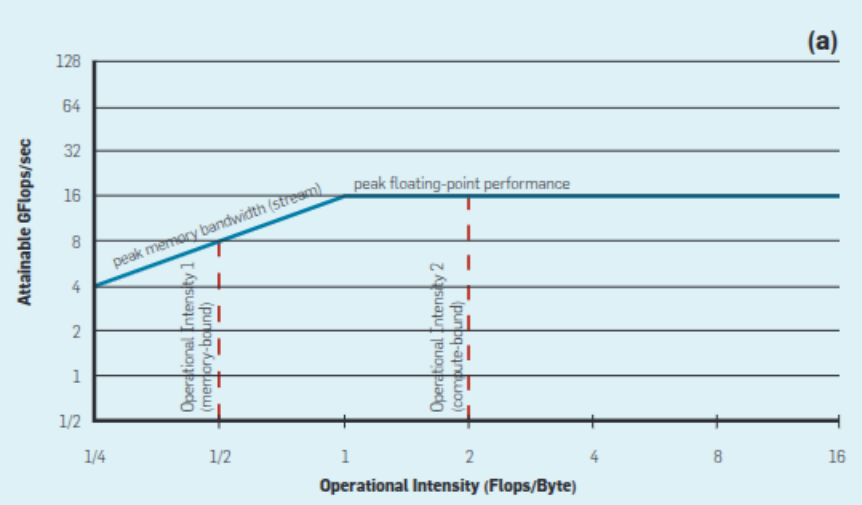
El modelo Roofline relaciona el rendimiento alcanzable de una carga con la capacidad de cómputo y el ancho de banda de memoria de la plataforma. La intensidad operacional decide cuál de los dos límites condiciona el rendimiento, según la expresión:

$$P = \text{mín}\,(P_{\text{pico}},\ I \cdot BW_{\text{pico}}) \tag{2}$$

donde P_{pico} es el rendimiento aritmético pico, BW_{pico} el ancho de banda pico de memoria e I la intensidad operacional (Williams et al., 2009). Si el rendimiento queda limitado por $I \cdot BW_{\text{pico}}$, la carga es *memory-bound*. Si el límite es P_{pico}, la carga es *compute-bound*. La Figura 1 muestra esta separación entre ambos regímenes.

Figura 1

Modelo Roofline Conceptual con el Techo de Ancho de Banda de Memoria y el Techo de Rendimiento de Cómputo



Nota. Tomado de Williams et al. (2009, Fig. 1(a)).

La intensidad operacional es la cantidad de operaciones por cada byte transferido desde el nivel de memoria considerado (Williams et al., 2009, p. 66). No depende solo del tamaño del conjunto de datos ni de que este quepa en caché. La reutilización de datos reduce el tráfico hacia memoria y aumenta la intensidad operacional, incluso cuando el conjunto de trabajo supera la caché

(Goto y van de Geijn, 2008, Secs. 3-4). El modelo fija un límite de rendimiento para cada intensidad, pero una carga puede quedar por debajo de ese límite por otros efectos de la ejecución.

Los dos regímenes se separan en el punto de inflexión , o *ridge point*, donde el techo de cómputo y el límite de ancho de banda tienen el mismo valor. El umbral se obtiene como:

$$I_{\text{ridge}} = \frac{P_{\text{pico}}}{BW_{\text{pico}}} \tag{3}$$

Una intensidad menor que I_{ridge} corresponde al régimen *memory-bound* y una mayor, al *compute-bound* (Williams et al., 2009). El *ridge point* es una frontera entre regímenes y no una frecuencia óptima de ejecución. Si cambia alguno de los techos, cambia también la posición de la frontera.

El techo de cómputo depende de la capacidad aritmética disponible, de la precisión de las operaciones y de la frecuencia de las unidades de cómputo (Antepara et al., 2025). El ancho de banda de memoria pertenece a otro dominio de la arquitectura y no cambia necesariamente en la misma proporción cuando se modifica la frecuencia (Ali et al., 2023; Guerreiro et al., 2019). Por eso el I_{ridge} usado para clasificar debe corresponder a la precisión y al punto de operación de la observación. Los valores nominales sirven de referencia, pero la clasificación requiere techos medidos en las condiciones de ejecución (National Energy Research Scientific Computing Center [NERSC], 2026).

La intensidad operacional observada se calcula con el trabajo aritmético y el tráfico hacia memoria medidos durante el mismo intervalo. Para un intervalo i es:

$$I_i = \frac{\text{FLOPs}_i}{B_i^{\text{DRAM}}} \tag{4}$$

donde FLOPs_i son las operaciones realizadas y B_i^{DRAM} los bytes transferidos desde o hacia la memoria principal en ese intervalo (Williams et al., 2009). Los eventos que dan los FLOPs dependen de la microarquitectura y se interpretan según la semántica de la PMU (Weaver y McKee, 2008). El tráfico de DRAM se obtiene de contadores del controlador de memoria, cuyo ámbito puede diferir

del de los contadores de núcleo. Ambas mediciones deben corresponder al mismo intervalo para que la intensidad operacional conserve su significado físico.

2.1.3 Telemetría Microarquitectónica con PMU

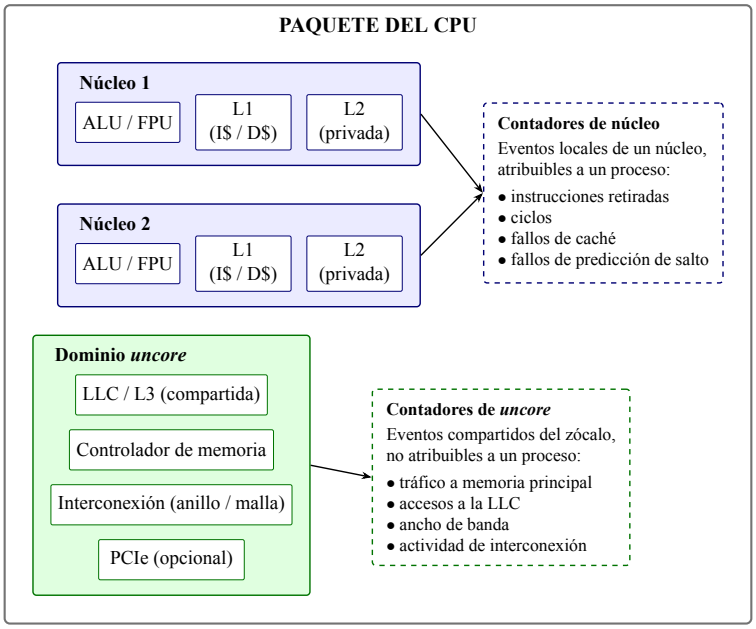
La clasificación en CPU requiere señales que describan el comportamiento de la ejecución sin reconstruir en línea la intensidad operacional, cuya medición exige condiciones que un agente en operación no puede asumir (sección 3.3.1). Las Unidades de Monitorización de Rendimiento (PMU) las proporcionan mediante contadores de hardware que acumulan eventos como instrucciones retiradas, ciclos y actividad de la jerarquía de memoria. Los contadores fijos registran eventos arquitectónicos predefinidos y los programables se configuran para eventos específicos de la microarquitectura (Intel Corporation, 2026a; Weaver, 2015). Una PMU no interpreta el comportamiento de la aplicación, porque entrega conteos cuyo valor depende del evento y del dominio observado. El mismo número solo adquiere significado al conocer qué recurso cuenta, durante qué intervalo y con qué alcance.

La multiplexación ocurre cuando se solicitan más eventos de los que el hardware puede medir al mismo tiempo. El sistema alterna grupos de eventos, cada contador queda activo solo durante una parte de la ventana y el valor registrado se ajusta según el tiempo que estuvo activo (Linux man-pages project, 2026; Liu et al., 2024). Esa estimación pierde precisión si la aplicación cambia de fase mientras un evento no se mide. Por eso conviene limitar la cantidad de señales simultáneas, y esa limitación fija el presupuesto de eventos del instrumento.

Los contadores también difieren en su dominio. Los de núcleo observan recursos asociados a un núcleo y pueden vincularse con la ejecución de una tarea. Los de *uncore* cubren componentes compartidos del zócalo, como el controlador de memoria integrado y partes de la interconexión, y desde ellos se observa el movimiento de datos hacia memoria principal. Como su alcance es compartido, el tráfico que registran no se puede atribuir a un proceso individual sin condiciones de aislamiento. Además, ambas fuentes pueden tener granularidades temporales distintas y no deben combinarse fila a fila sin establecer antes una correspondencia temporal. La Figura 2 muestra esta diferencia estructural.

Figura 2

Ámbito de Atribución de los Contadores de Núcleo y de Uncore Dentro del Paquete del Procesador



2.1.4 Espacio de Usuario e Interfaces de Observabilidad

Un agente en espacio de usuario no accede directamente a los recursos protegidos del sistema, como los contadores de hardware o los registros de frecuencia (Kerrisk, 2010, cap. 2). Depende de las interfaces que el sistema operativo expone para realizar operaciones controladas (McCarty, 2015). Esta condición determina cómo el agente consulta contadores, lee energía y solicita cambios de frecuencia sin modificar estructuras internas del kernel.

En Linux, los eventos de la PMU se obtienen con `perf_event_open`, que permite configurar los contadores y consultar sus valores durante la ejecución (Linux man-pages project, 2026). Su disponibilidad depende del soporte del procesador y de los permisos del sistema.

Estos conteos describen la actividad del procesador, pero no la energía. Para medirla se usa Intel RAPL (*Running Average Power Limit*), que expone contadores de energía de varios dominios, entre ellos el paquete del procesador y, cuando está disponible, la memoria DRAM (David et al., 2010; Thamm y Leser, 2025). Sus valores son energía acumulada, y la diferencia entre dos lecturas da el consumo de un intervalo, que luego se relaciona con métricas como el EDP. En GPU, la biblio-

teca NVML (*NVIDIA Management Library*) expone la utilización del dispositivo y de su memoria, la potencia, los relojes y la energía acumulada, y permite fijar el reloj de los multiprocesadores con privilegios administrativos (NVIDIA, 2026c).

2.1.5 DVFS, P-states y Mecanismos de Actuación

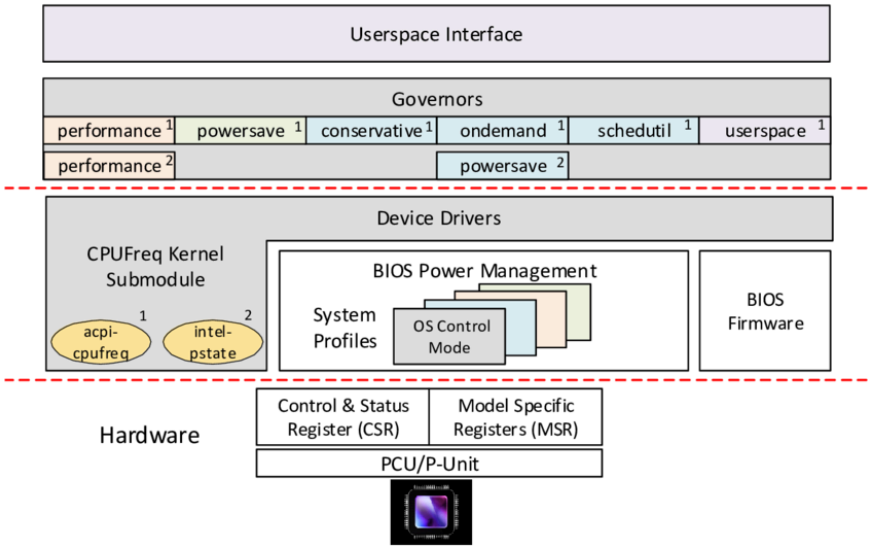
El escalado dinámico de voltaje y frecuencia (*Dynamic Voltage and Frequency Scaling*, DVFS) modifica el punto operativo del procesador para adaptar su capacidad de cómputo y su consumo a la carga. Los mecanismos de gestión de potencia reciben una solicitud de rendimiento y la materializan sobre el hardware (Hebbar y Milenković, 2022).

En CPU, los *Performance States* o P-states son niveles de rendimiento con el procesador activo. P0 es el mayor y los siguientes reducen la capacidad disponible. Sobre ellos operan los *governors*, políticas de software que deciden qué nivel solicitar según el estado del sistema. En Linux, `performance` y `powersave` favorecen extremos opuestos del rango, `ondemand`, `conservative` y `schedutil` ajustan la solicitud de forma dinámica, y `userspace` permite que una aplicación pida el nivel deseado (Hebbar y Milenković, 2022; Linux Kernel Documentation, 2026a).

La Figura 3 muestra el recorrido de una solicitud desde la interfaz de espacio de usuario, pasando por las políticas y los controladores del sistema, hasta los mecanismos que actúan sobre el hardware. Por eso una decisión de rendimiento no se traduce necesariamente de forma inmediata en la frecuencia observada, y el agente debe verificarla después de cada actuación.

Figura 3

Jerarquía Mediante la Cual una Política de Rendimiento Solicita Cambios que Posteriormente Son Materializados por el Hardware



Nota. Tomado de Hebbar y Milenković (2022).

En GPU, DVFS mantiene el mismo principio, pero la actuación suele hacerse sobre dominios de reloj distintos, principalmente el de cómputo y el de memoria . Los niveles de frecuencia disponibles y la forma de aplicar una solicitud dependen de los mecanismos de gestión de la arquitectura y del controlador (Ali et al., 2023; Guerreiro et al., 2019). Como en CPU, la transición entre niveles toma tiempo, por lo que el cambio de frecuencia no es instantáneo (Velicka et al., 2025).

2.1.6 Modelos de Clasificación Ligera e Hiperparámetros

Las observaciones obtenidas al ejecutar cargas acotadas, como *kernels* (Williams et al., 2009), *benchmarks* y *microbenchmarks* (Gregg, 2020), se representan mediante un conjunto de características y una clase asociada. Esa relación se aprende con clasificación supervisada para inferir la clase de nuevas observaciones. Cuando la inferencia debe tener un costo reducido conviene usar modelos ligeros, cuya complejidad mantiene un tiempo de inferencia bajo sobre pocas características.

La regresión logística establece una frontera lineal de bajo costo y sirve como referencia (James et al., 2021, cap. 4). Un árbol de decisión introduce relaciones no lineales mediante divi-

siones sucesivas y es fácil de interpretar, aunque su complejidad crece con la profundidad (James et al., 2021, cap. 8). *Random forest* combina varios árboles generados por muestreo y selección aleatoria de características (Breiman, 2001). Extra-Trees agrega aleatoriedad en los puntos de corte (Geurts et al., 2006). XGBoost construye los árboles de forma secuencial, de modo que cada uno corrige errores de los anteriores (Chen y Guestrin, 2016). Estas familias cubren desde una frontera lineal hasta ensambles de árboles, y en todas el costo de inferencia depende de la configuración.

Los hiperparámetros, como el número de árboles, la profundidad o la tasa de aprendizaje, modifican la capacidad de ajuste y la complejidad del modelo. Optuna explora ese espacio de forma secuencial (Akiba et al., 2019) y usa el estimador de Parzen estructurado (TPE) para proponer configuraciones con mayor probabilidad de mejorar el objetivo (Bergstra et al., 2011), sin recorrer todas las combinaciones.

2.1.7 Producto Energía-Retardo (EDP)

Los cambios de frecuencia mediante DVFS afectan tanto el consumo de energía como el tiempo de ejecución. Por eso reducir una de estas magnitudes no significa necesariamente que el resultado sea mejor. El Producto Energía-Retardo (*Energy-Delay Product*, EDP) relaciona ambas en una sola expresión:

$$\mathrm{EDP} = E \cdot T \quad [\mathrm{J} \cdot \mathrm{s}], \tag{5}$$

donde E representa la energía consumida, en julios, y T el tiempo de ejecución, en segundos (Laros et al., 2013). Un EDP menor indica una relación más favorable entre energía y tiempo. Se eligió el EDP y no la energía sola porque esta última premia cualquier reducción de frecuencia que ahorre julios aunque alargue la ejecución, y el objetivo del trabajo pide no penalizar de forma severa el rendimiento. Variantes como $E \cdot T^2$ dan todavía más peso al tiempo, y una restricción explícita de pérdida de rendimiento exigiría fijar un límite de tolerancia arbitrario; el EDP pondera energía y tiempo por igual sin ese parámetro. Como dos configuraciones pueden tener un EDP parecido con energía y tiempo muy distintos, ambas magnitudes se revisan también por separado.

2.2 Estado del Arte

La gestión energética en sistemas de alto rendimiento ha pasado de enfoques estáticos de reducción de potencia a estrategias adaptativas que buscan preservar el rendimiento sin costos energéticos desproporcionados. El escalado dinámico de voltaje y frecuencia (DVFS) es uno de los mecanismos más estudiados para modificar el punto operativo de procesadores y aceleradores. Su aplicación no da el mismo resultado en todas las cargas, porque una reducción de frecuencia puede disminuir la potencia instantánea, pero también prolongar el tiempo de ejecución y anular el beneficio energético. Por eso buena parte de la investigación pasó de aplicar DVFS de forma uniforme a reconocer el comportamiento de la carga y ajustar la frecuencia según sus características.

Los primeros trabajos que analizaron DVFS sobre regiones concretas de una aplicación mostraron que mantener una misma frecuencia durante toda la ejecución podía desaprovechar ahorros. Calore et al. (2017) estudiaron kernels y una aplicación HPC completa sobre una CPU Intel Haswell y una GPU NVIDIA K80, y observaron que el efecto energético variaba entre regiones y arquitecturas. Evaluaron cambios de frecuencia a nivel de función y mostraron que la conveniencia de cada configuración dependía de la región ejecutada y de la plataforma.

Esa dependencia de la carga llevó a buscar formas de anticipar su respuesta sin recorrer todas las frecuencias. Guerreiro et al. (2019) estudiaron aplicaciones GPGPU con métricas recolectadas en una configuración inicial y las usaron para estimar la respuesta ante cambios en los dominios de cómputo y memoria. Así redujeron la exploración del espacio DVFS y mostraron que esas métricas permiten agrupar aplicaciones por su sensibilidad a la frecuencia.

Ali et al. (2023) propusieron un método automatizado y portable para seleccionar frecuencias de GPU a partir de métricas tomadas en la configuración predeterminada del dispositivo. Con ellas construyeron modelos de potencia y de tiempo de ejecución para estimar el comportamiento en otros estados DVFS sin ejecutar la aplicación en cada uno. La selección consideró juntos la energía y la degradación del desempeño, de modo que una menor potencia con un aumento importante del tiempo no contaba como mejora. En las plataformas evaluadas, la propuesta redujo la energía en promedio 29.6 % con una pérdida de rendimiento de 5.2 % en NVIDIA GA100, y 22.6 % con

una pérdida de 4.7 % en NVIDIA GV100 (Ali et al., 2023).

Estos enfoques reducen el espacio de búsqueda, pero parten de una caracterización previa con la que se anticipa el comportamiento posterior de la aplicación. Cuando la carga cambia durante la ejecución hace falta reconocer el régimen que domina en cada momento. Antici et al. (2024) abordaron este problema con MCBound, que clasifica cargas como *memory-bound* o *compute-bound* a partir de información de ejecuciones HPC para caracterizar trabajos nuevos. Con datos del supercomputador Fugaku, la separación alcanzó una puntuación F1-macro de al menos 0.89 con un costo reducido para el sistema (Antici et al., 2024). Este resultado indica que el régimen de una carga puede inferirse de forma automática, sin clasificarla a mano.

MCBound es cercano al problema de identificar el régimen, aunque no gobierna los estados DVFS de CPU o GPU. Aun así, separa dos partes que aparecen en los trabajos revisados, primero reconocer el comportamiento de la carga y luego decidir cómo usar esa información para modificar el punto de operación. La segunda parte se ha explorado con aproximaciones que adaptan la frecuencia durante la ejecución.

Simsek et al. (2024) estudiaron esta actuación dinámica en simulaciones astrofísicas sobre GPU. Instrumentaron SPH-EXA para medir el comportamiento energético de distintas regiones y modificar la frecuencia del acelerador en puntos determinados de la ejecución. Frente a una frecuencia uniforme, redujeron el consumo por GPU hasta 7.82 % con una degradación del rendimiento de aproximadamente 2.95 %. Esto muestra el beneficio de cambiar la frecuencia en ejecución, pero la decisión queda ligada a regiones instrumentadas de la aplicación.

DRLCAP es un marco de *frequency capping* para GPU basado en aprendizaje por refuerzo profundo. Monitoriza información a nivel de sistema, detecta cambios de fase y ajusta el límite de frecuencia sin intervención manual, con una reducción promedio de 22 % en la energía de GPU y menos de 3 % de degradación de rendimiento en arquitecturas NVIDIA (Wang et al., 2024). El resultado muestra que el control de frecuencia puede automatizarse con señales observadas en línea, aunque con aprendizaje por refuerzo profundo y solo en GPU.

En CPU existen sistemas de ejecución que ya ajustan la frecuencia durante la ejecución de

aplicaciones HPC en producción. EAR (Corbalan et al., 2020) identifica de forma transparente las regiones iterativas de la aplicación, calcula en cada iteración una firma de métricas y aplica modelos de energía para estimar el tiempo y la potencia en cada frecuencia de CPU disponible, y elige la frecuencia según una política energética. GEOPM (Eastep et al., 2017) es un marco jerárquico de código abierto para la gestión de energía y potencia en tiempo de ejecución, extensible mediante complementos, que mejoró hasta 30 % el tiempo de solución de aplicaciones de referencia en un clúster Xeon Phi. COUNTDOWN (Cesarini et al., 2021) reduce el estado de potencia de la CPU durante las fases de comunicación y sincronización MPI, con ahorros de 6 a 50 % de energía en los NAS Parallel Benchmarks y una penalización de tiempo menor de 5 %. Estos sistemas actúan sobre la CPU y se apoyan en la estructura de la aplicación, como sus iteraciones o sus llamadas MPI, y no en una clasificación explícita del régimen Roofline.

Gran parte de los enfoques recientes de predicción y control dinámico se concentra en GPU, y CPU y GPU difieren tanto en sus mecanismos de frecuencia como en las señales disponibles para observar la ejecución. Calore et al. (2017) consideran ambas arquitecturas, pero Guerreiro et al. (2019), Ali et al. (2023), Simsek et al. (2024) y Wang et al. (2024) se centran en aceleradores gráficos, y los sistemas de ejecución de CPU citados no gobiernan la GPU. Queda espacio para estrategias que traten CPU y GPU en un mismo nodo y usen una clasificación de bajo costo del régimen de ejecución como apoyo a las decisiones DVFS, sin instrumentación específica de cada aplicación.

3. Metodología

3.1 Enfoque y Diseño

La investigación es cuantitativa, experimental y aplicada. Es experimental porque modifica de forma controlada los estados de frecuencia de CPU y GPU para medir su efecto sobre el tiempo de ejecución, la energía y el Producto Energía-Retardo. Es aplicada porque, a partir de esas me-

diciones, se desarrolla y evalúa un agente de software para la gestión dinámica de frecuencia. El estado del dispositivo sin actuación del agente es el **escenario base**, identificado como `REF` en la rejilla de frecuencias, las tablas por nivel y las figuras. En la Fase 4, esta referencia se mide con los dos gobernadores nativos que expone `intel_pstate` (Linux Kernel Documentation, 2026b) en la plataforma: *performance*, con una frecuencia fija de 3.2 GHz en los núcleos delegados, y *powersave*, con el perfil de energía-rendimiento (EPP) en *default* y un rango de 0.8 a 3.2 GHz. En ambos casos el turbo de CPU permanece desactivado, salvo en la serie de ocho bloques de E-B bajo *powersave*, cuya base se midió con el turbo activado y un brazo adicional sin turbo cuantificó su efecto (sección 4.4.7), y la GPU conserva la gestión automática del controlador. El turbo se desactiva para evitar ajustes autónomos que impidan verificar la frecuencia efectiva durante la medición. La eficiencia energética del objetivo general se mide con el EDP (sección 2.1.7), que en las Fases 2 y 4 se compara siempre con la base del mismo dispositivo y gobernador; la penalización del rendimiento se reporta aparte como la razón de duraciones frente a esa base. El trabajo se organiza en cuatro fases encadenadas. La primera construye el conjunto de datos etiquetado. La segunda entrena y valida los clasificadores, los exporta a ONNX y deriva las tablas de política. La tercera implementa el agente y la cuarta lo evalúa experimentalmente. El material complementario de este capítulo está en el repositorio del proyecto (sección 3.2.2).

Se estudió una muestra intencional de cargas científicas reproducibles para CPU y GPU, elegidas para representar regímenes de ejecución contrastantes. Cada carga debía verificar su resultado, contener trabajo de punto flotante medible por el instrumento y permitir una intensidad operacional válida para el etiquetado. La diversidad y la validación se definieron por familia algorítmica, y las variantes de tamaño de un mismo algoritmo no se contaron como familias independientes. El catálogo y su proceso de selección se presentan en la sección 3.2.1.

3.1.1 Plataforma e Instrumento de Medición

Todos los experimentos se ejecutaron en un único nodo con asignación exclusiva mediante Slurm, para que cargas ajenas no compartieran los recursos durante la medición. La plataforma (Tabla 1) ofrece las interfaces para observar energía y telemetría en CPU y GPU mediante RAPL y

NVML. Las actuaciones de frecuencia de CPU se aplicaron a los seis núcleos físicos delegados del primer zócalo y a sus hilos hermanos, porque ambos comparten el dominio físico de frecuencia. Los demás núcleos y el *uncore*, cuya frecuencia gestiona el propio procesador, no se actuaron. En la Fase 1 la energía de CPU corresponde al paquete y a la DRAM del zócalo donde se fija la carga, que incluye esos seis núcleos, dos núcleos físicos no delegados y el *uncore*. En la Fase 4 la energía de CPU suma los paquetes de ambos zócalos, sin la DRAM (sección 3.5.2). La sección 4.1 presenta la verificación de esta condición.

Tabla 1
Plataforma Experimental

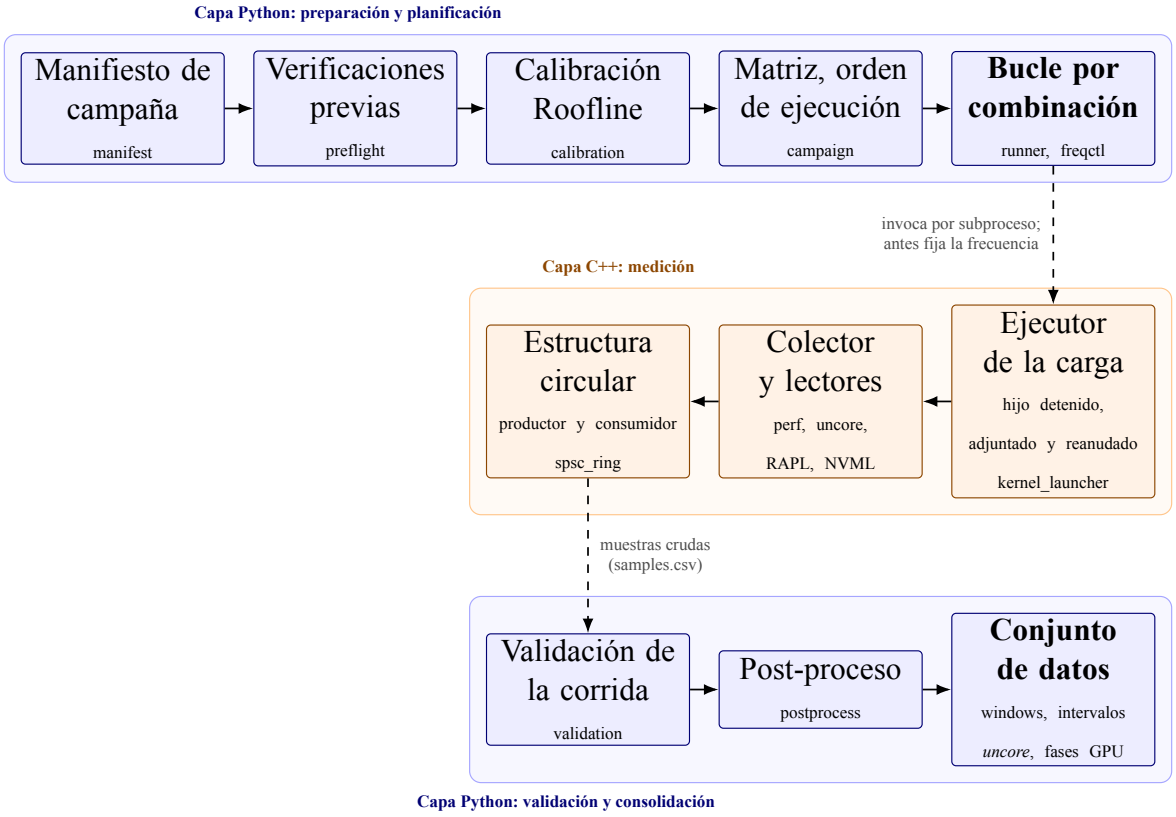
Componente	Descripción
CPU	2 × Intel Xeon Gold 5315Y (Ice Lake-SP), 8 núcleos físicos por zócalo, 2 hilos por núcleo, 2 nodos NUMA
Frecuencia	`intel_pstate`, 0.8 a 3.2 GHz (3.6 GHz con *turbo* activo), dominio por núcleo físico
Energía de CPU	Intel RAPL, dominios de paquete y de DRAM por zócalo. Fase 1: paquete y DRAM del zócalo de la carga; Fase 4: paquetes de ambos zócalos, sin DRAM (Ecuación 12)
GPU	NVIDIA A100 PCIe 40 GB, telemetría por NVML
Ejecución	Nodo en asignación exclusiva (Slurm)

El instrumento propio tiene dos capas (Figura 4). La capa C++ adquiere los datos. Inicia la carga como un proceso detenido, configura los contadores de hardware sobre su árbol de procesos e hilos y la reanuda cuando la medición está preparada. Un hilo productor realiza las lecturas periódicas y un hilo consumidor las persiste, y ambos se comunican mediante una estructura circular de un productor y un consumidor. La capa Python coordina el protocolo experimental. Interpreta el manifiesto de campaña, verifica la plataforma, realiza la calibración, planifica las corridas, aplica los estados de frecuencia, valida cada ejecución y construye el conjunto de datos.

Una **ventana de muestreo** es el intervalo entre dos lecturas consecutivas y contiene la diferencia entre los valores acumulados en ambos extremos. En CPU se usaron hasta nueve eventos simultáneos. Ese número sale del presupuesto de contadores físicos del nodo y se verificó experimentalmente para evitar la multiplexación y la extrapolación de las mediciones. El tráfico a DRAM

Figura 4

Recorrido de una Campaña a Través de las Capas de Preparación, Medición y Consolidación del Instrumento



se obtiene de los contadores *uncore* del controlador de memoria, con intervalos de al menos 10 ms. Ese piso es una decisión de diseño: la documentación de `perf stat` advierte que la sobrecarga de la medición por intervalos puede ser alta cuando estos son pequeños (Linux perf project, 2026), y en este trabajo no se midió el costo a escalas menores. Como el dominio *uncore* es compartido por todo el zócalo (Intel Corporation, 2021), cada campaña requiere asignación exclusiva del nodo para atribuir el tráfico a la carga medida. En GPU, el recolector solicita lecturas NVML cada 5 ms, pero los sensores se actualizan aproximadamente cada 100-120 ms. Por eso estas muestras se usan como telemetría agregada y no como ventanas GPU con resolución de 5 ms. Las lecturas que repiten un mismo valor del sensor se conservan sin deduplicar; como cada actualización se repite un número parecido de veces, la mediana y la desviación estándar por corrida se aproximan a las calculadas sobre las actualizaciones distintas, y en ningún caso las lecturas se tratan como obser-

vaciones independientes. El agente, en cambio, muestrea cada 50 ms (sección 3.4.1), de modo que la cadencia difiere entre entrenamiento y operación. Además, se registran la frecuencia observada de cada núcleo (`scaling_cur_freq`), la energía RAPL de paquete y de DRAM y, en GPU, la utilización de cómputo y de memoria, la potencia, el reloj del multiprocesador, la temperatura y la energía acumulada por NVML.

3.1.2 Modelo Roofline y Etiquetado

La etiqueta de cada observación (*compute-bound* o *memory-bound*) resulta de comparar su intensidad operacional I con el punto de inflexión del modelo Roofline (Ecuación 3). Los techos se calibran al inicio de cada campaña, con la misma asignación de recursos de las corridas y según la precisión aritmética. En los barridos de frecuencia, la calibración se ajusta además al nivel correspondiente. Para ello se usan microbenchmarks de ancho de banda y de cómputo, contrastados con una referencia declarada. La expectativa cualitativa de cada carga se conserva solo como referencia y no define la etiqueta.

La obtención de I difiere entre CPU y GPU. En CPU, los FLOPs de cada ventana suman los subeventos escalares y vectoriales de 128, 256 y 512 bits de `FP_ARITH_INST_RETIRED` (Intel Corporation, 2026b), ponderados por el número de elementos de doble precisión que procesa cada instrucción (Ecuación 6). Según la documentación de Intel, estos eventos cuentan dos veces las instrucciones FMA, porque realizan dos operaciones por elemento (Intel Corporation, 2026b), de modo que la ecuación cuenta cada FMA como dos operaciones sin un término adicional. Los bytes salen de los comandos CAS del controlador de memoria, por lo que I se calcula por intervalo durante la ejecución. En GPU, NVML no entrega FLOPs ni bytes por ventana o lanzamiento. Por eso la intensidad operacional se caracteriza fuera de línea con Nsight Compute (NVIDIA, 2026b), y el resultado se asocia a cada corrida. Para un kernel, esa intensidad suma sobre sus L lanzamientos perfilados las instrucciones de suma, multiplicación y FMA (esta última con dos FLOPs) y las divide por los bytes B_ℓ movidos a la memoria del dispositivo (Ecuación 7). Se considera estable para una configuración fija después de verificar su convergencia. Además, se distingue la precisión aritmética realmente ejercitada y se excluyen las cargas de aritmética entera o de precisión mixta,

que no permiten una intensidad operacional consistente.

$$\mathrm{FLOPs}_i^{\mathrm{HW}} = \Delta N_{\mathrm{esc},i} + 2\Delta N_{128,i} + 4\Delta N_{256,i} + 8\Delta N_{512,i} \tag{6}$$

$$I^{\mathrm{GPU}} = \frac{\sum_{\ell=1}^{L}(N_\ell^{\mathrm{add}} + N_\ell^{\mathrm{mul}} + 2N_\ell^{\mathrm{fma}})}{\sum_{\ell=1}^{L} B_\ell} \tag{7}$$

3.2 Fase 1: Recolección y Caracterización

3.2.1 Catálogo, Matriz Experimental y Control de Sesgos

Antes de cada campaña se ejecuta una batería de verificaciones automáticas sobre el entorno, el catálogo, la instrumentación y los permisos de actuación, y las verificaciones bloqueantes detienen la ejecución si falla una condición necesaria. El catálogo definitivo parte de familias elegibles sometidas a un cribado con tres niveles de frecuencia. El cribado descarta cargas no verificables, comprueba la calidad de las señales y cubre regímenes de ejecución contrastantes. El catálogo reúne cargas de suites públicas de HPC y cargas de elaboración propia, entre ellas familias con versión CPU y GPU del mismo algoritmo, y microbenchmarks de calibración que no forman parte del entrenamiento. La diversidad se controla por familia algorítmica, con un mínimo de 5 familias por clase y dispositivo. Las familias, suites de origen, referencias, variantes y sumas de verificación están en el catálogo versionado del repositorio y en el material complementario (sección 3.2.2).

La matriz experimental combina las cargas del catálogo, los estados de frecuencia de la Tabla 2 y tres repeticiones independientes por combinación. Los niveles se verifican contra los relojes soportados por el nodo, y en CPU el turbo permanece desactivado. Cada ventana conserva un estado de validez que distingue las observaciones aceptadas de las afectadas por la transición o por un nivel no aplicable.

Tabla 2
Rejilla Fina de Frecuencia de la Campaña Definitiva, con Pasos Uniformes de 300 MHz en CPU y de 150 MHz en GPU

Dispositivo	Niveles de reloj en MHz
CPU	`REF`, `F0` (3200), `F1` (2900), `F2` (2600), `F3` (2300), `F4` (2000), `F5` (1700), `F6` (1400), `F7` (1100) y `F8` (800).
GPU	`REF`, `F0` (1410), `F1` (1260), `F2` (1110), `F3` (960), `F4` (810), `F5` (660), `F6` (510), `F7` (360) y `F8` (210).

La caracterización previa del cambio de frecuencia mostró tiempos de asentamiento del reloj de 10-11 ms en CPU y 70-80 ms en GPU. Con base en ellos, cada nivel se verifica dentro de una tolerancia de $\pm 5\,\%$ y se aplica un margen de guarda de 0.15 s al inicio y al final de la región medible.

Para reducir sesgos, el orden de ejecución se aleatoriza con una semilla registrada en el manifiesto (Montgomery, 2017). Se incluyen repeticiones de una carga de referencia para detectar variaciones del entorno, y durante el barrido de un dispositivo el otro conserva su configuración de referencia.

3.2.2 Calidad de los Datos, Granularidad y Reproducibilidad

Ninguna observación se elimina sin conservar su estado de calidad. Las muestras afectadas por calentamiento, contadores incompletos, energía inválida u otras condiciones de adquisición se identifican en el postproceso, y así el filtrado queda separado de la adquisición. El transitorio inicial de cada carga se estima sobre el IPC en CPU y sobre la utilización en GPU, y se excluye solo durante el análisis. Se toma el primer instante en que el coeficiente de variación no supera 5 % en dos ventanas móviles consecutivas y se añade un margen de seguridad de 20 %. En GPU se exige además que la media de esas ventanas alcance 5 % de utilización. Si no aparece un estado estable, se aplica una detección recursiva de puntos de cambio y se elige el primer segmento con media de al menos 80 % de la meseta de mayor carga. Cada corrida también se valida por integridad de adquisición y por el criterio de éxito de la carga.

La unidad de entrenamiento depende del dispositivo. En CPU es un **intervalo del contro-**

lador de memoria, sobre el que las variables se recalculan a partir de los deltas agregados. En GPU es una **corrida completa**, representada por agregados de las muestras válidas posteriores al calentamiento. Las lecturas de una misma corrida no se consideran observaciones independientes.

Cada campaña se define mediante un manifiesto versionado que registra las cargas, los estados de frecuencia, las repeticiones, el período de muestreo, la asignación de núcleos, la semilla y las referencias de calibración. Junto con los datos se almacenan el perfil de capacidades del nodo, la calibración y los metadatos de cada corrida. Las ejecuciones interrumpidas se reanudan sin repetir corridas ya aceptadas, y una huella del protocolo impide combinar resultados obtenidos con configuraciones diferentes. El código, los datos procesados que respaldan cada tabla y figura, y el material complementario de este documento (catálogo completo, rejillas numéricas, protocolos por escenario y figuras adicionales) están en el repositorio público del proyecto, https://github.com/AdrianCCRS/hyperion, en el archivo `docs/libro/README.md`.

3.3 Fase 2: Modelos de Clasificación y Política de Frecuencia

3.3.1 Formulación, Variables y Validación

Se formula una clasificación binaria supervisada por dispositivo, cuya salida es la probabilidad de la clase *memory-bound*. En CPU se utilizan seis variables. Son el IPC, el MPKI, la tasa de fallos de caché, la fracción de ciclos detenidos por memoria, las instrucciones por segundo y la frecuencia observada. Esta última se conserva porque el cambio de frecuencia modifica el techo de cómputo y puede desplazar una carga respecto del punto de inflexión. Antes del entrenamiento se revisa la redundancia entre variables con la correlación y el factor de inflación de la varianza (VIF). Se toman como criterios de alerta los pares con correlación absoluta superior a 0.85 y un VIF mayor a 10, y el conjunto final queda registrado en un contrato versionado por dispositivo. Cuando dos columnas describen la misma señal se conserva la medición físicamente más directa. La intensidad operacional, los FLOPs, los bytes, el punto de inflexión y la etiqueta no pueden ser entradas, y el entrenamiento se aborta si alguna aparece. En GPU las variables candidatas son cinco agregados de las muestras NVML posteriores al calentamiento de cada corrida: las medianas de la utilización de GPU, de la utilización de memoria, de la potencia y del reloj SM, y la desviación

estándar de la utilización de memoria. La auditoría de invarianza (sección 3.3.3) redujo el modelo desplegado a tres: la mediana de la utilización de GPU, la mediana de la utilización de memoria y su desviación estándar. En el agente, esas tres variables se calculan sobre las muestras de la ventana sostenida de 3 s de cada episodio de actividad, tomadas cada 50 ms. El modelo se entrena, por tanto, con la corrida como unidad y se aplica sobre ventanas de 3 s. Para contrastar esa diferencia, las variables que el agente calcula en cada decisión de la Fase 4 se comparan con las de las corridas de entrenamiento del mismo kernel (sección 4.4.1).

El agente no calcula la intensidad operacional durante la ejecución, aunque el instrumento de la Fase 1 la obtiene por intervalo, porque su medición depende de condiciones que un agente en operación no puede asumir. En CPU, los bytes de DRAM salen de los contadores *uncore* del controlador de memoria, cuyo ámbito es el zócalo completo (Intel Corporation, 2021). Atribuirlos a una carga exige la asignación exclusiva del nodo, y leerlos requiere privilegios elevados (`CAP_PERFMON`). Los contadores de núcleo, en cambio, pueden asociarse a la carga que se ejecuta. En GPU, NVML no entrega FLOPs ni bytes, y la intensidad operacional solo se obtiene con Nsight Compute, que repite cada *kernel* perfilado para reunir sus métricas (NVIDIA, 2026b). Durante la caracterización, perfilar el GEMM de RAJAPerf alargó su ejecución unas 60 veces. Por eso la intensidad operacional se usa solo para construir la etiqueta, en campañas con el nodo reservado o fuera de línea, y el clasificador la aproxima con señales que el agente puede leer a bajo costo durante la ejecución: contadores de núcleo en CPU y NVML en GPU.

La validación usa *leave-one-family-out* (LOFO). En cada pliegue se retira por completo una familia, que no interviene en el ajuste, en la selección de representación ni en la elección del umbral (Figura 5) (Kaufman et al., 2012; Roberts et al., 2017). Para que las familias de mayor volumen no dominen el entrenamiento, cada celda de familia y clase se limita a 1000 intervalos con una semilla fija, y las observaciones se ponderan de forma inversa al tamaño de su celda, con media uno.

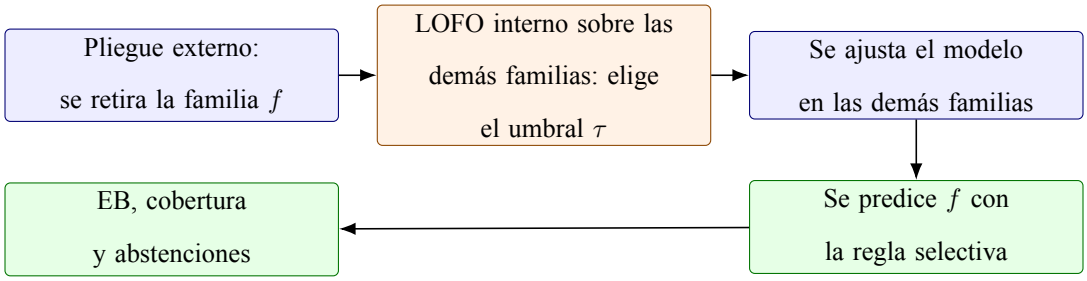
$$w_i = \frac{1/n_{fc}}{(1/N)\sum_{k=1}^{N} 1/n_{f_k c_k}} \tag{8}$$

La métrica principal es la exactitud media por celda de familia y clase (Brodersen et al., 2010),

$$\mathrm{EB} = \frac{1}{|\mathcal{C}|} \sum_{(f,c)\in\mathcal{C}} \frac{a_{fc}}{n_{fc}}, \tag{9}$$

donde $\mathcal{C}$ es el conjunto de celdas observadas (cada combinación de familia f y clase c), $|\mathcal{C}|$ es el número de celdas, a_{fc} son los aciertos de cada celda y n_{fc} sus observaciones totales. En el resto del documento se la llama **exactitud balanceada por celda**. No es la exactitud balanceada estándar, que promedia la sensibilidad de las dos clases, sino un promedio macro sobre las celdas de familia y clase, de modo que cada familia pesa lo mismo con independencia de su número de intervalos. Como complemento se reporta la puntuación F_1 macro agrupada. La incertidumbre se estima con cinco semillas de submuestreo y con *bootstrap* de familias con 2000 réplicas (Efron y Tibshirani, 1993).

Figura 5
Validación Anidada LOFO



Nota. La familia retirada no interviene en ninguna decisión de ajuste.

3.3.2 Decisión Selectiva, Calibración y Selección de Modelos

Sea p_i la probabilidad de *memory-bound* y $q_i = \max(p_i, 1 - p_i)$ la confianza. El modelo emite una clase solo cuando $q_i \geq \tau$, y en caso contrario se abstiene con la salida *revisar*:

$$\widehat{y}_i = \begin{cases} \textit{memory-bound}, & p_i \geq 0.5 \ \wedge \ q_i \geq \tau, \\ \textit{compute-bound}, & p_i < 0.5 \ \wedge \ q_i \geq \tau, \\ \textit{revisar}, & q_i < \tau. \end{cases} \tag{10}$$

En la evaluación, el umbral τ se selecciona con un LOFO interno sobre una rejilla de valores. Se maximiza la exactitud media por celda sobre las predicciones aceptadas y se exige una cobertura mínima de 0.60. Para el agente se congela un único umbral por dispositivo, elegido sobre la curva de exactitud y cobertura del modelo final, y se verifica que la exactitud obtenida con el umbral elegido por pliegue sea equivalente. La cobertura es la proporción de observaciones que reciben una clase y cuantifica el costo de la abstención. La calibración de la confianza se evalúa con el error de calibración esperado (ECE), calculado sobre diez intervalos de igual amplitud:

$$\mathrm{ECE} = \sum_{b=1}^{10} \frac{n_b}{N} \left| \mathrm{conf}_b - \mathrm{acc}_b \right|, \tag{11}$$

donde N es el número total de observaciones evaluadas, n_b el número de observaciones del rango b, conf_b la confianza media de ese rango y acc_b su proporción de aciertos.

Bajo el mismo protocolo se comparan una regla mayoritaria, un árbol de profundidad uno, regresión logística, *random forest*, *extra trees* y XGBoost, y en GPU se añade un árbol de profundidad seis (Demšar, 2006). Los hiperparámetros principales se fijan antes del ajuste y, como análisis complementario, se evalúa una búsqueda anidada con Optuna dentro de cada pliegue externo (Cawley y Talbot, 2010). También se aplican diagnósticos de separabilidad y de capacidad del modelo, entre ellos un clasificador de vecinos más cercanos sin ajuste (sección 4.1.3.5). La selección final considera el desempeño LOFO, la calibración, la cobertura y la estabilidad de las entradas frente a la acción del agente. Los modelos y umbrales resultantes se presentan en el capítulo de resultados.

3.3.3 Invarianza de las Entradas frente a la Acción del Agente

Además del desempeño predictivo, se evalúa si la propia actuación del agente puede modificar indirectamente las entradas y generar retroalimentación. En GPU se aplican dos pruebas sobre los datos disponibles. La prueba contrafactual sustituye el valor de una entrada potencialmente afectada y mide los cambios de predicción. La prueba de exclusión repite la validación LOFO sin esa variable. Con ambas se decide si la entrada se conserva, se excluye o requiere una restricción operativa.

El modelo se evalúa de nuevo con el contrato de entradas resultante. En CPU se aplica la misma auditoría a la frecuencia observada, con sustituciones coherentes con el rango de actuación previsto. La latencia de inferencia se reporta por separado de la latencia completa entre detección y actuación. Los resultados de estas auditorías están en las secciones 4.2.2 y 4.1.3.1.1.

3.3.4 Derivación de la Política de Frecuencia

La política de frecuencia se deriva de forma independiente del clasificador, a partir de las mediciones de la Fase 1. Para cada clase, el EDP de cada nivel se compara con la base sobre el mismo kernel, y la ganancia agregada se estima mediante *bootstrap* de kernels con 4000 réplicas (intervalo de confianza de 95 %). Un nivel solo entra en la política si su mejora está respaldada estadísticamente y supera el efecto mínimo relevante de 1 %. Las diferencias pareadas por kernel se contrastan con la prueba de rangos con signo de Wilcoxon (Hollander et al., 2014), o con la prueba t pareada cuando hay ocho o más pares y Shapiro-Wilk no rechaza la normalidad. En GPU la decisión usa la familia como unidad y la prueba de Wilcoxon unilateral, que contrasta si el nivel mejora frente a la base. Como el nivel se elige entre varios después de inspeccionarlos y estas pruebas no se corrigen por multiplicidad, la ganancia del nivel elegido sobre todas las familias se trata como una estimación interna y descriptiva. La evidencia de generalización es la validación dejando una familia fuera, en la que el nivel se elige sin la familia evaluada y se mide sobre ella.

La regla se valida dejando una familia fuera y se congela antes de evaluarse sobre familias externas. En GPU, la clase de un kernel es la mayoritaria entre sus corridas, su margen es la fracción de corridas que pertenecen a esa clase, y se marca ambigua si el margen es menor que 0.75. Los kernels ambiguos cercanos al punto de inflexión se conservan en el análisis y los tamaños de un mismo algoritmo se agrupan por familia.

3.4 Fase 3: Agente de Control DVFS

La Fase 3 recibe congelados de la Fase 2 el modelo de cada dispositivo, su umbral de abstención y la tabla de política. El modelo y el umbral proceden de la validación del clasificador, y la política, de las comparaciones de EDP entre niveles. Así el desempeño predictivo no se asocia directamente con la conveniencia de una acción DVFS. Los modelos se exportan a ONNX y se

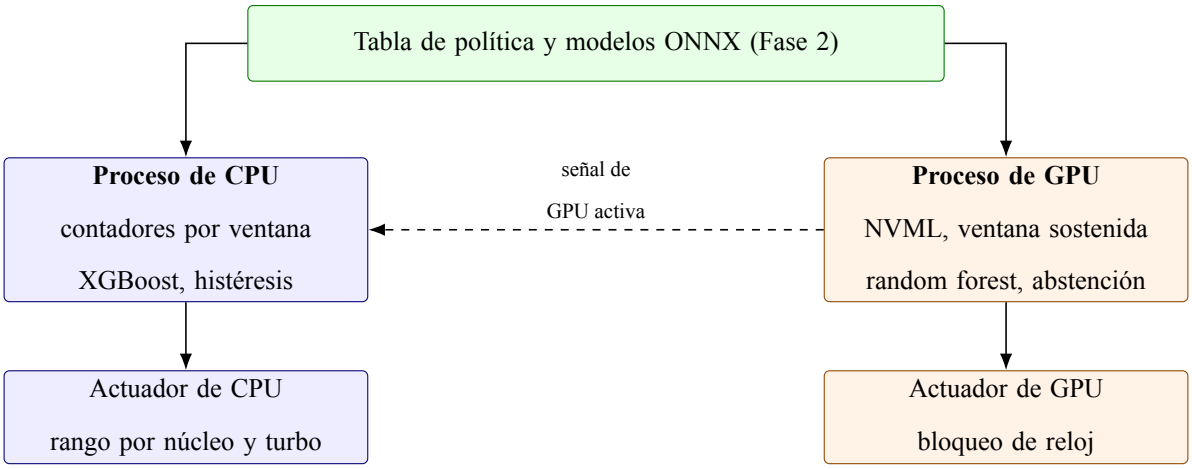
ejecutan desde C++ con ONNX Runtime, tras verificar que sus probabilidades y etiquetas coinciden con las de los modelos originales. Con estos artefactos, la Fase 3 implementa la detección, la decisión, la actuación, la coordinación y la restauración que luego se evalúan en la Fase 4.

3.4.1 Arquitectura, Detección de Fase y Parámetros

El agente se implementa con dos procesos C++ independientes, uno para CPU y otro para GPU, coordinados mediante una señal binaria (Figura 6). Así los bucles de control pueden tener escalas temporales distintas y la actuación de cada dispositivo queda aislada. Ambos procesos cargan al inicio los modelos, umbrales y tablas de política congelados.

Figura 6

Arquitectura del Agente: Dos Procesos, una Tabla de Política Compartida y una Señal de Coordinación



Cada proceso admite los tres brazos de la Tabla 3. El brazo *base* es la referencia contra la que se compara todo, y el brazo *sombra* mide la sobrecarga del agente sin su efecto sobre el hardware.

Tabla 3
Brazos del Agente

Brazo	Agente	Comportamiento
Base	No corre	En CPU, turbo desactivado y gobernador *performance* (3.2 GHz fijos) o *powersave* (EPP *default*, 0.8 a 3.2 GHz); en GPU, gestión automática del controlador
Sombra	Muestrea, clasifica y decide	Registra cada decisión y nunca escribe sobre el hardware
Activo	Muestrea, clasifica, decide y escribe	Aplica la frecuencia decidida y restaura el estado al terminar

El agente de GPU identifica episodios sostenidos de actividad con la utilización que reporta NVML, acumula 3 s de observación y decide una vez por episodio. Ante una abstención libera el reloj. En CPU la decisión se actualiza por ventana y una actuación requiere 50 propuestas consecutivas del mismo estado. La Tabla 4 resume los parámetros operativos.

Tabla 4
Parámetros del Agente

Proceso	Parámetro	Valor y criterio
GPU	Umbral de actividad	5 % de utilización
GPU	Ventana sostenida	3 s; una decisión por episodio de actividad
GPU	Cadencia de muestreo	50 ms
GPU	Umbral de abstención	0.90; seleccionado en la Fase 2 (sección 4.2.2)
GPU	Permanencia mínima	3.7 s; definida a partir de la latencia de actuación medida en la Fase 3 (sección 4.3.3)
CPU	Umbral de abstención	0.85; seleccionado en la Fase 2 (sección 4.1.3.4)
CPU	Histéresis	50 ventanas consecutivas
CPU	Piso con GPU activa	3.2 GHz; verificado en la Fase 3 (sección 4.3.4)

3.4.2 Política, Actuación y Coordinación

La Tabla 5 resume las acciones congeladas para la evaluación. La política operativa mantiene la actuación de GPU derivada en la Fase 2. En CPU se añaden dos variantes experimentales para estudiar el efecto de cerrar el lazo aunque la política derivada no recomiende reducir la frecuencia. Estas variantes no modifican el clasificador ni la política original y se usan solo como brazos de evaluación.

Tabla 5
Acciones del Agente

Dispositivo	Clase	Acción
CPU	*compute-bound*	Fija F0 (3.2 GHz, turbo desactivado); experimental
CPU	*memory-bound*	Baja a F1 (2.9 GHz) con confianza mínima de 0.85; experimental
GPU	*memory-bound*	Fija el reloj en 1260 MHz (F1)
GPU	*compute-bound*	No actúa: libera el reloj a la gestión automática del controlador
GPU	*revisar* (abstención)	No actúa: libera el reloj a la gestión automática del controlador

Nota. Las filas de GPU provienen de la política derivada en la Fase 2; las filas de CPU son variantes experimentales evaluadas en la Fase 4.

El actuador de CPU reutiliza las restricciones de dominio de la Fase 1, aplica el cambio sobre los núcleos delegados y sus hilos SMT y verifica el estado después de cada actuación. Antes de modificar el dispositivo guarda su configuración inicial, y la restaura al finalizar o ante una terminación anómala. El actuador de GPU sigue la misma disciplina al fijar o liberar el reloj.

La coordinación entre ambos procesos es una regla de protección independiente de la política. Durante un episodio de actividad GPU, el proceso CPU mantiene un piso de 3.2 GHz sin esperar la histéresis, para que una reducción de frecuencia en una CPU que espera al acelerador no prolongue la ejecución GPU. Esta regla no modifica la clasificación ni la tabla de política, y su efecto sobre el EDP se evalúa por separado en la Fase 4.

3.4.3 Aplicaciones Compuestas y Verificación Previa

Las aplicaciones compuestas registran, con reloj monotónico, fronteras de fase conocidas por construcción para contrastar las decisiones del agente (Tabla 6). La aplicación A usa familias presentes en el entrenamiento y la B, familias que no lo estuvieron. Una tercera aplicación alterna trabajo CPU y GPU. Cada fase mantiene al menos 3 s de actividad y se separa de la siguiente con 1 s de reposo para observar el cambio.

Tabla 6
Aplicaciones Compuestas con Fronteras y Clases Conocidas por Construcción

Alcance	Aplicación	Fases (clase)
CPU	A (vistos)	`dgemm_n2048` (*compute*), `npb_cg` (*memory*)
CPU	B (inéditos)	`cpu_xsbench_omp` (*memory*), `cpu_rsbench_omp` (*compute*)
GPU	A (vistos)	`gpu_cutlass_simt_dgemm_n4096` (*compute*), `gpu_rajaperf_stream_triad` (*memory*)
GPU	B (inéditos)	BabelStream con `--numtimes 1000` (*memory*), `rodinia_lavamd` con `-boxes1d 100` (*compute*)

Antes de la evaluación final se definieron pruebas específicas para la latencia de actuación, la sobrecarga, la restauración ante terminación anómala y el cumplimiento de la regla de coordinación. Sus resultados están en la sección 4.3.

3.5 Fase 4: Validación Experimental

3.5.1 Diseño de la Matriz y Escenarios

La matriz inicial evalúa el agente en tres dimensiones, que son el alcance experimental (CPU, GPU o ejecución conjunta), la procedencia de las familias (vistas durante el entrenamiento o inéditas) y el brazo de ejecución (Tabla 7). El orden de las celdas se aleatoriza con semilla fija y cada aplicación ejecuta dos ciclos completos por repetición. El brazo *sombra* estima la sobrecarga del agente sin actuación sobre el hardware. Los brazos *activo-F0* y *activo sin piso* son variantes experimentales que aíslan el efecto de la actuación y el de la regla de coordinación. Los alcances con tres repeticiones se analizan de forma descriptiva, con valores individuales y razones de medianas.

Tabla 7
Matriz Experimental de la Fase 4

Alcance	Brazos	Vistos	Inéditos	Rep.
CPU	base, sombra, activo, activo-F0	A	B	3
GPU	base, sombra, activo	A	B	3
Conjunto	base, sombra, activo con piso, activo sin piso	A	B	5

Además de la matriz inicial se ejecutan los protocolos complementarios de la Tabla 8, cuyas

condiciones completas están en el material complementario (sección 3.2.2). Cada protocolo se fija antes de sus mediciones y se reporta sin importar el resultado. Los escenarios D y CloverLeaf incorporan aplicaciones completas externas al catálogo inicial (Mallinson et al., 2013; Plimpton, 1995). El escenario E es dirigido, pues selecciona familias que en la Fase 2 mostraron una reducción de EDP de al menos 10 % en F1, y añade una variante con familias no vistas. Por eso sirve para estudiar el agente en condiciones favorables identificadas de antemano y no como estimación global de desempeño. En C y E se repite además la medición de la base como control de repetibilidad. El escenario E-C inserta entre dos fases de memoria de E-A el kernel de cómputo con mayor pérdida de EDP de GPU en F1 durante la Fase 2 que no está limitado por potencia, `rodinia_heartwall`, criterio fijado antes de medir. Es favorable a la clasificación, como E-A lo es a la política, y sirve para contrastar la conmutación por fase con un reloj fijo.

3.5.2 Medición y Análisis

En cada celda experimental se registran la duración, la energía de CPU mediante RAPL y la energía de GPU mediante NVML. La métrica de la Fase 4 es el EDP CPU+GPU,

$$\mathrm{EDP}_{\mathrm{CPU+GPU}} = \left(E_0^{\mathrm{pkg}} + E_1^{\mathrm{pkg}} + E_{\mathrm{GPU}}\right) T \quad [\mathrm{J}\cdot\mathrm{s}], \tag{12}$$

donde E_0^{pkg} y E_1^{pkg} son las energías RAPL de paquete de los dos zócalos, E_{GPU} la energía acumulada que reporta NVML para la A100 y T la duración de la aplicación. La frontera del sistema son esos tres dominios: quedan fuera la DRAM, la placa base, el almacenamiento, la ventilación y las pérdidas de la fuente, por lo que la métrica no es la energía total del nodo. Las aplicaciones compuestas y LAMMPS se fijan en los seis núcleos delegados del primer zócalo, pero CloverLeaf y los procesos del agente no se fijan y pueden ejecutarse en cualquier núcleo; por eso se suman ambos paquetes, para que la energía de la aplicación y la sobrecarga del agente queden completas en la medición. Como el segundo zócalo permanece casi en reposo en todos los brazos, su energía no cambia las diferencias entre brazos, pero acerca las razones a 1, de modo que los ahorros relativos reportados son conservadores. Se conserva además $E_{\mathrm{GPU}}T$ para los análisis específicos del acele-

Tabla 8

Mapa de los Escenarios de la Fase 4 y Carácter de su Análisis

Escenario	Contraste principal	Carácter del análisis
Matriz inicial	Dispositivo, familias vistas e inéditas y brazo	Descriptivo; Mann-Whitney en el alcance conjunto
C	Repetición independiente de CPU y conjunto	Réplica descriptiva
E	GPU con fases largas de memoria, familias vistas (E-A) e inéditas (E-B)	Exploratorio (tres repeticiones)
Repetición de base	Segunda medición de la base como control de repetibilidad	Control
D	LAMMPS con cuatro entradas y fases naturales	Descriptivo
F	Diagnóstico de la solicitud de F1 fijo	Diagnóstico
E-A por bloques	Cinco bloques con base, sombra, agente y F1 fijo	Prueba de signos declarada; IC95 y contraste con F1 fijo *post hoc*
E-B ampliado	Ocho bloques de E-B y diagnóstico de variabilidad de *myocyte*, con telemetría de CPU y GPU	Exploratorio y diagnóstico
E-B con *powersave*	Ocho bloques de E-B con los núcleos delegados en el gobernador *powersave* (EPP *default*), contraste de gobernador nativo	Exploratorio
Serie *powersave*	Matriz inicial, E, E-A por bloques, D y CloverLeaf repetidos con los núcleos delegados en *powersave*, cada brazo frente a la base del mismo gobernador	Réplica con el mismo análisis de cada escenario
CloverLeaf	Cinco bloques con base, sombra y agente en ejecución continua	Sin prueba declarada; prueba de signos e IC95 *post hoc*
E-C	Fase de cómputo sensible al reloj entre las fases de memoria de E-A; seis bloques con base, agente y F1 fijo, bajo *performance* y *powersave*	Prueba de signos agente/F1 fijo declarada como principal; IC95 y contrastes con la base declarados como secundarios; prueba de mecanismo

Nota. Todos los protocolos se fijaron antes de medir; la última columna distingue la prueba declarada en el protocolo de los análisis añadidos después de medir (*post hoc*) y de los escenarios exploratorios o diagnósticos. El protocolo completo está en el repositorio.

rador. La ventana energética cubre toda la ejecución de la aplicación con los procesos del agente ya activos, y por eso incluye su sobrecarga. El brazo sombra mide la sobrecarga total de observación, construcción de variables e inferencia, y el activo le suma la actuación. Esa sobrecarga total acota por arriba el costo de la inferencia, cuya latencia se mide aparte (sección 4.3.3), pero no la separa de las demás componentes. Al terminar cada celda se verifica que los procesos correspondientes hayan terminado y se restauran los estados de frecuencia modificados.

En la matriz inicial y en el escenario E, las aplicaciones compuestas registran fronteras

de fase conocidas por construcción y las decisiones del agente, con instante, clase, confianza y actuación. LAMMPS se ejecuta con fases naturales sin etiquetar y CloverLeaf como aplicación continua. En ambos casos se conservan la telemetría y la actuación observada, pero no se asigna una verdad de fase independiente.

Para cada combinación experimental se calculan las medianas de duración, energía y EDP, y su razón respecto del brazo *base*. En la matriz conjunta, con observaciones no apareadas, el EDP se contrasta con una prueba bilateral de Mann-Whitney (Hollander et al., 2014). Los experimentos por bloques, con protocolo declarado antes de medir, se analizan con una prueba de signos exacta bilateral sobre el EDP (Hollander et al., 2014). Con cinco bloques, el menor valor p que puede dar esta prueba es $2 \cdot (1/2)^5 = 0.0625$, cuando todos los bloques favorecen al mismo brazo, por lo que solo informa la dirección del efecto: un resultado de cinco de cinco bloques se lee como una dirección consistente en ese escenario y no como una confirmación estadística. El protocolo de E-A preveía una prueba exacta de permutaciones entre brazos, cuyo mínimo con cinco corridas por brazo es 0.008, y el de CloverLeaf no fijaba una prueba. Como la aleatorización se hizo por bloques, esa prueba admitiría asignaciones que el diseño no podía producir, por lo que al analizar se sustituyó por la prueba de signos, más conservadora. E-C, diseñado después, usó seis bloques para que esta prueba pudiera llegar a 0.031.

La magnitud del efecto se estima con la media geométrica de las razones brazo/base por bloque y su intervalo de confianza de 95 %, obtenido con una prueba t pareada sobre el logaritmo de esas razones (Montgomery, 2017). En E-A y CloverLeaf este análisis se añadió después de medir, y con cinco bloques no es posible verificar el supuesto de normalidad del logaritmo; en E-C se declaró antes de medir como análisis secundario. Por eso este intervalo describe la magnitud y no sustituye a la prueba declarada en el protocolo. Las razones individuales de cada bloque en duración, energía y EDP se muestran en las figuras, para que el efecto pueda leerse sin el supuesto paramétrico.

El brazo F1 fijo bloquea el reloj de GPU en 1260 MHz durante toda la aplicación y registra la utilización y el reloj SM cada 250 ms. El protocolo declarado exigía que toda muestra con utili-

zación de al menos 10 % quedara entre 1230 y 1290 MHz. Como el límite de potencia de la A100 lleva el reloj por debajo de 1260 MHz en la fase de cómputo de E-A también sin bloqueo, después de medir se restringió esa verificación a las fases de memoria, que son las que la política actúa. En E-A el contraste con F1 fijo es, por tanto, un análisis posterior a la medición, igual que el intervalo de confianza por bloques. En E-C el contraste se declaró antes de medir y se aplicó el criterio de reloj del protocolo sin restricción.

La clasificación se evalúa solo en las aplicaciones compuestas. Cada decisión se atribuye a la fase que contiene su instante, y se reportan aciertos, errores, abstenciones y cobertura por alcance, tipo de kernel y dispositivo. En CPU la puntuación es por ventanas, según la granularidad de decisión del agente. La comparación entre A y B muestra la variación del desempeño sobre familias no vistas dentro de este protocolo. En LAMMPS y CloverLeaf se reportan las decisiones y actuaciones observadas, pero no una exactitud de clasificación, porque no hay una referencia independiente de fase con esa granularidad.

La Tabla 9 resume las principales amenazas a la validez y las medidas adoptadas para controlarlas.

Tabla 9

Amenazas a la Validez de Mayor Peso y Medidas de Control

Amenaza	Medida de control
Interferencia de procesos ajenos en RAPL y en la GPU	Asignación exclusiva del nodo; ninguna carga adicional durante una medición
Frecuencia efectiva distinta de la pedida	Turbo apagado, rango por núcleo y hermanos *SMT*, verificación por ventana
Fuga de la etiqueta hacia las variables	Lista de columnas prohibidas comprobada por código
Pseudo-replicación entre tamaños y repeticiones	La familia como unidad de validación (LOFO) y de comparación de la política
Retroalimentación del agente sobre el modelo	Auditoría de invarianza de las variables frente a la acción del agente
Sesgo temporal o de orden	Orden aleatorizado con semilla registrada y repeticiones independientes
Muestra intencional en un solo nodo	Se declara como limitación; no se generaliza a otras plataformas

Nota. La lista completa está en el material complementario del repositorio.

4. Resultados

Este capítulo presenta los resultados de las campañas de CPU y GPU, del agente de control y de su evaluación frente a la referencia en aplicaciones compuestas. Las tablas y figuras de sensibilidad, desglose y trazabilidad están en el material complementario del repositorio (sección 3.2.2); las referencias del tipo figura 7.1 del material complementario remiten a él. El cuerpo principal reúne la evidencia necesaria para sustentar cada conclusión.

4.1 Resultados de CPU

La campaña CPU cubrió 49 configuraciones de kernel en el nivel nativo y nueve niveles fijos de frecuencia, con tres repeticiones por combinación. Sus 1470 corridas quedaron aceptadas. Con los calentamientos calibrados, los transitorios fueron inferiores a 2 % de la duración de corrida en la mayoría de los kernels y de 4.8 % como máximo.

De 1 424 531 intervalos procesados, 1 309 755 (91.9 %) cumplieron simultáneamente los criterios de telemetría y de frecuencia definidos en Metodología. Se rechazaron 71 658 (5.0 %) por frecuencia no utilizable y 43 118 (3.0 %) por proceder de una ventana no válida. Los rechazados se conservaron para auditoría, pero no recibieron una etiqueta de entrenamiento. Entre los elegibles, 526 279 (40.2 %) son *compute-bound* y 783 476 (59.8 %) *memory-bound*.

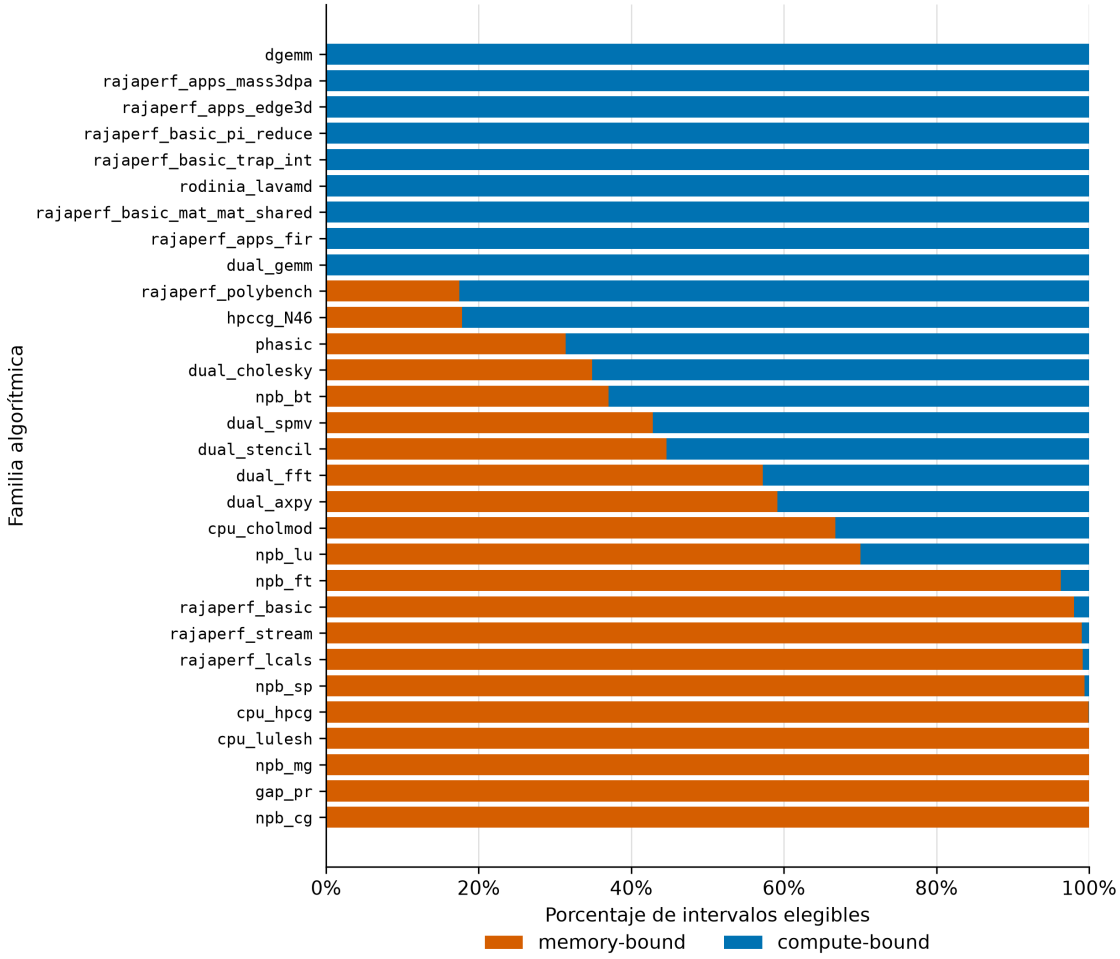
Toda corrida se ejecutó con turbo desactivado y con la frecuencia solicitada verificada en cada ventana, con una tolerancia de ± 5 %. Una ventana que no cumple ese criterio no recibe etiqueta. Tras cada cambio de nivel se descartan unos 10 ms, porque un procesador delegado tardaba entre 10 y 11 ms en alcanzar el límite solicitado. También se descartan 0.15 s al inicio y al final de cada corrida, porque en los kernels de una sola región paralela larga un hilo que termina antes queda ocioso y diluye el promedio de frecuencia. Al restringir solo los procesadores lógicos delegados, la razón de rendimiento entre el nivel más alto y el más bajo se mantuvo en 0.995. Al restringir también sus hermanos de *SMT*, esa razón subió a 3.78, cercana a la razón de cuatro esperada entre esos niveles.

4.1.1 Composición de Fase por Kernel y por Frecuencia

Las familias de cargas, sus variantes y su suite de procedencia se enumeran en el material complementario del repositorio (sección 3.2.1). La Figura 7 resume la composición por familia, y el desglose por configuración de kernel queda también en el material complementario. El conjunto cubre ambos regímenes y conserva configuraciones próximas al *ridge*.

Figura 7

Porcentaje de Intervalos Elegibles Clasificados Compute-bound y Memory-bound por Familia Algorítmica



Nota. Las variantes de tamaño se agregan dentro de cada familia.

La composición intermedia no se concentra en un solo algoritmo. *npb_bt*, *npb_lu* (Bailey et al., 1991), *cpu_cholmod* (Chen et al., 2008), *hpccg_cpu_N46* (Heroux et al., 2009), las variantes

phasic y los tamaños dirigidos de FFT y Cholesky aportan intervalos de ambos regímenes. Estas configuraciones evitan que el conjunto quede limitado a ejemplos alejados del punto de inflexión.

Tres kernels de patrón de acceso irregular de Rodinia (Che et al., 2009), *kmeans*, *srad* y *particlefilter*, se midieron por separado y resultaron casi puramente *memory-bound*. No aportaban ejemplos *compute-bound* ni mixtos, y quedaron fuera del conjunto de entrenamiento porque solo añadían redundancia *memory-bound*.

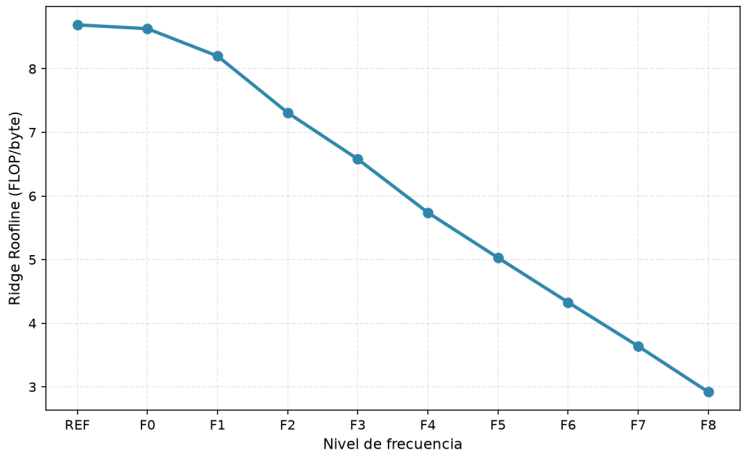
La proporción global de intervalos *compute-bound* aumentó desde 29.0 % en el nivel nativo hasta 46.4 % en F8, el nivel fijo más bajo. La figura 7.1 del material complementario presenta esta variación. Es consistente con un *ridge* que baja junto con el techo de cómputo, por lo que una parte de los intervalos cercanos a la frontera cambia de etiqueta. La cifra agrega todas las configuraciones, y las cargas no aportan el mismo número de intervalos.

4.1.2 Techos Roofline y Ridge

Los techos del modelo Roofline se calibraron de forma empírica sobre la propia plataforma (sección 2.1.2). El punto de inflexión desciende con el reloj, de 8.69 FLOP/byte en la referencia nativa a 2.92 FLOP/byte en F8 (Figura 8). Baja porque el techo de cómputo baja, mientras el ancho de banda de memoria se mantiene.

Figura 8

Ridge Roofline según el Nivel de Frecuencia CPU



Nota. El punto de inflexión desciende de 8.69 FLOP/byte en referencia nativa a 2.92 FLOP/byte en F8.

4.1.2.1 Validación de los Techos. Para verificar la calibración con un mecanismo independiente se usó Intel Advisor, que contabiliza FLOPs y bytes mediante instrumentación binaria y no con contadores de PMU (Intel Corporation, 2025). El ancho de banda calibrado queda entre 8 y 13 % por debajo del de Advisor (58.4 a 58.8 GB/s frente a 64.0 a 66.8 GB/s). El techo de cómputo (417.8 a 555 GFLOP/s) es consistente con la referencia de Advisor, que midió 577.7 GFLOP/s sobre los mismos seis núcleos. El punto de inflexión del nivel nativo queda entre 7.1 y 9.5 FLOP/byte, compatible con los 8.69 FLOP/byte usados en la campaña.

4.1.3 Clasificador de Fase en CPU

Los resultados del clasificador siguen el protocolo LOFO y la exactitud balanceada por celda definidos en la sección 3.3. El modelo CPU se ajustó con 42 458 intervalos de 30 familias (21 350 *compute-bound* y 21 108 *memory-bound*), con un tope de 1000 intervalos por celda de familia y clase (figura 7.2 del material complementario). La sensibilidad a ese tope mantuvo la exactitud entre 0.720 y 0.739 al variarlo de 100 a 30 000, sin tendencia creciente. La unidad de generalización sigue siendo la familia, no cada intervalo.

4.1.3.1 Variables de Entrada. El candidato CPU usa las seis variables definidas en la sección 3.3. La auditoría de correlación (sección 3.3.1) se calculó sobre las doce columnas candidatas (figura 7.3 del material complementario). El par más alto, de 0.89 (Pearson), une MPKI con las referencias de caché por millar de instrucciones, dos normalizaciones de la misma señal de fallos de caché, y de ese par se retuvo MPKI. Sobre las seis variables finales ningún par supera 0.85, y la mayor correlación absoluta es 0.74 (Spearman), negativa, entre IPC y la fracción de ciclos detenidos por memoria. El factor de inflación de la varianza máximo es 5.5 con Pearson y 7.3 con rangos, ambos de IPC, por debajo del umbral de alerta de 10. No fue necesario retirar ninguna variable adicional.

Ampliar el vector de seis a doce entradas redundantes cambió la exactitud balanceada por celda de 0.728 a 0.724 (IC95 de la diferencia, -0.026 a $+0.011$). Retirar la frecuencia observada la baja a 0.720, y usarla como única entrada da 0.592. Por eso se conservaron seis variables, incluida la frecuencia. La redundancia perjudicó más a la regresión logística, y a XGBoost no le ofreció una

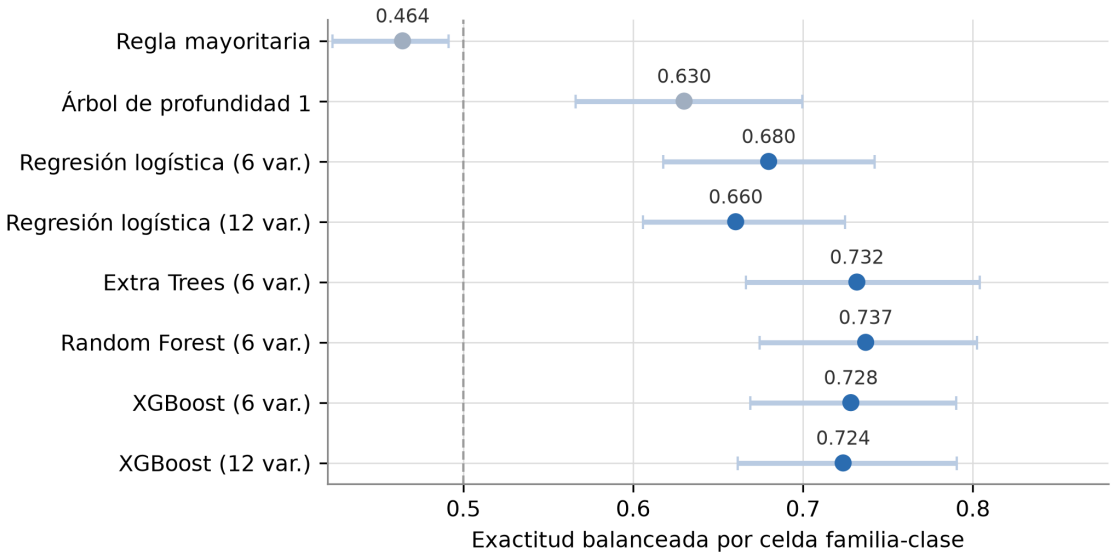
mejora medible de inferencia ni de precisión.

4.1.3.1.1 Invarianza frente a la Acción del Agente. La frecuencia observada es una de las seis variables del clasificador de CPU y el agente la modifica. Por eso se le aplicó la prueba contrafactual de la sección 3.3.3, que fuerza la frecuencia a un valor fijo y mide cuántas predicciones cambian. Sobre el modelo final y sus 1.31 millones de intervalos de 49 kernels, forzar la frecuencia a 3.2, 2.9, 2.0 y 0.8 GHz cambia, respectivamente, el 10.6, 10.1, 4.5 y 8.8 % de las predicciones. Existe entonces una dependencia moderada. Pero el cambio que el agente ejecuta es acotado, de F0 (3.2 GHz) a F1 (2.9 GHz), y sobre las 179 774 filas observadas cerca de 3.2 GHz solo cambia el 0.42 % de las predicciones. Con ese espacio de acción, la retroalimentación por esta variable es despreciable, y el modelo de CPU se despliega sin modificaciones.

4.1.3.2 Selección de Modelo e Hiperparámetros. La Figura 9 compara los candidatos con hiperparámetros fijos, cinco semillas y el mismo protocolo LOFO. XGBoost, *extra trees* y *random forest* quedan dentro del mismo intervalo de incertidumbre, y la regresión logística queda por debajo en exactitud balanceada por celda (0.680, frente a 0.728 de XGBoost), aunque su F_1 macro agrupada es la más alta. La puntuación F_1 macro agrupada de los modelos de seis variables es de 0.205 para la regla mayoritaria, 0.713 para el árbol de profundidad 1, 0.765 para la regresión logística, 0.717 para *extra trees*, 0.763 para *random forest* y 0.760 para XGBoost. Se seleccionó XGBoost por su menor latencia entre los ensambles de árboles. La latencia de una fila y un hilo, medida en el nodo con el proceso fijado a cuatro núcleos, es de 243 µs de mediana (404 µs en el percentil 99) para XGBoost, frente a 5.3 ms (5.5 ms) de los dos bosques, unas 22 veces mayor. El árbol de profundidad 1 y la regresión logística toman 105 µs (109) y 450 µs (475).

Figura 9

Comparación de Modelos del Clasificador CPU



Nota. Exactitud balanceada por celda de familia y clase de cada modelo, con intervalo de confianza del 95 % obtenido por remuestreo de las 30 familias, con hiperparámetros fijos. Las variantes de doce variables se detallan en el material complementario del repositorio. Los intervalos se solapan entre todos los modelos con capacidad suficiente.

4.1.3.2.1 Búsqueda de Hiperparámetros. La búsqueda anidada con Optuna (Akiba et al., 2019) no produjo una mejora operativamente útil. Para XGBoost, la diferencia pareada por familia frente a la configuración fija es de $+0.019$ (IC95 de -0.007 a $+0.050$, que incluye el cero; 14 de 30 familias mejoran y 8 empeoran). Para la regresión logística es distinguible de cero, pero es irrelevante en magnitud ($+0.002$). El puntaje de validación interna de Optuna en cada pliegue varía poco (de 0.734 a 0.761 de exactitud balanceada por celda), mientras la exactitud balanceada por celda sobre la familia externa retenida va de 0.487 a 1.000. Por eso el criterio interno no distingue configuraciones buenas de malas para una familia nueva. Se conservó la configuración fija de XGBoost, la más simple de reproducir. Junto con la comparación de modelos y de vectores de entrada, esto indica que el desempeño no está limitado por la capacidad del modelo ni por sus hiperparámetros.

La configuración congelada es la de la Tabla 10.

Tabla 10
Configuración Final del Clasificador CPU

Elemento	Valor
Modelo	XGBoost, 100 árboles, profundidad máx. 6, objetivo `binary:logistic`
Entradas	6 variables PMU (sección 3.3)
Ponderación	Peso total igual por celda de familia y clase (Ecuación 8), con `scale_pos_weight` $= 1$
Salida	Probabilidad de *memory-bound*; confianza ≥ 0.85 (*revisar* por debajo)
Semilla	20260918

4.1.3.3 Evaluación del Modelo Final. La métrica principal, la exactitud balanceada por celda, fue 0.728 sin abstención (IC95 de 0.669 a 0.790). La variación entre las 30 familias limita la distinción entre configuraciones cercanas. Esa cifra depende de la unidad de separación, porque mezclar intervalos al azar da 0.874 y dejar fuera un kernel da 0.733. Los valores de las demás métricas de CPU están en la Tabla 11 (la Tabla 12 de la sección 4.2.2 trae las de GPU, para comparar). Sobre los 1 309 755 intervalos elegibles, el modelo alcanzó una F1 macro agrupada de 0.760 y una exactitud global de 0.769, con sensibilidad de 0.714 para *compute-bound* y de 0.806 para *memory-bound*. La Figura 10 muestra las matrices de confusión.

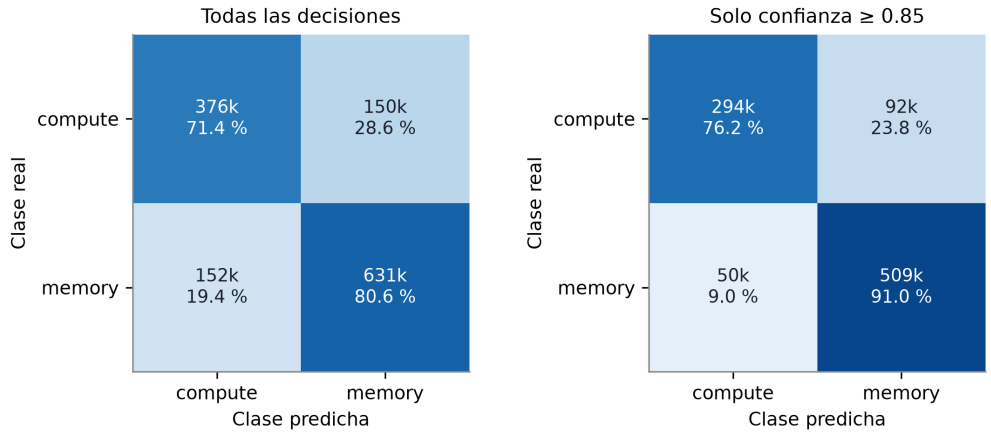
Tabla 11
Métricas del Clasificador CPU sin Abstención

Métrica	Valor
Exactitud	0.769
Sensibilidad *compute-bound*	0.714
Sensibilidad *memory-bound*	0.806
F1 macro agrupada	0.760
AUC-ROC agrupada	0.842
Coeficiente de Matthews	0.520
Exactitud balanceada por celda	**0.728**

Nota. Sin abstención (Chicco y Jurman, 2020; Sokolova y Lapalme, 2009), media de cinco semillas de muestreo. El detalle con desviaciones e intervalos está en el material complementario.

Figura 10

Matrices de Confusión del Modelo Final

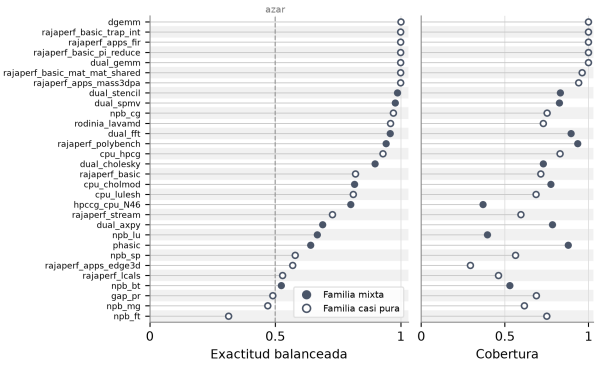


Nota. A la izquierda, cuando se obliga a emitir una clase en todos los intervalos; a la derecha, solo sobre los intervalos con confianza mayor o igual a 0.85.

El resultado no es uniforme entre familias (Figura 11). Catorce familias superan 0.9 de exactitud y cinco quedan por debajo de 0.55, con *npb_ft* en 0.314. Las mejores tienen un comportamiento extremo y estable. Las peores tienen un perfil de contadores que coincide con el de familias de la clase contraria.

Figura 11

Resultado del Clasificador CPU por Familia



Nota. Exactitud balanceada por celda (izquierda) y cobertura de la decisión automática (derecha) para cada familia retirada, ordenadas por exactitud. El marcador lleno distingue las familias mixtas, entre 10 y 90 % de intervalos *memory-bound*, y el hueco las casi puras; la línea discontinua marca el azar.

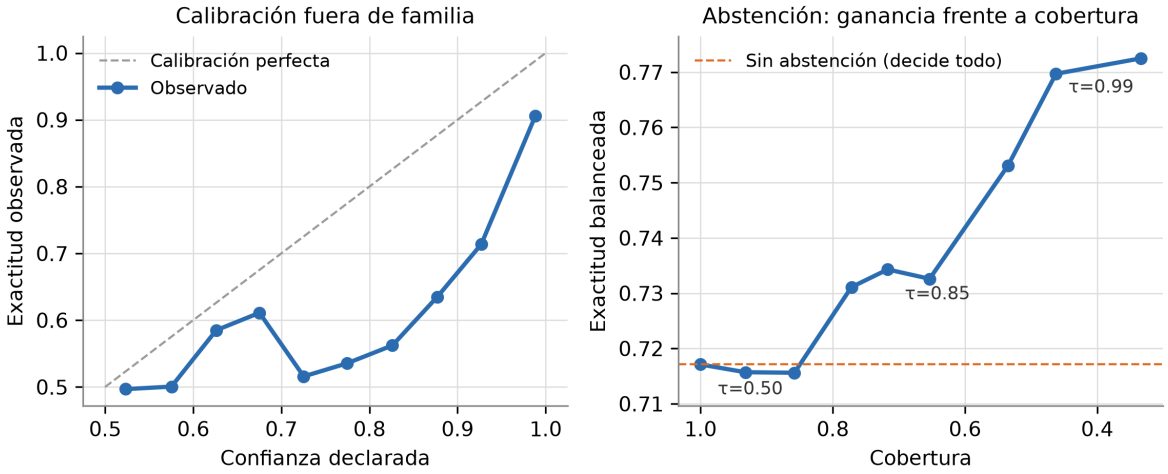
Al bajar el reloj, la exactitud balanceada por celda y la cobertura de la decisión automática caen juntas (figura 7.4 del material complementario). Es consistente con el desplazamiento del *ridge*. En los niveles bajos, más intervalos pasan a la clase de cómputo, quedan cerca de la frontera y el modelo se abstiene más. La exactitud global repunta en F8 (0.757, frente a 0.710 en F7), pero es un efecto de composición. F8 aporta más intervalos que los demás niveles (256 531, frente a 85 447 a 186 309), en su mayoría *compute-bound*, y la exactitud balanceada por celda no rebota.

4.1.3.4 Decisión Selectiva y Costo de Inferencia. Se congeló el umbral de abstención CPU en 0.85 (Figura 12, derecha). Elevarlo a 0.90 aumentaba la exactitud de selección solo 0.003 y reducía la cobertura en 6.5 puntos. Es una decisión de diseño posterior al resultado, que la Metodología declara (sección 3.3). Elegir el umbral por pliegue, dentro de cada familia retirada, da prácticamente lo mismo (0.734 sobre lo decidido), por lo que congelar un único valor no infla el resultado. Con ese umbral, sobre el modelo final, el agente decide el 72.2 % de los intervalos (75.1 % promediando por familia y 29.3 % en la familia peor cubierta), y el resto recibe *revisar*. Sobre lo que decide, la exactitud balanceada por celda sube solo 0.010, de 0.728 a 0.738, a cambio de dejar sin decisión el 27.8 % de los intervalos. Las métricas agrupadas suben más (la F1 macro agrupada de 0.760 a 0.842, y la sensibilidad de las clases *compute-bound* y *memory-bound* de 0.714 y 0.806 a 0.762 y 0.910; Figura 10), pero esa subida refleja sobre todo qué intervalos quedan fuera y no una mejora equivalente por familia, que es la métrica principal. Abstenerse al azar en una proporción equivalente de intervalos no mejora nada (0.716 y 0.752). Por eso la ganancia viene de que la confianza identifica casos difíciles, y no de decidir menos (Figura 12, derecha).

El límite de la abstención es que la confianza está mal calibrada fuera de la familia de entrenamiento (Figura 12, izquierda). El modelo declara una confianza media de 0.890 y acierta con 0.764 de exactitud (error de calibración esperado de 0.126, con una sola semilla, frente a 0.769 de exactitud global promediando cinco). Además, hay familias que fallan con confianza alta, a las que el umbral no protege. Entre los intervalos con confianza mayor o igual a 0.9, *npb_ft* acierta el 32 % y *npb_mg* el 35 %.

Figura 12

Calibración y Compromiso de la Abstención en CPU



Nota. A la izquierda, confianza declarada frente a exactitud observada sobre familias no vistas; la curva por debajo de la diagonal indica sobreconfianza. A la derecha, exactitud balanceada por celda sobre lo decidido en función de la cobertura al variar el umbral τ (el eje de cobertura está invertido: a la derecha, el umbral es más estricto); la línea discontinua naranja es la exactitud sin abstención; ambas curvas usan una sola semilla de muestreo, por lo que sus valores difieren ligeramente de los promedios de cinco semillas que se dan en el texto.

Ante la salida *revisar*, el proceso de CPU del agente conserva la última frecuencia aplicada; el de GPU, en cambio, libera el reloj (sección 3.4.1). Las latencias de inferencia de la selección de modelo incluyen la sobrecarga del intérprete de Python, y son una cota superior para la implementación integrada en el agente (sección 4.3.3).

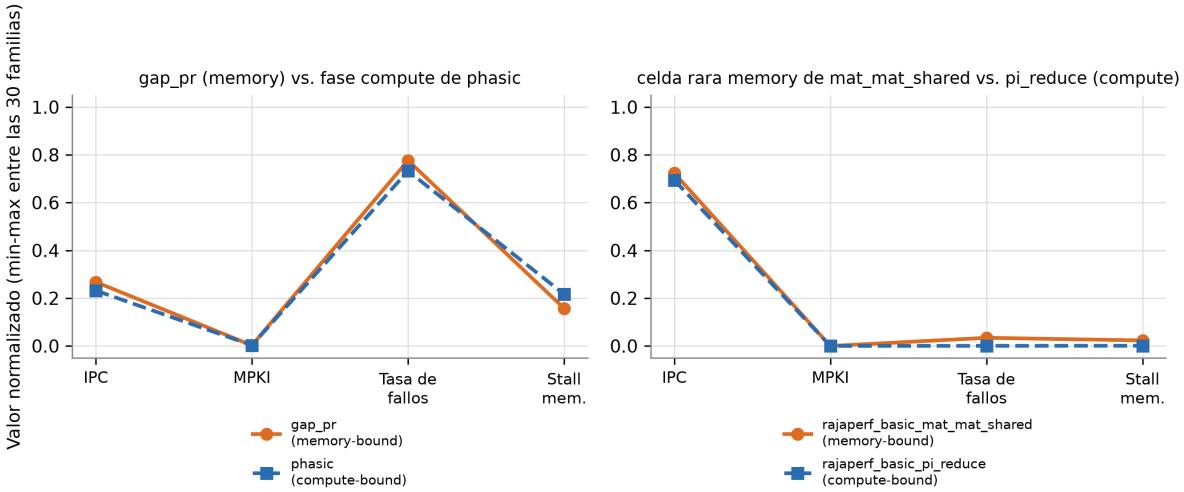
4.1.3.5 Límite de la Exactitud del Clasificador de CPU. Cuatro análisis indican que el límite proviene de la información contenida en las seis variables de entrada y no del modelo.

El primero es que un clasificador sin ajuste, de 25 vecinos más cercanos sobre cuatro tasas estandarizadas, alcanza 0.684 con el mismo protocolo, y 0.699 si recibe además la frecuencia. Queda a menos de 0.045 del modelo final. El segundo es la existencia de perfiles gemelos entre familias de clase opuesta (Figura 13). El caso más simétrico enfrenta a *gap_pr* (Beamer et al., 2015), *memory-bound*, con la fase *compute-bound* de *phasic*. La primera tiene MPKI de 0.82, IPC

de 1.11 y tasa de fallos de 0.76, y la segunda tiene MPKI de 0.94, IPC de 0.97 y tasa de fallos de 0.72. El par más cercano en distancia estandarizada es otro. La celda rara *memory-bound* de *rajaperf_basic_mat_mat_shared* (0.1 % de sus intervalos; la familia es *compute-bound*) queda a la mitad de distancia de la celda *compute-bound* típica de *rajaperf_basic_pi_reduce*, de la misma suite. Ningún modelo que reciba solo esas señales puede separar estos pares, y las tres familias involucradas están entre las de peor resultado.

Figura 13

Perfiles de Celdas Confusables entre Clases Opuestas



Nota. Cuatro variables de entrada, normalizadas al rango de las 30 familias, para los dos pares de celdas más confusables entre clases opuestas.

El tercero es la posición respecto del *ridge*. Los intervalos a menos de un factor dos del punto de inflexión son el 12.5 % del total y se clasifican con 0.521 de exactitud, frente a 0.799 en el resto. Sin embargo, concentran solo el 25.3 % de los errores, por lo que tres cuartas partes ocurren lejos de la frontera, donde la etiqueta no es ambigua. El error tampoco es simétrico. En la franja *compute-bound* inmediata al *ridge* la exactitud es de 0.33, por debajo del azar, y en la franja simétrica *memory-bound* es de 0.75. Es consistente con el déficit de sensibilidad de compute-bound (sección 4.1.3.3) y con los perfiles gemelos (figura 7.5 del material complementario).

El cuarto es la curva de aprendizaje por número de familias (figura 7.6 del material comple-

mentario). Al ajustar con subconjuntos de 3 a 23 familias, el resultado sobre la familia retirada sube de 0.615 a 0.718, unos 0.037 por cada duplicación del catálogo. Es consistente con que la diversidad algorítmica gobierne el resultado, aunque el diseño no separa el número de familias del tamaño de la muestra. Dentro del alcance de este trabajo, la mejora disponible por esa vía es pequeña.

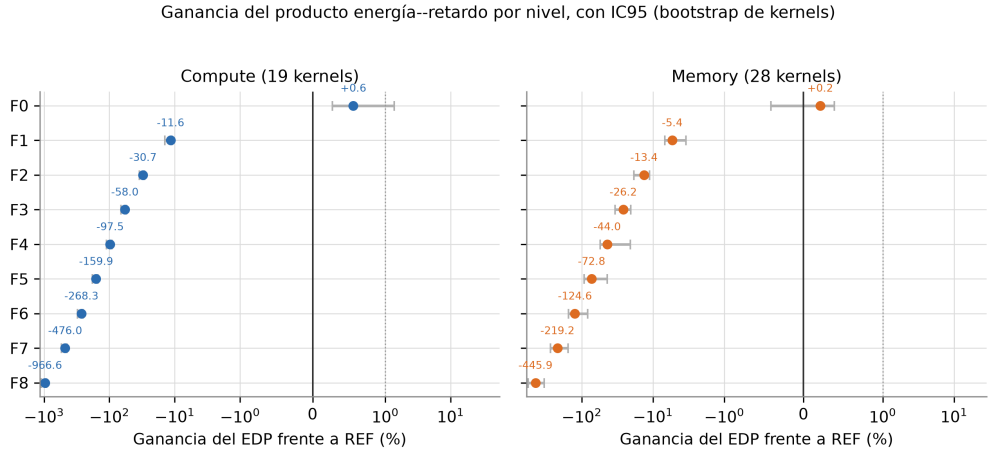
Como referencia de límite superior, el mismo modelo con la intensidad operacional medida como entrada adicional alcanza 0.9989, frente a 0.728 sin ella. No es una mejora del modelo, porque la intensidad operacional es la cantidad con la que se define la etiqueta, y por eso queda fuera del vector de entrada. Solo muestra que la brecha entre 0.728 y 1 es información que las seis variables del agente no transportan. Los contadores de FLOPs y de bytes de DRAM sí la contienen, pero no se pueden usar durante la operación del agente (sección 3.3.1).

4.1.4 Política de Frecuencia por Clase en CPU

Se comparó el EDP de cada nivel fijo frente a la base conforme al procedimiento de la sección 3.3. La política resultante es *no actuar* en ambas clases (Figura 14). La Figura 15 separa la energía y el tiempo que producen ese resultado. En cómputo, F0 mejora 0.56 % frente a la base, menos que el efecto mínimo relevante de 1 %, y todos los niveles inferiores empeoran el EDP. En memoria tampoco hay ganancia agregada a menor reloj. F1 empeora 5.4 % (IC95 $[-6.8; -3.4]$) y los niveles inferiores empeoran más. Algunos kernels de STREAM (McCalpin, 1995) presentan ahorros puntuales entre 4 y 11 % en su mejor nivel, pero el signo y la magnitud varían entre niveles y repeticiones, y no justifican una regla común.

Figura 14

Ganancia del EDP frente a la Base por Nivel y Clase (Cómputo: 19 Kernels; Memoria: 28), con IC95 por Bootstrap de Kernels; Eje Simétrico-logarítmico

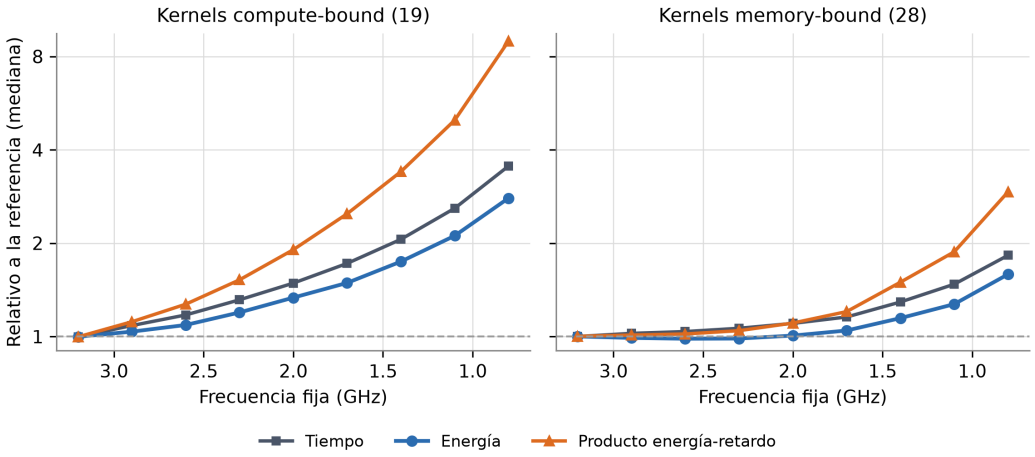


Nota. Valores p y límites numéricos en el material complementario.

La energía sube al bajar la frecuencia porque el tiempo crece más de lo que cae la potencia (Figura 15). En memoria se mantiene cerca de la referencia hasta unos 2 GHz. La razón es que unos 85 W, el 84 % de la potencia de paquete y DRAM del zócalo de la carga a 3.2 GHz, se consumen sea cual sea la frecuencia de los núcleos delegados (Figura 16). Ese piso incluye dos núcleos físicos no delegados, el *uncore* y la DRAM, que la actuación no controla, por lo que el resultado se limita a actuar sobre la frecuencia de los núcleos delegados; actuar sobre la frecuencia del *uncore* no se evaluó. Reducir la frecuencia a 0.8 GHz ahorra solo 15 % de potencia mientras alarga la ejecución. Incluso un oráculo que eligiera el mejor nivel medido por kernel ganaría 2.2 % en promedio, y 22 de 47 kernels quedarían por debajo del efecto mínimo de 1 % (los 49 kernels del clasificador, sin los tres *phasic* de duración fija, y con `ptrchase`, que solo se midió para esta política). Esta cota es optimista, porque el mejor nivel también captura ruido de medición.

Figura 15

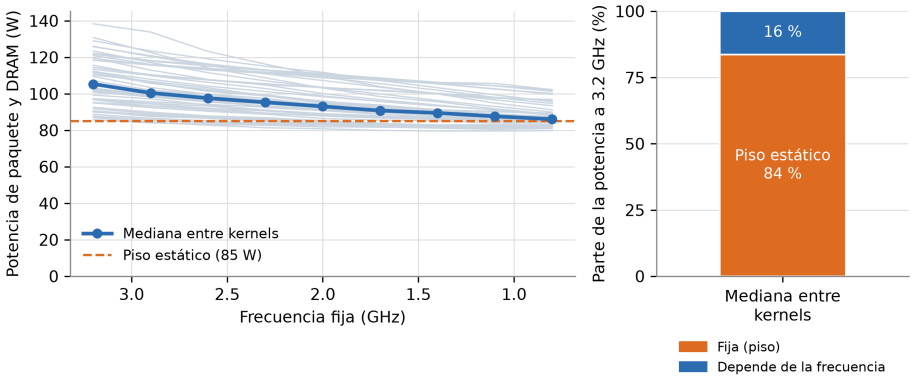
Energía, Tiempo y EDP por Nivel de Frecuencia en CPU



Nota. Mediana entre kernels de la energía (paquete y DRAM), el tiempo y el EDP de cada nivel fijo de CPU, relativos a la referencia nativa, para las dos clases. El eje vertical es logarítmico.

Figura 16

Potencia de Paquete y DRAM frente a la Frecuencia



Nota. A la izquierda, potencia media de paquete y DRAM de cada kernel (líneas grises) y su mediana en cada nivel fijo, con el piso estático ajustado. A la derecha, parte de la potencia a 3.2 GHz que corresponde al piso: la barra muestra la mediana entre kernels (84 %), y según el kernel va de 62 a 94 %.

4.1.4.1 Tamaño de Efecto Detectable. Un resultado de *no actuar* solo es informativo si se conoce el efecto que el experimento podía detectar. La variabilidad tiene dos fuentes. El error

de medición es pequeño, porque repetir el mismo kernel en el mismo nivel da un producto energía-retardo con una desviación estándar del logaritmo de 1.05 % en mediana. La diferencia entre familias es mayor. La dispersión entre las 19 familias *memory-bound* es de 5.35 % en F1 y 9.70 % en F2, entre cinco y diez veces la primera. Por eso repetir más las mediciones no reduciría la incertidumbre de la ganancia promedio, y solo lo haría un catálogo con más familias. Un modelo de potencia $P(f) = P_0 + c\, f^k$, ajustado a cada kernel con un error mediano de 0.34 W y un piso casi idéntico entre kernels (80 a 100 W), muestra que bajar la frecuencia mejora el EDP únicamente si la pendiente β de $\log T$ frente a $\log f$ es menor que un valor crítico. Ese valor tiene mediana 0.14 y en ningún kernel supera 0.33, mientras que los kernels de cómputo denso tienen un β cercano a 1. El oráculo que elige el mejor nivel de cada kernel gana como máximo 12.5 %. Decidir actuar exige entonces identificar la región de β bajo, y la clase Roofline por sí sola no la determina.

4.2 Resultados de GPU

4.2.1 Etiquetado, Conjunto de Datos y Clasificador de Fase en GPU

El actuador de GPU sostuvo el bloqueo del reloj de *streaming multiprocessors* bajo carga. Con objetivos de 600 y 1200 MHz se registraron 305 y 174 lecturas activas coincidentes con el valor solicitado, y la restauración se verificó al finalizar las campañas dedicadas (la premisa física de la política se comprueba en la sección 4.3.2). En el nivel de referencia, los puntos de inflexión medidos fueron 14.97 FLOP/byte para FP32 y 6.90 FLOP/byte para FP64. La caracterización solicitó ambas precisiones a la vez y mostró que la precisión declarada no siempre es la ejecutada. *rodinia_backprop* y *rodinia_myocyte*, declarados FP32, ejecutaron 82.3 y 71.3 % de sus FLOPs en FP64, y se conservaron con precisión FP64 porque su clasificación *memory-bound* no cambia con ninguno de los dos techos. *rodinia_hotspot*, con 30.4 % en FP64 y una intensidad de 7.21 FLOP/byte en el borde del *ridge* FP32 (7.28 a 7.98), se retiró del catálogo por no admitir un único techo.

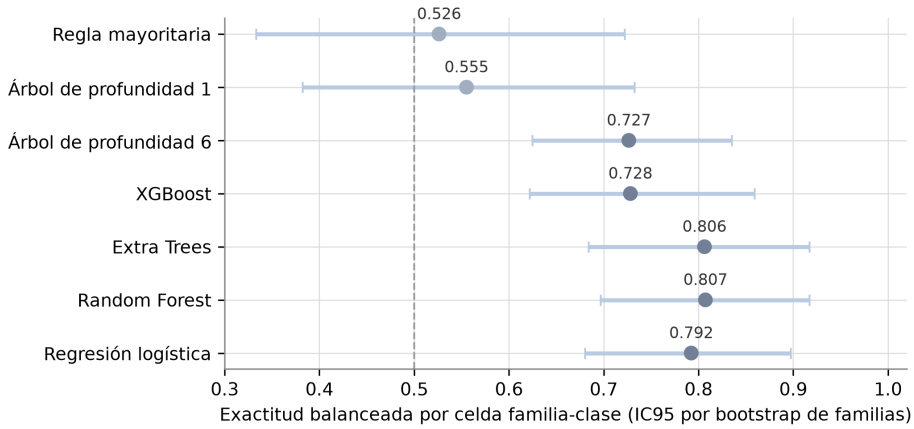
El conjunto de entrenamiento del clasificador GPU reúne 483 corridas aceptadas de 19 kernels y 16 familias algorítmicas, en los diez niveles de frecuencia (referencia nativa y F0 a F8). Proceden de la campaña GPU final y de tres campañas adicionales de *kmeans*, miniBUDE y *conv2d*. Cada corrida tiene una etiqueta Roofline obtenida fuera de línea y señales NVML agregadas por

corrida. Los tres *kernels* CUDA de RAJAPerf (Beckingsale et al., 2019) (GEMM, *stencil* de calor y Jacobi 2D) forman familias separadas, porque sus perfiles Roofline difieren entre sí. La composición es de 171 corridas *compute-bound* y 312 *memory-bound*, y 13 de las 16 familias son homogéneas respecto de su etiqueta.

Los candidatos reciben cinco señales agregadas por corrida, que son las medianas de utilización GPU, utilización de memoria, potencia y reloj SM, más la desviación estándar de la utilización de memoria. Esta última evita que la mediana diluya cargas de lanzamientos cortos y repetidos como *kmeans*, de intensidad operacional alta pero con una firma NVML agregada similar a la de una carga poco ocupada. Las cinco señales no presentan colinealidad relevante. El factor de inflación de varianza (VIF) fue de 1.89 para la utilización GPU, 1.76 para la de memoria, 2.26 para la potencia, 1.37 para el reloj SM y 1.32 para la desviación estándar de la utilización de memoria, todos muy por debajo del umbral de alerta de 10 del contrato de variables.

Con el protocolo de validación definido en la sección 3.3, ninguno de los siete modelos candidatos domina de forma concluyente en exactitud balanceada por celda (Figura 17). *Random forest* (0.807), *extra trees* (0.806) y la regresión logística (0.792) resultaron estadísticamente indistinguibles entre sí por remuestreo pareado de familias, con probabilidades de superioridad, la fracción de réplicas *bootstrap* en que un modelo supera al otro, entre 0.51 y 0.59. *Random forest* sí superó con evidencia razonable a XGBoost, con probabilidad de superioridad de 0.984. La búsqueda anidada de hiperparámetros empeoró cuatro de los cinco modelos ajustables, por ejemplo *random forest* pasó de 0.807 a 0.778, y solo mejoró el árbol de profundidad 6, de 0.727 a 0.783. Por eso se descartó.

Figura 17
Comparación de Modelos Candidatos GPU



Nota. Exactitud balanceada por celda familia-clase de los siete modelos candidatos con las cinco señales, con intervalo de confianza del 95 % por *bootstrap* de familias.

El *kernel* GEMM de RAJAPerf y *LUD* de Rodinia no se resuelven con ninguna variable admisible, por una razón física. Cambian de régimen Roofline según el nivel de frecuencia, porque el punto de inflexión se desplaza con el reloj mientras la intensidad operacional del kernel es fija (*LUD* es *memory-bound* en los relojes más altos y *compute-bound* en los más bajos). Distinguirlos exigiría una variable de distancia al *ridge*, y esa variable sería fuga de la propia etiqueta.

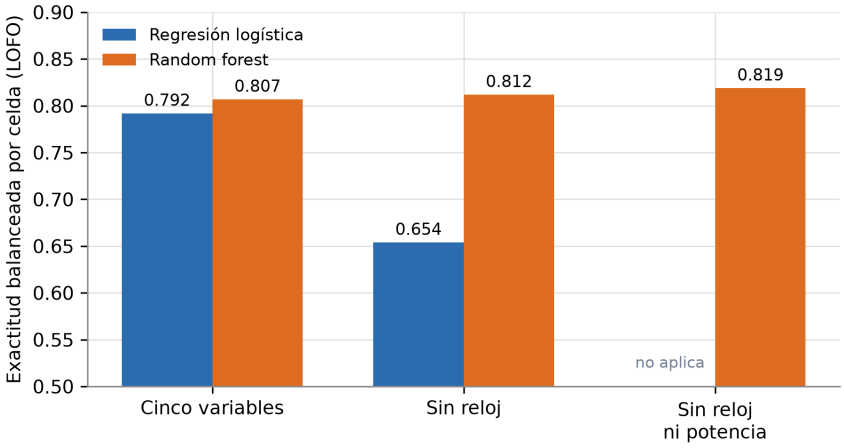
4.2.2 Candidato Operativo de GPU: Invarianza frente a la Acción del Agente

El agente fija el reloj de la GPU, y el reloj y la potencia entre las señales candidatas dependen de esa acción. Por eso la selección exige entradas invariantes (sección 3.3.3). La auditoría, sobre las mismas 483 corridas y sin mediciones nuevas, discrimina entre los tres modelos de exactitud indistinguible (Figura 18). Al fijar de forma contrafactual el reloj de cada observación en 1410 MHz, la regresión logística (0.792 con las cinco señales, ECE 0.0569) cambia de clase el 26.7 % de sus predicciones, y su exactitud baja a 0.654 al excluir el reloj. El *random forest* conserva el desempeño al retirar las entradas afectadas por la actuación, con 0.807 de exactitud balanceada por celda usando cinco variables, 0.812 sin reloj y 0.819 sin reloj ni potencia, sin diferencia distinguible entre las tres variantes. El modelo desplegado es, por tanto, el *random forest* de tres señales invariantes (utiliza-

ción de GPU, utilización de memoria y desviación estándar de esta última), elegido por invarianza y no por exactitud. La regresión logística de cinco señales queda como referencia estática.

Figura 18

Invarianza de los Candidatos GPU frente al Reloj



Nota. Exactitud balanceada por celda (LOFO, 483 corridas, 16 familias) al excluir las variables que dependen de la acción del agente.

Al evaluar cada familia con un modelo entrenado sin ella, el *random forest* clasifica correctamente el 89.2 % de las corridas, con 0.819 de exactitud balanceada por celda (Tabla 12). Reconoce el 82.5 % de las *compute-bound* y el 92.9 % de las *memory-bound* (Figura 19, izquierda). El resultado no es uniforme entre familias (figura 7.8 del material complementario). Diez de las dieciséis se clasifican sin error, mientras *rodinia_gaussian* (0.50, la familia más cercana al *ridge* del catálogo), *dual_cholesky* (0.60) y *rodinia_lud* (0.60, una de las que cambian de régimen con la frecuencia) quedan en el nivel del azar o por debajo. Su ECE con validación por familia es 0.0868, calculado sobre diez rangos de confianza de igual amplitud. La confianza alta corresponde a una exactitud alta: las 263 corridas con confianza superior a 0.95 se aciertan el 95.4 %, mientras que los rangos intermedios agrupan pocas corridas de una o dos familias y la exactitud oscila. El mínimo, en 0.75 a 0.80, son sobre todo corridas de *rodinia_gaussian* (figura 7.9 del material complementario, izquierda).

Con el umbral de 0.90 alcanza 0.900 de exactitud balanceada por celda, con una cobertura

Tabla 12
Métricas del Clasificador GPU sin Abstención

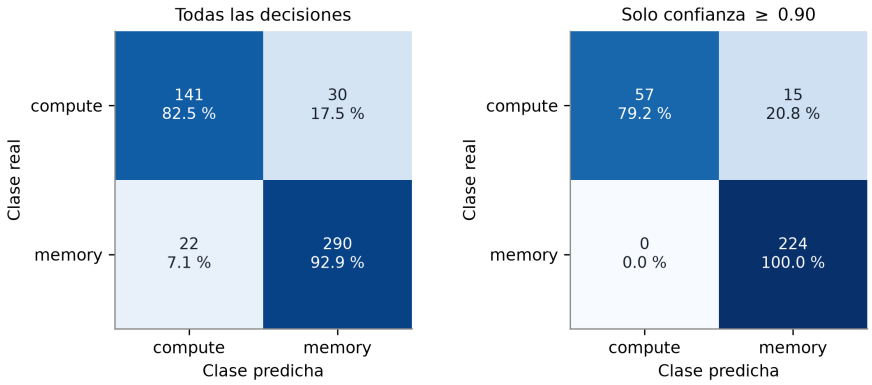
Métrica	Valor
Exactitud	0.892
Sensibilidad *compute-bound*	0.825
Sensibilidad *memory-bound*	0.929
F1 macro agrupada	0.881
AUC-ROC agrupada	0.913
Coeficiente de Matthews	0.763
Exactitud balanceada por celda	**0.819**

Nota. Sin abstención (Chicco y Jurman, 2020; Sokolova y Lapalme, 2009), validación por familia retirada sobre 483 corridas y 16 familias. El detalle con intervalos está en el material complementario.

media por celda de 65.7 % (62.1 % de las corridas; figura 7.9 del material complementario, derecha). Sobre las 296 corridas que decide, no comete errores en *memory-bound* (224 aciertos), y sus 15 errores son *compute-bound* clasificadas como memoria (Figura 19, derecha). Como la política solo actúa sobre la clase *memory-bound*, esa es la clase que importa para la decisión: de las 239 corridas que activarían F1, 224 son de memoria (precisión de la clase accionable de 0.937) y 15, el 6.3 %, serían actuaciones sobre cargas de cómputo. Estas cifras son de clasificación y no de beneficio: `rodinia_gaussian`, correctamente etiquetada como memoria, pierde EDP con F1 (sección 4.2.3). La abstención recae sobre todo en las familias de cómputo, ya que decide el 42 % de las *compute-bound* y el 72 % de las *memory-bound*. El modelo se ajustó sobre las 483 corridas después de cerrar la evaluación LOFO. La inferencia de una fila tarda 4.7 ms en mediana y 5.0 ms en el percentil 99 en Python con *scikit-learn*; dentro del agente, con ONNX Runtime, tomó 97 μs en mediana y 232 μs en el percentil 99 sobre las 658 decisiones registradas en la Fase 4. Ambas son pequeñas frente a los segundos que toma detectar y actuar sobre una fase. El agente decide una sola vez por fase, por lo que una abstención no se reevalúa.

Figura 19

Matrices de Confusión del Random Forest de GPU



Nota. Matrices de confusión del modelo desplegado bajo validación por familia (483 corridas, una semilla; las métricas del texto por celda son promedios de cinco semillas). A la izquierda, todas las decisiones; a la derecha, solo las de confianza mayor o igual a 0.90.

La importancia por permutación (figura 7.10 del material complementario) muestra que la utilización de memoria y su variabilidad concentran la dependencia predictiva del *random forest* desplegado. La utilización global de GPU aporta una señal menor.

4.2.3 Política de Frecuencia por Clase en GPU

La política GPU se derivó sobre las mismas 483 corridas, conforme al procedimiento de la sección 3.3. El análisis usa la familia como unidad de agregación para que las variantes de tamaño de un mismo algoritmo no pesen varias veces en la decisión. La comparación es pareada con la base, por lo que cinco de los 19 kernels no entran. `minibude_cuda_bm1` solo tiene corridas válidas en la base, porque en los demás niveles el reloj observado no coincidió con el solicitado. Las variantes pequeñas de `dual_axpy` y `dual_stencil`, el GEMM y el *stencil* de calor de RAJAPerf no tienen corridas aceptadas en la base, ya que son más cortas a relojes altos y no alcanzaron la fracción mínima de muestras NVML posteriores al arranque. Quedan 5 kernels *compute-bound* y 9 *memory-bound*, agrupados en 5 y 8 familias (los dos tamaños de `dual_spmv` forman una sola).

Tabla 13
Síntesis de la Política GPU

Clase	Evidencia decisiva	Política
Cómputo	F1: $+2.1$ % (IC95 $[-12.7; 14.2]$, $p = 0.438$); sin mejora confiable	No actuar
Memoria	F1: $+8.9$ % (IC95 $[3.0; 13.3]$, $p = 0.020$); siete de ocho familias mejoran	Fijar F1

Nota. Las ganancias son de EDP frente a la base, con la familia como unidad; p de la prueba de rangos con signo de Wilcoxon unilateral (mejora frente a la base). La rejilla completa está en la Figura 20.

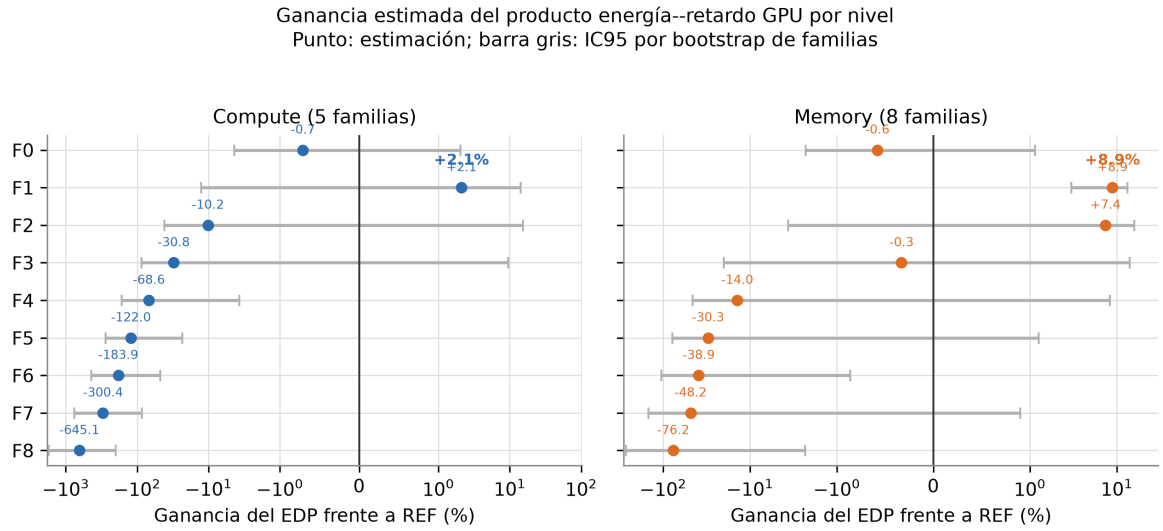
El resultado es **no actuar** en *compute-bound* y **actuar en F1 para *memory-bound*** (Tabla 13 y Figura 20). En *memory-bound*, la validación por familia retirada conserva la dirección en siete de ocho pliegues, con ganancias de 4.2 a 14.8 %, y arroja una ganancia fuera de muestra de 6.5 % (IC95 de -4.0 a 13.1 %). Como F1 se eligió entre ocho niveles después de inspeccionarlos y sin corrección por multiplicidad, el 8.9 % es una estimación interna y descriptiva; la evidencia de generalización es este 6.5 %, cuyo intervalo incluye el cero. La excepción es `rodinia_gaussian`, que pierde 27.3 %. Es la familia más cercana al *ridge* del catálogo, y por eso su tiempo responde al reloj como el de una carga de cómputo pese a su etiqueta de memoria. Se dejó en el análisis, porque quitarla después de ver la pérdida sesgaría la política a su favor. F2 alcanza una ganancia de 7.4 %, pero no es significativa al usar la familia como unidad.

El mecanismo es el mismo que en CPU. Sobre los 18 kernels con al menos tres niveles de reloj medidos (se excluye `minibude_cuda_bm1`, que solo tiene corridas en la base), los *compute-bound* tienen β mediano de 0.837 (rango 0.344-1.216, $n = 6$) y los *memory-bound* de 0.189 (rango 0.093-0.941, $n = 12$). Su tiempo apenas responde al reloj SM porque el cuello de botella, el ancho de banda de la memoria del dispositivo, pertenece a un dominio de reloj que la matriz no varía. La excepción es `rodinia_gaussian`, con $\beta = 0.941$ pese a ser *memory-bound* sin ambigüedad, y por eso es la única familia que pierde en la validación por familia retirada. `rodinia_lud`, con el menor margen de clase (0.60), gana 13.9 %. Los errores del clasificador se concentran en un kernel cuya etiqueta cambia con el nivel, y los de la política, en un kernel con etiqueta estable cuya

respuesta al reloj es similar a la de la otra clase. Ninguna de las dos familias se excluyó, porque hacerlo seleccionaría datos por el resultado que producen.

Figura 20

Ganancia del EDP GPU por Nivel y Clase, en Forma de Forest Plot



Nota. Ganancia del EDP GPU frente a la base por nivel y clase, con la familia como unidad (cómputo: 5 familias; memoria: 8). Cada punto es la ganancia estimada (en %, con su valor rotulado) y la barra gris, su IC95 por *bootstrap*, cuyos límites numéricos están en el repositorio; el rótulo en negrita señala F1. Valor p de la prueba pareada: en cómputo entre 0.438 y 1.000; en memoria, 0.020 en F1, 0.156 en F2 y entre 0.422 y 0.926 en los demás niveles. El eje horizontal es simétrico-logarítmico para mostrar a la vez F1, cercano a cero, y F8, con pérdidas de más de un orden de magnitud en *compute-bound*.

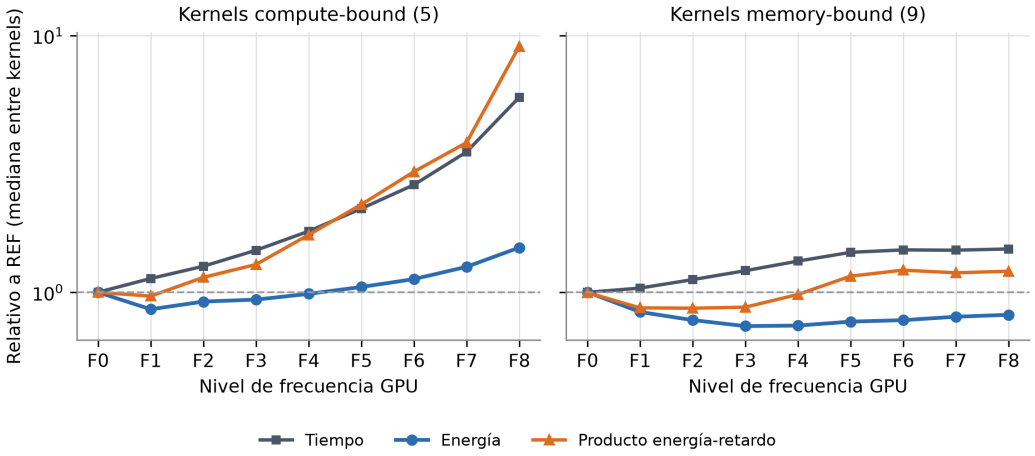
En *compute-bound*, el promedio descriptivo de F1 (+2.1 %) no es una mejora confiable (Tabla 13). La validación LOFO de la regla que escoge F1 produce una ganancia realizada de -4.8 % (IC95 de -13.4 a 0 %), y aplicada a una familia que no participó en elegir el nivel, empeora en promedio. Por eso el daemon conserva la base en *compute-bound*, y la conclusión coincide con la de CPU.

La Figura 21 hace visible el contraste entre clases. En *compute-bound* el tiempo y el EDP crecen juntos desde F1, porque la energía apenas cae. En *memory-bound* el tiempo crece más des-

pacio y la energía cae de forma sostenida hasta F3, por lo que el EDP se mantiene por debajo de 1 en F1-F2. Después la caída de energía se agota, y a partir de F5 el tiempo domina.

Figura 21

Energía, Tiempo y EDP por Nivel de Frecuencia en GPU



Nota. Mediana entre kernels del tiempo, la energía y el EDP de cada nivel de reloj de GPU, relativos a la base, para las dos clases (5 y 9 kernels con corridas en la base, kernel como unidad, descriptivo). El eje vertical es logarítmico.

La diferencia viene del piso estático de potencia. Se ajustó $P(f) = P_0 + c\,f^k$ a la potencia de cada kernel frente al reloj SM y se conservaron los 12 kernels donde el ajuste es confiable. Los otros seis se descartaron por tener menos de siete niveles con datos o $R^2 < 0.8$. El piso P_0 resulta de 38.5 W en mediana y es el 55.8 % de la potencia al reloj más alto (figura 7.12 del material complementario). Es una fracción menor que el 84 % de CPU, por lo que queda más potencia dinámica por recortar. Es consistente con que *memory-bound* encuentre una región de mejora en GPU que CPU no encontró en ninguna clase.

4.3 Resultados del Agente de Control

4.3.1 Exportación de los Modelos

Antes de evaluar el agente se comprobó que la exportación a ONNX no altera los modelos. En CPU, el XGBoost exportado se comparó con el original sobre los 1 309 755 intervalos elegibles. La mayor diferencia de probabilidad fue de 3×10^{-7} y ninguna etiqueta cambió. En GPU, el *random*

forest exportado se comparó sobre las 483 corridas, con una diferencia máxima de 9×10^{-8} y sin cambios de etiqueta. Ambas están muy por debajo de la tolerancia de 10^{-4}, por lo que lo que decide el agente en C++ coincide con lo que se validó en Python.

4.3.2 Bloqueo del Reloj de GPU y Premisa Física de la Política

Con una carga sintética de fases controladas se comprobó que, al fijar el reloj con `nvidia-smi -lgc`, el reloj observado coincide con el solicitado en los seis valores probados y que el rendimiento responde con la forma que supone la política. El de una carga *compute-bound* escala aproximadamente con el reloj, mientras que el de una carga *memory-bound* no cae hasta llegar a 510 MHz. Es la premisa física de bajar el reloj en la clase *memory-bound* sin costo de tiempo, medida sobre el mismo instrumento que aplicará el agente.

4.3.3 Latencias de Conmutación y Sobrecarga

Fijar el reloj de GPU tarda unos 50 ms, y el reloj observado alcanza el valor pedido a los 70 a 80 ms. Un cambio completo, con el asentamiento de las lecturas de NVML, cuesta entre 0.35 y 0.37 s. Con la GPU en reposo, el proceso usa entre 0.0 y 0.1 % de un núcleo. En CPU, escribir los límites en los núcleos delegados toma 27.9 ms en secuencia y 2.85 ms en paralelo, y la inferencia del clasificador toma 16.4 μs en la mediana y 19.3 μs en el percentil 99. En GPU, la inferencia con ONNX Runtime toma 97 μs en la mediana (sección 4.2.2). La permanencia mínima de GPU, diez veces la latencia de un cambio completo, fue de 3.7 s. La sobrecarga energética del brazo *sombra* se cuantifica en la Fase 4.

4.3.4 Restauración y Coordinación

En cada prueba de terminación anómala inducida (`SIGTERM` o `SIGINT`) el agente atendió la señal y devolvió el dispositivo a su estado inicial, con código de salida 0. En GPU el reloj volvió al valor nativo y en CPU el turbo y el rango de frecuencia quedaron idénticos al estado inicial, también cuando ambos procesos corrían a la vez. La prueba de coexistencia ejecutó ambos procesos en brazo activo sobre la aplicación compuesta A de GPU. El piso de CPU de 3.2 GHz se respetó en el 99.2 % de las muestras, y las cuatro restantes (de unos 0.24 s cada una) están en los flancos de subida.

4.3.5 Clasificación de Fase de GPU sobre Aplicaciones Compuestas

Con el umbral de 0.90, en la aplicación A (tres ciclos) cada brazo tuvo tres decisiones correctas, ninguna errónea y dos abstenciones. Las correctas tuvieron confianza 1.00 y las abstenciones, entre 0.57 y 0.67. En la B (dos ciclos, brazo sombra) hubo dos correctas y dos abstenciones, y *LavaMD*, una familia inédita, se clasificó correctamente con confianza 1.00. Con conteos tan pequeños, la exactitud se mide en la Fase 4.

4.4 Evaluación del Agente frente a la Base

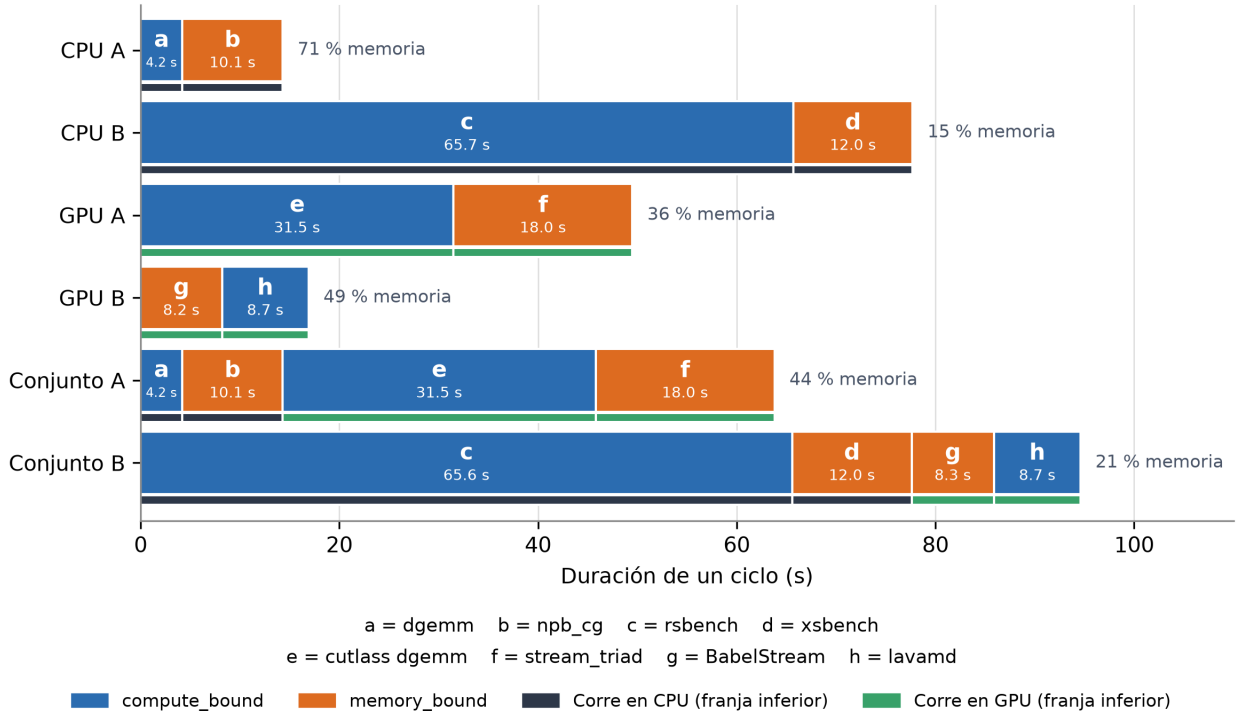
Cada razón divide la mediana de una métrica del brazo evaluado entre la mediana de esa misma métrica en el brazo base, sobre la misma combinación experimental de la matriz descrita en la sección 3.5. Las métricas son la duración, la energía de CPU, la energía de GPU y el EDP CPU+GPU. Un valor de 1 significa que el brazo iguala a la base. Un valor menor que 1 indica una mejora, y uno mayor que 1, un empeoramiento. Salvo que se indique otra cosa, la base usa el gobernador *performance*; las mediciones con *powersave* se comparan con la base de ese mismo gobernador. En los experimentos por bloques, el efecto se estima además con la media geométrica de las razones por bloque (sección 3.5.2), que es la cifra que retoman la Discusión y las Conclusiones; las razones de medianas describen las figuras.

4.4.1 Aplicaciones Compuestas y Clasificación de Fase en el Lazo

La Figura 22 presenta las aplicaciones compuestas, la aplicación A de familias vistas y la B de familias inéditas. Cada bloque de la figura es una fase, con la duración media en segundos impresa dentro del bloque y el color según su clase Roofline. La franja inferior marca el dispositivo de esa fase, y en las aplicaciones conjuntas distingue el tramo que corre en CPU del que corre en GPU. La fracción de tiempo en fases *memory-bound* varía entre 15 % en la aplicación B de CPU y 71 % en la A, lo que acota el ahorro posible de una política que actúa solo en esa clase.

Figura 22

Ciclo de Cada Aplicación Compuesta



Nota. Un ciclo de cada aplicación, por fase (letra del kernel, clave debajo; color según su clase; franja inferior según el dispositivo). A la derecha, la fracción del ciclo en fases *memory-bound*.

Cada decisión del agente se puntúa contra la fase real correspondiente. En CPU, el agente acierta entre 96.7 y 97.5 % de las decisiones sobre las familias vistas y entre 99.6 y 99.8 % sobre las inéditas (dos kernels por aplicación, ponderadas por ventanas, y por eso no comparables con la exactitud LOFO). La diferencia favorable a las inéditas no debe leerse como mejor generalización. En la aplicación B, una sola fase de cómputo ocupa el 85 % del tiempo, y ponderar por ventanas la hace dominante. El resultado depende entonces de cómo se separan esos dos kernels, no de familias arbitrarias. En GPU no hubo ninguna decisión errónea entre las que el agente tomó, pero se abstuvo con frecuencia, con dos de doce fases en A y entre cuatro y cinco de doce en B (en el alcance conjunto, tres o cuatro de veinte en A y nueve de veinte en B). Toda fase, en cualquier alcance, recibió al menos una decisión. La exactitud del brazo activo y la del brazo sombra son indistinguibles, como corresponde a un clasificador cuya entrada no depende de la acción del agente.

4.4.1.1 Entradas por Ventana frente a Entradas por Corrida. El modelo de GPU se entrenó con agregados de corridas completas y el agente lo aplica sobre ventanas de 3 s. Para contrastar esa diferencia se compararon las tres entradas de las 622 decisiones de la Fase 4 atribuibles a una fase con las de las corridas de entrenamiento del mismo kernel, en los cinco kernels vistos que aparecen en las aplicaciones compuestas. En las tres fases de memoria de E-A y en DGEMM, las ventanas reproducen la corrida: la mediana de la utilización de memoria es de 1 % en ambas unidades para los tres kernels de memoria y de 29 % frente a 28 % en DGEMM, y la utilización de GPU y la desviación estándar quedan dentro o cerca del rango de entrenamiento. En esos cuatro kernels el agente acertó las 261 decisiones, sin abstenerse y con confianza de al menos 0.98. En el *triad* de RAJAPerf la diferencia es total. Su corrida de entrenamiento, de lanzamientos breves, está dominada por el reposo (utilización mediana de GPU de 2 % y de memoria de 0 %, desviación de 45), mientras que la ventana del agente, tomada durante la actividad sostenida, ve 100 % en ambas (desviación de 13). Ahí el agente se abstuvo en 64 de 109 decisiones y acertó las 45 restantes. La diferencia de unidad no afecta, entonces, a las cargas que ocupan la GPU de forma sostenida durante toda la corrida, pero sí a las de lanzamientos cortos, cuyos agregados por corrida no describen su actividad, y explica las abstenciones en esta familia vista.

4.4.2 Tiempo, Energía, EDP y Sobrecarga del Agente frente a la Base

La Figura 23 reporta la duración, las energías de CPU y de GPU por separado y las razones frente a la base de la matriz inicial, con el agente en su configuración final. Tres hechos separan los alcances:

- En **GPU** el agente activo no cambia el resultado. El EDP CPU+GPU queda en 1.00 frente a la base en A y en B (y el brazo sombra, en 1.01 y 1.00). El agente de GPU no impone sobrecarga medible.
- En **CPU** el brazo activo queda en 1.15 (A) y 1.06 (B) veces el EDP de la base, con un aumento de 5.1 y 1.6 % en tiempo y de 11 y 5 % en energía de CPU.
- En el alcance **conjunto** el EDP queda en 1.04 (A) y 1.06 (B) veces el de la base. Con cinco

repeticiones por brazo, la prueba de Mann-Whitney da $p = 0.0079$ para sombra, activo y activo sin piso frente a la base, en A y en B. Las cinco corridas de cada brazo quedaron por encima de las de la base.

El brazo sombra, que decide pero no escribe ninguna frecuencia, ya presenta la mayor parte del exceso de CPU (1.10 y 1.05 de EDP en A y B). El agente de CPU añade entre 0.4 y 1.4 % de tiempo y entre 5 y 9 % de energía de CPU (matriz inicial), una sobrecarga total del proceso de CPU que no depende de la frecuencia que se escriba, mientras que el de GPU no presentó sobrecarga medible (sombra entre 1.002 y 1.010 del EDP de la base).

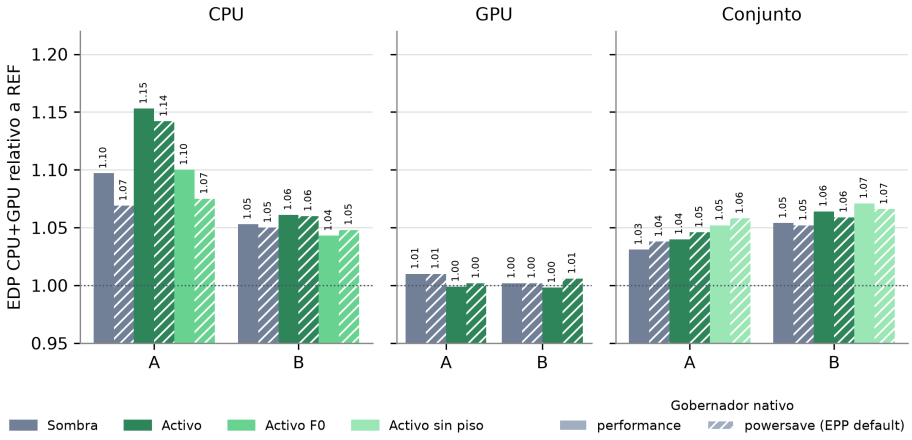
El agente de CPU usa un único hilo de inferencia de ONNX Runtime, porque con el conjunto de hilos por defecto de la biblioteca los hilos quedan en espera activa entre inferencias y compiten con la aplicación por los núcleos. Con un solo hilo, la sobrecarga restante es de potencia y no de tiempo: entre el 80 y el 93 % de la energía de CPU adicional del brazo sombra corresponde a mayor potencia a igual duración, unos 13.6 W con la aplicación de kernels vistos y 6.2 W con la de inéditos, con valores casi idénticos en la matriz inicial y en el Escenario C. No proviene del costo de una inferencia individual, que toma 16 μs. El agente de GPU, con el mismo ONNX Runtime, no añade potencia medible (-0.1 a 0.7 W), pero los dos agentes difieren en la cadencia: el de CPU lee nueve contadores en seis núcleos e infiere en cada ventana de 1 ms, unas mil veces por segundo, y su consumidor revisa la cola cada 100 μs, mientras que el de GPU muestrea cada 50 ms e infiere una vez por episodio. La comparación no separa, por tanto, el costo acumulado de las inferencias del de la lectura de contadores y los despertares, que pueden mantener activos núcleos que de otro modo reposarían, y por eso la cifra se reporta como sobrecarga total del proceso de CPU y no se atribuye a la inferencia. La causa no se aisló, y tampoco se explica que el costo sea mayor con la aplicación de kernels vistos.

La acción de frecuencia añade entre $+0.8$ % y $+5.1$ % (Figura 23; razón activo/sombra de 1.051 en CPU con kernels vistos, 1.008 en CPU con inéditos y 1.009 en el alcance conjunto, en ambos). En ningún caso mejora el EDP respecto del brazo sombra, y el máximo es en CPU con kernels vistos, donde bajar el reloj en memoria empeora el EDP 5 %. Usar F0 también en *memory*

no cambia el resultado de forma apreciable, y quitar el piso de coordinación lo empeora (Escenario C, tabla 8, abajo). Una segunda medición de la base (*base sin turbo*, el mismo estado) quedó entre 0.995 y 1.001 de la primera en EDP en las cinco combinaciones compuestas comparables del Escenario C y del Escenario E (se excluye la aplicación de familias inéditas del Escenario E, abreviada E-B más adelante, cuya mediana de tres repeticiones no es comparable por la bimodalidad de *myocyte*, sección 4.4.4). Eso acota la repetibilidad de la medición.

Figura 23

EDP CPU+GPU Relativo a la Base en la Matriz Inicial



Nota. EDP CPU+GPU, $(E_{\text{CPU}} + E_{\text{GPU}})\,T$, relativo a la REF de su gobernador, por alcance, aplicación (A: vistos, B: inéditos) y brazo, en la matriz inicial, bajo *performance* (barras lisas) y *powersave* con EPP *default* (barras rayadas). La línea punteada marca la paridad con la REF.

Una repetición independiente de los alcances de CPU y conjunto (Escenario C, tabla 8) reprodujo estas razones con diferencias menores de 0.013 y la misma exactitud de clasificación. En ella, bajar a F1 en memoria en CPU dio 1.154 de EDP frente a 1.085 con F0, y quitar el piso de coordinación empeoró el alcance conjunto entre 0.5 y 1.6 puntos.

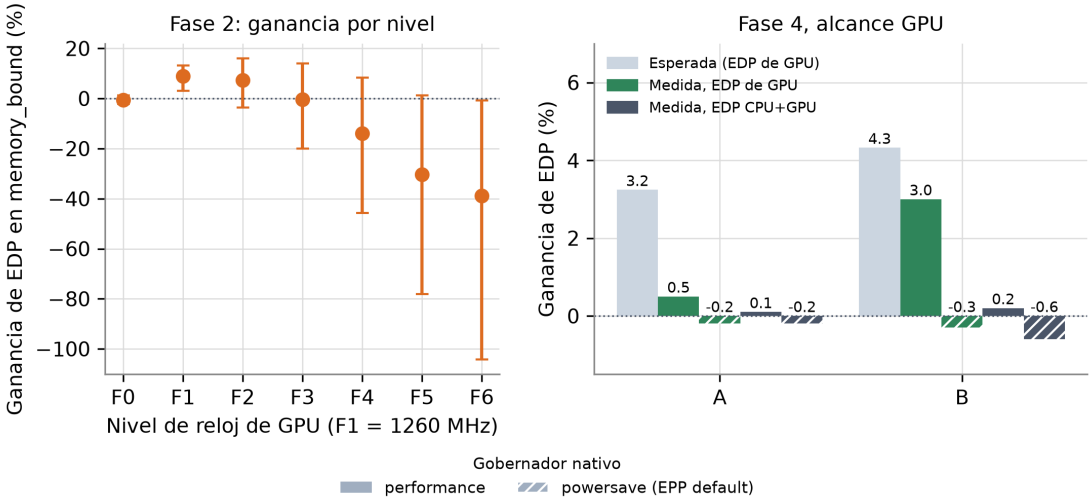
4.4.3 Potencial de la GPU

La ganancia máxima que la política de GPU puede aportar está acotada por la Fase 2. Fijar F1 (1260 MHz) en fases *memory-bound* mejora el EDP en 8.9 % (IC95 3.0 a 13.3 %), y los niveles más bajos no mejoran o empeoran (Figura 24, izquierda). En una aplicación completa ese efecto se

multiplica por la fracción de tiempo en memoria: con 36 % (A) y 49 % (B), la ganancia esperada de EDP de GPU es de 3.2 y 4.3 %. La medida con el agente activo fue de 0.5 y 3.0 % en EDP de GPU y de 0.1 y 0.2 % en el EDP CPU+GPU (Figura 24, derecha), y bajo *powersave*, de -0.2 a -0.6 %; con tres repeticiones, ninguna se separa de cero.

Figura 24

Potencial de la GPU en Fases Memory-bound



Nota. Izquierda: ganancia de EDP por nivel de reloj de GPU en fases *memory-bound* (Fase 2, IC95 sobre familias). Derecha: ganancia esperada de EDP de GPU (fracción de tiempo en memoria por la ganancia de F1) frente a la medida con el agente activo en el alcance GPU, en EDP de GPU y en EDP CPU+GPU (medianas de tres repeticiones), bajo *performance* (barras lisas) y *powersave* (barras rayadas).

En A, el agente aplicó F1 en cinco de las seis fases de *triad*, y la causa de que la ganancia medida quede por debajo de la esperada no está aislada. En B, cuatro de las seis decisiones sobre las fases de BabelStream, de 8 s, fueron abstenciones, y con esa cobertura la ganancia esperada baja a cerca de 1.4 %. La ganancia de la GPU depende, entonces, de que la aplicación pase una parte apreciable del tiempo en fases largas *memory-bound* y de que el agente no se abstenga en ellas, dos condiciones que evalúa el escenario E.

4.4.4 Escenario E: GPU Dominada por Fases de Memoria

El Escenario E (tabla 8) evalúa la política de GPU en aplicaciones dominadas por fases *memory-bound* largas, donde la reducción del reloj puede mantenerse durante más tiempo. Sus dos aplicaciones se abrevian E-A, de familias vistas, y E-B, de familias inéditas; ambas siglas se usan así en el resto del capítulo y en la discusión. La primera decisión de GPU llegó entre 13 y 16 s después del inicio de cada fase de memoria. El inicio de fase se registra al lanzar el proceso del kernel, y cada una de estas cargas pasa entre 10 y 13 s inicializando sus arreglos en el host, de hasta 10 GB, con la utilización de GPU por debajo del umbral de actividad. A ese reposo se suman los 3 s de la ventana sostenida. La fase de cómputo, que ocupa la GPU a los 0.7 s, recibió su decisión a los 3.7 s. Por eso las fases de 46 a 60 s pasaron entre 64 y 77 % de su duración en F1, que equivalen a entre 91 y 94 % del tiempo con la GPU activa.

La Tabla 14 resume las dos aplicaciones declaradas antes de medir. En B, *myocyte* recibe una etiqueta de memoria por kernel pero casi no ocupa la GPU, y solo el 42 % del ciclo corresponde a memoria con actividad GPU. Esta diferencia entre etiqueta y oportunidad de actuación condiciona la interpretación del resultado.

Tabla 14
Composición del Escenario E

Aplicación	Fases de memoria	Fase de cómputo	Ciclo en memoria
A (vistas)	Stencil (58-60 s), SpMV (46 s), AXPY (47-48 s)	DGEMM (31 s)	84 %
B (inéditas)	BabelStream (36 s), Myocyte (40-41 s)	LavaMD (9 s)	89 % por etiqueta; 42 % con actividad de GPU

Nota. Duraciones aproximadas por fase medidas antes de la matriz.

4.4.4.1 Familias Vistas (E-A). Un experimento por bloques declarado antes de medir, con cinco bloques aleatorizados y cinco repeticiones válidas por brazo, dio para el agente activo 0.9257 de energía de GPU, 0.9883 de duración y 0.9558 de EDP CPU+GPU frente a la base, como razones de medianas. El brazo sombra quedó en 0.9970, por lo que el efecto no se debe al costo de observar.

Las cinco diferencias activo-base fueron favorables, y con bloques como unidad la prueba de signos da $p = 0.0625$, el menor valor posible con cinco bloques. La media geométrica de las cinco razones de EDP CPU+GPU es 0.962 (IC95 de 0.946 a 0.977; $p = 0.002$ en la prueba t pareada sobre el logaritmo), una reducción de 2.3 a 5.4 %, y la de energía de GPU es 0.928 (IC95 de 0.923 a 0.932). La duración quedó en 0.992 (IC95 de 0.984 a 1.001), un cambio menor de 1 % cuyo intervalo incluye la paridad. El intervalo del brazo sombra incluye la paridad (0.980 a 1.011). Una medición exploratoria previa de tres repeticiones señaló en el mismo sentido bajo los dos gobernadores (0.935 de energía de GPU y 0.990 de EDP CPU+GPU bajo *performance*, y 0.939 y 0.982 bajo *powersave*; Figura 25).

El ahorro en EDP CPU+GPU es modesto porque la CPU consume el 64 % de la energía medida y no se modifica. En la medición exploratoria, con el agente de CPU en observación, su costo (3.4 % más de energía de CPU y 0.4 % más de tiempo) iguala el ahorro de GPU, y el EDP CPU+GPU vuelve a 1.003 (1.017 bajo *powersave*). Conviene, por tanto, un agente de GPU sin agente de CPU, coherente con la política de no actuar sobre la CPU.

4.4.4.2 Agente frente a F1 Fijo (E-A). El brazo F1 fijo bloquea el reloj en 1260 MHz durante toda la aplicación, incluidos los intervalos de reposo de la GPU. Con el criterio del protocolo ninguna de sus celdas fue válida, porque entre 122 y 124 muestras de cada bloque, cerca del 21 % de las muestras bajo carga, quedaron fuera del intervalo de 1230 a 1290 MHz. Todas forman un único tramo de unos 31 s, la fase DGEMM, donde el reloj bajó a entre 1065 y 1185 MHz por el límite de potencia de la A100, como ya ocurría sin bloqueo en la premedición diagnóstica (escenario F de la Tabla 8). En las tres fases de memoria el reloj se mantuvo en 1260 MHz en todas las muestras. Con el criterio restringido a esas fases (sección 3.5.2), las cinco celdas son válidas bajo los dos gobernadores.

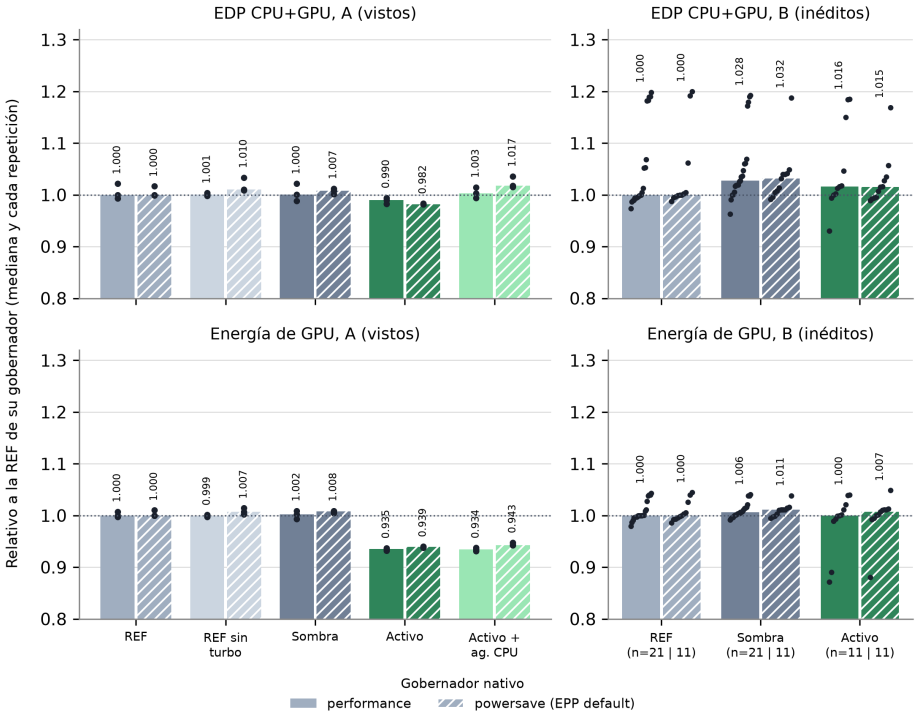
Frente a la base, F1 fijo dio 0.964 de EDP CPU+GPU (IC95 de 0.952 a 0.975) y 0.939 de energía de GPU, con los cinco bloques favorables, una reducción equivalente a la del agente. La razón agente/F1 fijo es de 0.998 en EDP CPU+GPU (IC95 de 0.990 a 1.006; tres de cinco bloques favorables al agente) y de 0.988 en energía de GPU (IC95 de 0.985 a 0.991; los cinco bloques

favorables). Bajo *powersave* el patrón se repite: F1 fijo queda en 0.981 de EDP CPU+GPU, y la razón agente/F1 fijo es de 0.993 en EDP (IC95 de 0.982 a 1.003) y de 0.986 en energía de GPU (IC95 de 0.983 a 0.989), otra vez con los cinco bloques favorables en energía de GPU. Fijar 1260 MHz durante toda la aplicación logra, entonces, la misma reducción de EDP CPU+GPU que el agente, y la conmutación por fase solo añade entre 1.2 y 1.4 % de ahorro de energía de GPU. Esa diferencia es consistente con que F1 fijo mantenga el reloj bloqueado durante el reposo inicial de las fases, mientras el agente lo deja al controlador antes de su primera decisión y después de la fase de cómputo, aunque este mecanismo no se aisló. En este escenario la fase de cómputo no discrimina entre las dos estrategias, porque el límite de potencia ya mantiene DGEMM por debajo de 1260 MHz. La ventaja que se espera del clasificador, no reducir el reloj en fases de cómputo cuyo tiempo sí depende de él (sección 4.2.3), se pone a prueba en el escenario E-C (sección 4.4.5).

4.4.4.3 Familias Inéditas (E-B). La fase de *myocyte* dura unos 41 s o unos 50 s en todos los brazos, en proporciones parecidas, por una variabilidad de la aplicación cuya causa no se identificó. Por eso se agruparon las tres corridas de esta aplicación: tres repeticiones de la etapa E y ocho de una repetición en bloques para los tres brazos, más diez de una corrida de diagnóstico de la bimodalidad, que solo incluyó la base y el brazo sombra. De ahí las 21 repeticiones de la base y de sombra y las 11 del activo (Figura 25). Sombra y activo quedan en 1.028 y 1.016 del EDP de la base y en 1.006 y 1.000 de su energía de GPU. Como las corridas agrupadas no tienen el mismo diseño ni los mismos brazos, esta comparación es descriptiva y no una prueba causal. BabelStream recibió una decisión de memoria en 2 de las 11 repeticiones del brazo activo (confianza 1.00), con 11 a 13 % menos de energía de GPU que una repetición rápida de la base, y en las otras nueve se abstuvo (confianza 0.56 a 0.67); *myocyte* no superó el umbral de actividad GPU. El límite es la cobertura de decisión sobre familias inéditas, no un empeoramiento por F1. En la medición exploratoria de tres repeticiones, F1 fijo, que sí cumplió el criterio de reloj, dejó la energía de GPU en 0.892 de la base bajo *performance* y en 0.918 bajo *powersave*, frente a 0.966 y 1.006 del agente: en esta medición exploratoria, cuando el agente se abstiene, una regla fija aprovecha una oportunidad que el agente deja pasar.

Figura 25

Escenario E: EDP CPU+GPU y Energía de GPU Bajo Ambos Gobernadores



Nota. EDP CPU+GPU (arriba) y energía de GPU (abajo), con la mediana (barra) y cada repetición (punto), para familias vistas (A) e inéditas (B), bajo *performance* (barras lisas) y *powersave* con EPP *default* (barras rayadas), cada uno frente a la REF de su gobernador. En B se agrupan las repeticiones de las corridas con la misma configuración; bajo cada brazo, el número de repeticiones de cada gobernador.

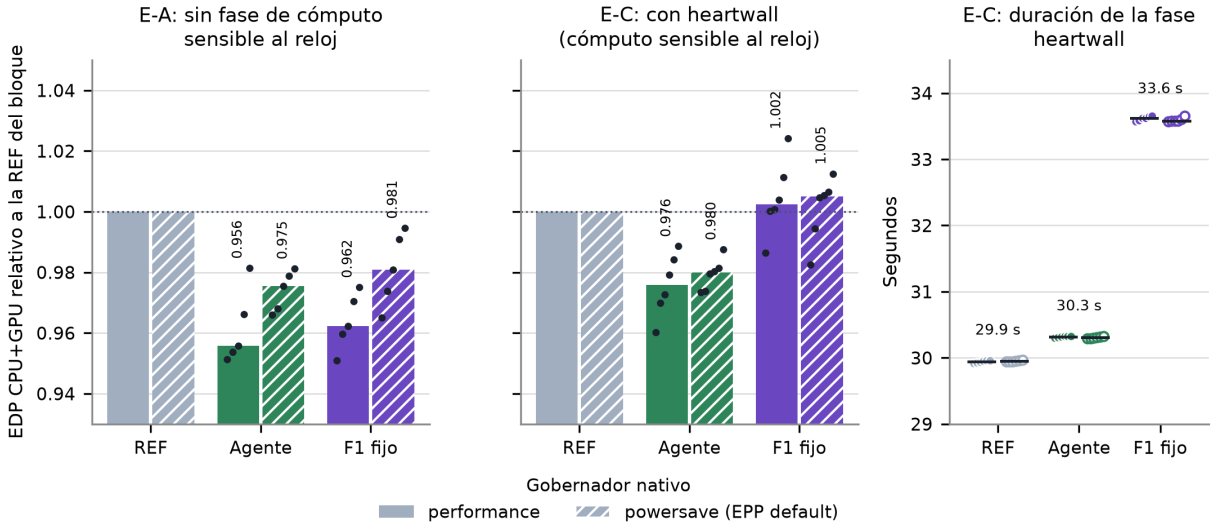
4.4.5 Escenario E-C: Fase de Cómputo Sensible al Reloj

E-C mantiene las fases de memoria de *stencil* (58 s) y SpMV (46 s) de E-A e inserta entre ellas `rodinia_heartwall`, de cómputo, con 10 097 cuadros, unos 30 s. Un sondeo previo, sin agente, confirmó que no está limitado por potencia (reloj mediano de 1410 MHz) y que su tiempo aumenta 11 % con F1. El experimento por bloques midió seis bloques aleatorizados con la base, el agente de GPU y F1 fijo bajo *performance*, y una réplica declarada antes de medir repitió el diseño bajo *powersave*, con otra semilla y cada brazo frente a la REF de su gobernador. Las 36 celdas fueron válidas, y en las doce de F1 fijo todas las muestras bajo carga quedaron en 1260 MHz.

El agente clasificó *heartwall* como *compute-bound* en 11 de los 12 bloques, con confianza 1.00, a los 3.4 a 3.6 s del inicio de la fase, y se abstuvo en el restante (confianza 0.85); en ambos casos liberó el reloj. `rodinia_heartwall` es una de las 16 familias con que se entrenó el modelo de GPU, por lo que esta clasificación es dentro de muestra. En la validación dejando su familia fuera, sobre corridas completas, el modelo acertó sus 30 corridas, pero solo 6 con confianza de al menos 0.90; con una familia inédita de este tipo el agente se habría abstenido con más frecuencia, lo que también libera el reloj, pero no se midió. Las fases de memoria recibieron F1, como en E-A. La fase de *heartwall* duró 29.9 s en la base, 30.3 s con el agente, que la ejecutó en F1 hasta su decisión, y 33.6 s con F1 fijo, bajo los dos gobernadores.

La razón agente/F1 fijo fue de 0.972 en EDP CPU+GPU (IC95 de 0.958 a 0.985), con los seis bloques favorables al agente ($p = 0.031$ en la prueba de signos, declarada como contraste principal; las comparaciones con la base se declararon como secundarias, con la misma prueba). F1 fijo alargó la aplicación 2.9 % frente a la base y 2.5 % frente al agente, y su menor energía de GPU no compensó ese tiempo. Frente a la base, el agente quedó en 0.976 de EDP CPU+GPU (IC95 de 0.965 a 0.987; seis de seis bloques, $p = 0.031$), con 0.900 de energía de GPU y la duración sin cambio apreciable (1.004), mientras que F1 fijo quedó en 1.004 de EDP (IC95 de 0.991 a 1.018), sin mejora. Bajo *powersave* el resultado se reprodujo: agente/F1 fijo quedó en 0.978 de EDP CPU+GPU (IC95 de 0.970 a 0.987; seis de seis bloques, $p = 0.031$), el agente en 0.979 de la base (seis de seis bloques, $p = 0.031$) y F1 fijo en 1.001, y la REF quedó en 0.998 del EDP de la REF bajo *performance*. La Figura 26 contrasta ambos escenarios. En este escenario, construido con el kernel más favorable a la diferencia y con una familia que el modelo conocía, la clasificación por fase conservó la reducción de EDP que el reloj fijo perdió. Es una prueba de mecanismo dentro de una aplicación compuesta, no una validación externa general.

Figura 26
Agente frente a F1 Fijo en E-A y E-C

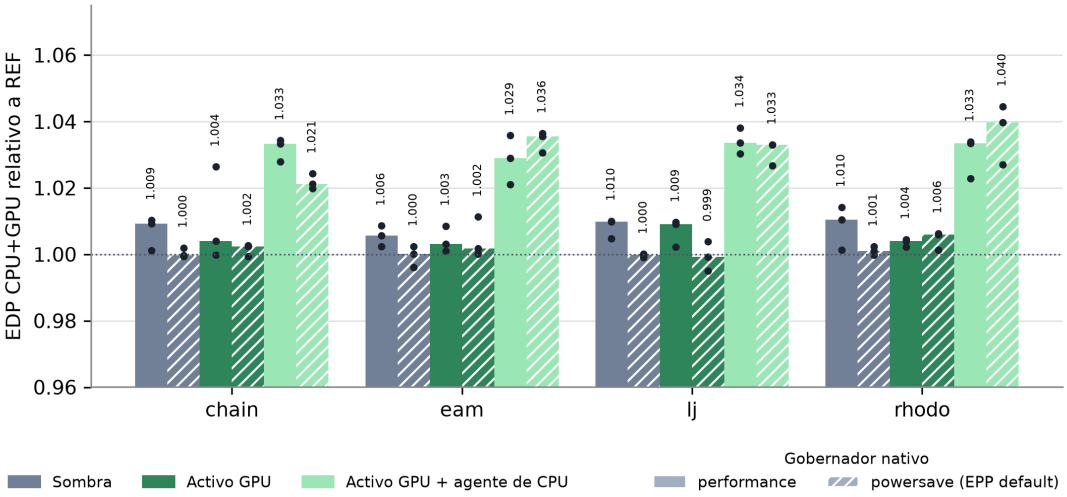


Nota. EDP CPU+GPU relativo a la REF del mismo bloque en E-A, bajo *performance* (barras lisas) y *powersave* (barras rayadas), y lo mismo en E-C; la barra es la mediana y los puntos, cada bloque. A la derecha, duración de la fase de *heartwall* en E-C por brazo, bajo *performance* (puntos llenos) y *powersave* (puntos vacíos), con la mediana como línea.

4.4.6 Aplicaciones Externas: LAMMPS y CloverLeaf

4.4.6.1 LAMMPS como Aplicación de Terceros. El Escenario D (tabla 8) evalúa LAMMPS 2023 con el paquete GPU del clúster, sin imponer fronteras de fase al programa. Se ejecutaron tres bloques de las entradas `chain`, `eam`, `lj` y `rhodo`, con la base, sombra, agente de GPU activo y una variante que mantiene el agente de CPU en observación. Las 48 celdas terminaron con código de aplicación y de daemon válido, y restauraron el estado de frecuencia. La Figura 27 muestra que el brazo activo de GPU queda entre 1.003 y 1.009 de la base en EDP, según la entrada. No se observa una mejora persistente, y la variante con el agente de CPU en observación incrementa el EDP entre 2.9 y 3.4 %. En `chain` y `rhodo` el agente no emitió decisiones, y en `lj` y `eam` solo emitió *compute-bound*, por lo que no aplicó F1. Este resultado extiende la evaluación a una aplicación externa, pero no mide el efecto de la política GPU sobre una fase actuada.

Figura 27
EDP de LAMMPS por Entrada, Brazo y Gobernador



Nota. EDP CPU+GPU de LAMMPS por entrada y brazo, relativo a la REF de su gobernador, bajo *performance* (barras lisas) y *powersave* (barras rayadas); la barra es la mediana y los puntos, cada bloque.

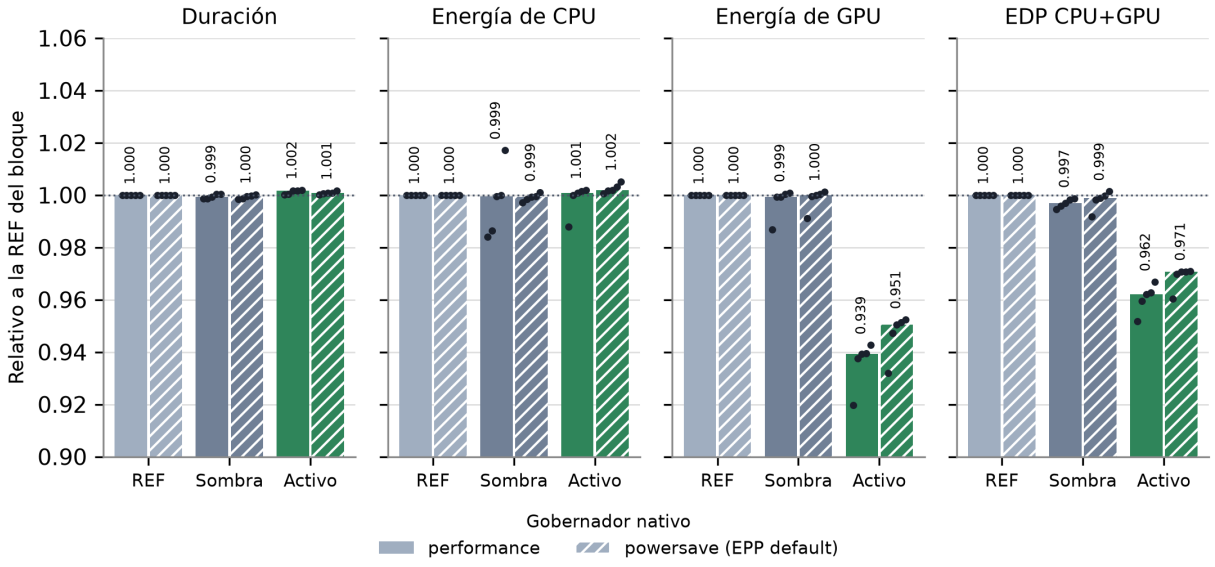
4.4.6.2 CloverLeaf CUDA Fortran como Aplicación Continua. CloverLeaf CUDA Fortran se evaluó como una aplicación continua de dinámica de fluidos, sin fronteras de fase impuestas al programa, con cinco bloques aleatorizados de la base, sombra y activo. Las quince corridas fueron válidas y su energía cinética final coincidió dentro de una tolerancia relativa de 10^{-8}. En cada corrida activa el agente emitió una decisión *memory-bound* con confianza 1.00 tras unos 3 s, y todas las muestras bajo carga posteriores quedaron en 1260 MHz. Como CloverLeaf no aporta una etiqueta Roofline por fase, el experimento verifica la actuación y su efecto energético, pero no puntúa la decisión; y como esa única decisión se sostuvo toda la ejecución, el agente se comportó como un F1 fijo mientras la GPU estuvo activa.

El brazo activo mantuvo la duración en 1.0015 de la base, redujo la energía de GPU a 0.9393 y dejó el EDP CPU+GPU en 0.9625 (medianas). El brazo sombra quedó en 0.997 de EDP (IC95 de 0.995 a 0.999), dentro del rango de repetibilidad de la base (sección 4.4.2). Las cinco diferencias activo-base fueron favorables ($p = 0.0625$, el mínimo con cinco bloques), y la media

geométrica de EDP CPU+GPU es 0.961 (IC95 de 0.954 a 0.968; $p < 0.001$ en la prueba t pareada sobre el logaritmo), una reducción de 3.2 a 4.6 %, con 0.936 de energía de GPU (IC95 de 0.924 a 0.947) y 1.001 de duración (IC95 de 1.000 a 1.002). La actuación reduce la energía del dispositivo sin modificar de forma apreciable el tiempo (Figura 28). CloverLeaf valida, por tanto, una acción sostenida y su ahorro en una aplicación externa, pero no la exactitud por fase ni una ventaja sobre F1 fijo, que no se midió en este escenario.

Figura 28

Resultados de CloverLeaf frente a la Base



Nota. CloverLeaf CUDA Fortran: duración, energía de CPU, energía de GPU y EDP CPU+GPU de cada brazo relativos a la REF de su bloque, bajo *performance* (barras lisas) y *powersave* (barras rayadas). La barra es la mediana de los cinco bloques, los puntos son los bloques individuales y la línea punteada marca la paridad con la base.

4.4.7 Contraste con el Gobernador Nativo powersave

Los escenarios de la Fase 4, salvo la repetición del Escenario C, se repitieron con los núcleos delegados en *powersave*, con EPP *default*, que la plataforma resolvió a *balance_performance*, turbo desactivado y el mismo techo de 3.2 GHz, con los mismos brazos y bloques (en E-B, 11 repeticiones por brazo). Cada brazo se compara con la REF de su propio gobernador y aparece con barras rayadas

en las figuras anteriores. La REF bajo *powersave* quedó entre 0.974 y 1.021 del EDP de la REF bajo *performance*, según el escenario y la entrada, sin un sentido sistemático y dentro de la dispersión esperable con tres a once repeticiones por REF. En la serie de ocho bloques de E-B la REF se midió con turbo activado; una REF sin turbo de la misma serie quedó en 1.001 de su EDP, de modo que, con el techo de 3.2 GHz, el turbo no cambia la comparación. Como las dos REF resultaron casi iguales, esta serie no constituye una base energéticamente distinta, sino una comprobación de que las conclusiones no dependen del gobernador.

Los efectos del agente se reprodujeron. En la matriz inicial el agente de CPU volvió a empeorar el EDP (1.142 y 1.060 en A y B, frente a 1.153 y 1.061 bajo *performance*, y 1.046 y 1.059 en el alcance conjunto, frente a 1.040 y 1.064), y el de GPU quedó entre 0.999 y 1.006. En E-A y CloverLeaf los cinco bloques volvieron a favorecer al agente ($p = 0.0625$), con medias geométricas de EDP CPU+GPU de 0.974 (IC95 de 0.968 a 0.980) y 0.968 (IC95 de 0.964 a 0.972), frente a 0.962 y 0.961 bajo *performance*, y de energía de GPU de 0.932 y 0.947, frente a 0.928 y 0.936; el intervalo del brazo sombra incluyó la paridad en ambos. En E-B, sombra y activo quedaron en 1.032 y 1.015 del EDP de su REF (1.028 y 1.016 bajo *performance*; descriptivo), y la bimodalidad de *myocyte* reapareció en todos los brazos. El activo con el agente de CPU en observación quedó en 1.147, pero 6 de sus 11 repeticiones cayeron en la fase lenta de *myocyte*, y con solo las rápidas queda en 1.001. En LAMMPS el agente de GPU quedó entre 0.999 y 1.006, y la variante con el agente de CPU en observación subió el EDP entre 2.1 y 4.0 % (2.9 a 3.4 % bajo *performance*). El gobernador nativo cambia algunas magnitudes en décimas de punto, pero ninguna conclusión: el costo del agente de CPU, la ausencia de costo del de GPU, la mejora en fases de memoria largas, la ventaja sobre F1 fijo en E-C y la paridad en LAMMPS y en las familias inéditas se mantienen bajo ambos gobernadores.

5. Discusión

5.1 Validez de la Medición

Buena parte de los trabajos de selección de frecuencia revisados (Ali et al., 2023; Guerreiro et al., 2019; Wang et al., 2024) evalúan la calidad de la decisión y dan por hecho que el hardware ejecuta la frecuencia que se le pide. En esta plataforma ese supuesto no se cumplió en todos los casos, y por ello se verificó de forma explícita. En CPU, restringir solo los procesadores lógicos delegados no cambió el rendimiento, y el reloj respondió únicamente cuando se restringieron también sus hermanos de *SMT*. En GPU, el bloqueo de reloj sostuvo los valores pedidos en la prueba de actuación y en las fases de memoria del brazo F1 fijo, pero en la fase de cómputo el límite de potencia de la A100 lo llevó por debajo de lo solicitado. Ambos casos indican que una corrida debe aceptarse por el reloj que efectivamente se midió y no por el comando que se envió. La verificación de los techos Roofline con una herramienta independiente y el control del número de contadores simultáneos cumplieron el mismo propósito, que fue evitar que una etiqueta se construyera sobre una medición equivocada.

Algo parecido ocurrió con la precisión aritmética en GPU (sección 4.2.1). El tipo de dato que declara un kernel no siempre coincide con el que ejecuta el hardware, y en un kernel cercano al punto de inflexión, como *rodinia_hotspot*, esa diferencia basta para cambiar la etiqueta Roofline. Cuando la etiqueta depende del número de operaciones de cada precisión, conviene caracterizar la precisión efectiva antes de entrenar.

5.2 Generalización de los Clasificadores

En CPU, la exactitud depende sobre todo de cómo se separan los datos para validar. Si los intervalos se mezclan al azar, el modelo reconoce algoritmos que ya vio durante el entrenamiento y la exactitud sube a 0.874; si cada familia se retira completa, baja a 0.728 (sección 4.1.3.3). Esta segunda cifra es la que interesa para un agente que encontrará programas nuevos. Bajo ese protocolo, ni cambiar el modelo, ni ampliar las variables, ni buscar hiperparámetros mejoró el resultado de forma distinguible, mientras que añadir familias sí lo hizo. Una explicación coherente con estos

datos es que la etiqueta se define por la razón entre FLOPs y bytes de DRAM, y el modelo solo observa fallos de caché, ciclos detenidos e instrucciones retiradas, cuya relación con esa razón cambia de un algoritmo a otro. La abstención filtra parte de los errores, pero no los que el modelo comete con confianza alta en familias que no conoce.

Littman y Deakin (2025) abordaron el mismo problema en CPU con un enfoque cercano, basado en contadores de `perf`, etiquetas confirmadas con el modelo Roofline y modelos clásicos, sobre ocho aplicaciones de referencia. Obtuvieron exactitud perfecta con una partición estratificada al azar y entre 0.833 y 0.917 con validación cruzada dejando uno fuera, y observaron que cada programa ocupa una región propia en el espacio de contadores, una «huella de rendimiento». Como su validación no agrupa por familia, las variantes de tamaño de un mismo programa pueden quedar a ambos lados de la partición, y esa huella favorece la exactitud. Es el mismo efecto que en este trabajo separa el 0.874 del 0.728, por lo que sus cifras no son comparables con la de familia retirada.

En GPU, Guerreiro et al. (2019), con entradas parecidas a las de este trabajo (utilización gráfica y de DRAM), y Ali et al. (2023) también evaluaron sus modelos en aplicaciones que no participaron en el ajuste. La diferencia principal está en lo que se predice. Ellos aprenden directamente la respuesta medida a DVFS, mientras que aquí el modelo aprende la clase Roofline y la política se deriva por separado, con su propia evidencia estadística. Esa separación tiene un costo, visible en `rodinia_gaussian`, que es *memory-bound* pero cuyo tiempo responde al reloj como el de una carga de cómputo. La elección del modelo también cambió al tener en cuenta el lazo de control. La regresión logística de cinco señales tenía buena exactitud y calibración, pero usaba el reloj y la potencia, que son precisamente lo que el agente modifica, y el *random forest* desplegado prescinde de ellas para evitar esa retroalimentación. Su umbral de 0.90 sacrifica cobertura a cambio de decisiones confiables, y como el agente no reevalúa una fase después de abstenerse, las abstenciones ante memoria inédita siguen siendo su principal limitación. Acelerar la inferencia ayudaría poco, porque tarda microsegundos frente a los segundos que toma detectar una fase.

5.3 Alcance de la Ganancia Energética

Que la política resultara distinta en CPU y en GPU se explica por la forma en que cada dispositivo consume potencia. En la CPU de esta plataforma, la mayor parte de la potencia es un piso que no baja con el reloj, de modo que reducir la frecuencia ahorra poco y alarga la ejecución, y la política fue no actuar en ninguna de las dos clases. Ese piso incluye el *uncore*, la DRAM y los núcleos no delegados, que el agente no controla, por lo que la conclusión se refiere a la frecuencia de los núcleos delegados y no a la CPU en general. En la GPU el piso pesa menos y las fases limitadas por memoria apenas responden al reloj SM, lo que deja margen para que F1 mejore el EDP (sección 4.2.3). Aun así, la clase Roofline indica dónde hay una oportunidad sin garantizar que todos sus kernels la aprovechen, como mostró `rodinia_gaussian`.

La Fase 4 mostró en qué aplicaciones esa oportunidad se convierte en un ahorro real. El brazo sombra permitió separar lo que cuesta observar de lo que aporta actuar: el agente de GPU no tuvo un costo medible, mientras que el de CPU añadió entre 5 y 9 % de energía de CPU. E-A (sección 4.4.4) y CloverLeaf (sección 4.4.6.2), dos aplicaciones con fases de memoria largas, redujeron el EDP CPU+GPU en los cinco bloques, y LAMMPS y las familias inéditas quedaron cerca de la paridad porque el agente no llegó a aplicar F1 de forma sostenida. La mejora depende, en consecuencia, de que existan fases de memoria largas y de que el clasificador decida sobre ellas con confianza.

La comparación con un reloj fijo en F1 precisa cuánto de esa mejora se debe a clasificar por fase. En E-A, fijar 1260 MHz durante toda la aplicación dio el mismo EDP CPU+GPU que el agente, porque su única fase de cómputo ya corría por debajo de ese reloj por el límite de potencia de la A100. En CloverLeaf el agente mantuvo F1 mientras la GPU estuvo activa, comportamiento equivalente al de un reloj fijo, y en la medición exploratoria de E-B un reloj fijo ahorró energía donde el agente se abstuvo. En estas aplicaciones, la regla estática «F1 mientras la GPU esté ocupada» habría obtenido el mismo resultado. La clasificación marca la diferencia cuando hay fases de cómputo cuyo tiempo sí depende del reloj, y E-C mostró una dirección consistente con ello: con `rodinia_heartwall` entre las fases de memoria, el agente superó a F1 fijo en 2.8 % de EDP

CPU+GPU en los seis bloques (2.2 % bajo *powersave*), mientras que F1 fijo no mejoró la base (Figura 26). Este resultado es una prueba de mecanismo y no una validación externa: E-C se construyó con el kernel más favorable a esa diferencia y con una familia que el modelo conocía, dentro de una aplicación compuesta, de modo que su magnitud no se traslada a cualquier fase de cómputo. La gestión dinámica por fases en una aplicación externa con fases verificadas de forma independiente sigue sin validarse, porque CloverLeaf recibió una sola decisión y LAMMPS no activó F1.

Frente a los gobernadores nativos, el balance del agente depende de cuál de sus dos procesos se considere. El de GPU redujo el EDP CPU+GPU en las aplicaciones con fases de memoria largas sin alargar la ejecución, bajo ambos gobernadores, y su inferencia, de 97 μs una vez por episodio, no tuvo un costo apreciable. El de CPU, en cambio, tuvo una sobrecarga total de 6 a 14 W sin actuar, que no separa lectura de contadores, despertares e inferencia y no tiene contrapartida cuando la política es no actuar. En la plataforma medida se recomienda desplegar únicamente el proceso de GPU. El agente reacciona a cada fase en lugar de anticiparla, porque necesita 3 s de actividad para decidir; anticiparla exigiría información como las fronteras de lanzamiento que podría aportar CUPTI.

5.4 Contraste con el Estado del Arte

Las reducciones obtenidas son menores que las de los trabajos de GPU revisados. Ali et al. (2023) redujeron la energía de GPU 29.6 % con una pérdida de rendimiento de 5.2 % en GA100, la misma arquitectura de la A100 usada aquí; DRLCAP (Wang et al., 2024), 22 % con menos de 3 % de pérdida, y Simsek et al. (2024), hasta 7.82 % con cerca de 2.95 % de pérdida. En E-A y CloverLeaf, la energía de GPU bajó 7.2 y 6.4 % sin cambios de duración mayores de 0.8 %, y el EDP CPU+GPU, 3.8 y 3.9 % (medias geométricas por bloque). La distancia se explica sobre todo por la métrica que se reporta, el criterio con que se decide y el margen que deja el propio dispositivo.

Esos trabajos miden la energía de la GPU, mientras que aquí la cifra principal es el EDP CPU+GPU (Ecuación 12), que incluye la energía de paquete de la CPU. En E-A esa energía es cerca del 64 % del total y el agente de GPU no la modifica; medida solo en la GPU, la reducción es del orden de la de Simsek et al. El criterio también difiere. Ali et al. y DRLCAP aceptan perder

algo de rendimiento a cambio de energía, mientras que la política de este trabajo minimiza el EDP, que penaliza el tiempo, y por ello se quedó en F1, el nivel más cercano al nativo, en lugar de bajar a relojes con más ahorro. En cuanto al margen, el controlador de la A100 PCIe ya reduce el reloj cuando la carga alcanza el límite de potencia, como en la fase DGEMM de E-A, y ese ahorro queda incluido en la base contra la que se compara. Por último, aquí se aplicó una misma regla a toda una clase, lo que limita el ahorro pero evita caracterizar cada aplicación de antemano.

En CPU, los sistemas de ejecución EAR, GEOPM y COUNTDOWN (Cesarini et al., 2021; Corbalan et al., 2020; Eastep et al., 2017) obtienen ahorros actuando durante las esperas de comunicación o con modelos por región. El resultado negativo de este trabajo no los contradice, porque se limita a los núcleos delegados de un nodo, sin comunicación entre nodos ni actuación sobre el *uncore*.

6. Conclusiones

Las mediciones se validaron antes de entrenar los modelos: se comprobó que los contadores leídos con Perf no se multiplexaran, que la frecuencia efectiva coincidiera con la solicitada y que los techos Roofline calibrados concordaran con una herramienta independiente, mientras la energía se tomaba de RAPL en CPU y de NVML en GPU. Sobre esa base, recorriendo diez estados de frecuencia en cada dispositivo, la campaña de CPU produjo 1470 corridas aceptadas y 1 309 755 intervalos elegibles, y la de GPU, 483 corridas de 16 familias. Contar con datos verificados permitió estudiar por separado si las etiquetas eran correctas, si los modelos generalizaban a algoritmos nuevos y si actuar sobre la frecuencia ahorraba energía.

En CPU, el clasificador XGBoost alcanzó una exactitud balanceada por celda de 0.728 cuando cada familia algorítmica se dejó fuera del entrenamiento, y de 0.738 sobre el 72.2 % de los intervalos en que decidió con el umbral de 0.85. Ese resultado dependió más de la variedad de familias del catálogo que del algoritmo elegido o de sus hiperparámetros. La política derivada fue

no actuar en ninguna de las dos clases Roofline, porque en la asignación evaluada cerca de 85 W de la potencia de paquete y DRAM del zócalo de la carga, el 84 % de la medida al mayor nivel fijo, permanecieron al reducir el reloj de los seis núcleos delegados. Incluso eligiendo el mejor nivel para cada kernel, la ganancia media sería de 2.2 %, y en las aplicaciones compuestas de la Fase 4 actuar sobre la frecuencia de CPU tampoco mejoró el EDP frente a la base.

En GPU, el modelo desplegado es un *random forest* con tres señales que no cambian cuando el agente actúa, con una exactitud balanceada por celda de 0.819 sobre corridas completas y un error de calibración esperado de 0.0868. La regresión logística de cinco señales era una alternativa competitiva en la evaluación estática (0.792), pero dependía del reloj y de la potencia, que el propio agente modifica. Con el umbral de 0.90, las decisiones emitidas alcanzaron 0.900 de exactitud, con una cobertura media por celda de 65.7 %. La política fijó F1 para las fases *memory-bound*. Al elegir el nivel sin la familia evaluada, la mejora del EDP de GPU fue de 6.5 %, con un intervalo de -4.0 a 13.1 % que incluye el cero; el 8.9 % (IC95 de 3.0 a 13.3 %) obtenido sobre todas las familias es una estimación interna, porque F1 se eligió entre ocho niveles con los mismos datos. En las fases *compute-bound* se conservó la gestión automática del controlador. Con lecturas de NVML, el clasificador aproxima una etiqueta que de otro modo exigiría perfilar con Nsight Compute, que en este trabajo alargó unas 60 veces la ejecución de un kernel, y lo hace sin un costo medible sobre la aplicación. Que la GPU sí ofrezca margen se debe a que su piso de potencia pesa menos que el de la CPU y a que las cargas limitadas por memoria apenas responden al reloj de cómputo.

El agente quedó como dos procesos en C++, uno para cada dispositivo, que corren en paralelo a la aplicación sin modificarla y devuelven el hardware a su estado original si se interrumpen. Cada inferencia con ONNX Runtime toma 16 μs en CPU y 97 μs en GPU, un tiempo despreciable frente al que exige detectar una fase. Se evaluó con tres brazos: la base, la sombra, que observa sin actuar, y el activo. Con solo su proceso de GPU, redujo el EDP CPU+GPU 3.8 % en la serie por bloques de E-A y 3.9 % en CloverLeaf CUDA Fortran, una aplicación externa de ejecución continua, y la energía de GPU 7.2 y 6.4 %, sin alargar la ejecución. Son medias geométricas por bloque, con intervalos de confianza del 95 % calculados *post hoc* de 2.3 a 5.4 % y de 3.2 a 4.6 % en

el EDP CPU+GPU. En ambas aplicaciones los cinco bloques favorecieron al agente ($p = 0.0625$, el mínimo posible con cinco bloques), una dirección consistente en esos escenarios más que una confirmación estadística, y el patrón se repitió con el gobernador *powersave*.

Estos resultados deben acotarse. E-A se diseñó en condiciones favorables, y CloverLeaf no ofrece una etiqueta física independiente con la cual juzgar la clasificación. En E-A, además, un reloj fijo en 1260 MHz logró la misma reducción que el agente; en CloverLeaf no se midió esa alternativa, aunque el agente sostuvo F1 durante toda la actividad de la GPU. La ventaja de clasificar por fase apareció en E-C, donde una fase de cómputo sensible al reloj se intercaló entre las mismas fases de memoria: el agente superó a F1 fijo en 2.8 % de EDP CPU+GPU (IC95 de 1.5 a 4.2 %; seis de seis bloques, $p = 0.031$) porque no bajó el reloj en esa fase. E-C es una prueba de mecanismo en una aplicación compuesta, construida con el kernel más favorable y una familia vista, y no sustituye la validación en una aplicación externa con fases verificadas, que queda pendiente. El caso contrario se observó en E-B, donde, en una medición exploratoria de tres repeticiones, un reloj fijo dejó la energía de GPU en 0.892 de la base frente a 0.966 del agente, que se abstuvo en la mayoría de las fases. En LAMMPS y en las familias GPU inéditas, el agente quedó cerca de la paridad.

El proceso de CPU, en cambio, empeoró el resultado. En la matriz inicial aumentó el EDP entre 6 y 15 % en el alcance de CPU, con tres repeticiones y sin contraste estadístico, y entre 4 y 6 % en el alcance conjunto, con cinco repeticiones ($p = 0.0079$). La mayor parte de ese aumento proviene de la sobrecarga total del proceso, que el tiempo de una inferencia individual no explica, y por ello no se recomienda desplegarlo en esta plataforma. Respecto de la pregunta de investigación, esa sobrecarga total incluye el costo de la inferencia y lo acota por arriba: en GPU no tuvo un efecto medible sobre el rendimiento ni sobre el EDP, y en CPU aumentó el EDP, sin que ese aumento pueda atribuirse a la inferencia.

6.1 Limitaciones

Del lado de los modelos, los intervalos de CPU provienen de 30 familias y no son unidades independientes de generalización; su etiqueta tiene la granularidad del *uncore*, cercana a 10 ms, y

la abstención no elimina los errores de alta confianza en familias nuevas. El modelo de GPU se entrenó con corridas completas y el agente lo aplica sobre ventanas de 3 s. Las entradas coinciden en las cargas de actividad sostenida, pero no en las de lanzamientos cortos, donde el agente se abstiene, y la comparación entre ambas unidades solo pudo hacerse en los cinco kernels vistos que aparecen en las aplicaciones compuestas; reentrenar con ventanas eliminaría esa diferencia. Ese modelo se evaluó únicamente con validación por familia retirada sobre 16 familias, NVML no observa los contadores que definen el régimen Roofline y algunos kernels cambian de régimen al variar la frecuencia. Además, el modelo de GPU se entrenó con corridas de los diez niveles de reloj sin recibir el reloj como entrada, aunque la etiqueta de algunos kernels cambia con el nivel; definir la clase en el estado previo a la acción, o predecir directamente el beneficio de F1, exigiría reentrenar y no se abordó. El umbral de 0.90 se eligió por exactitud y cobertura, no por una función de costo que penalice más una actuación errónea. La política F1, por su parte, se validó con ocho familias *memory-bound*, cuatro de ellas de elaboración propia, y la ganancia del nivel elegido no se corrigió por la selección entre niveles.

En el diseño experimental, el intervalo del efecto en E-A y CloverLeaf se calculó después de medir, y con cinco bloques no fue posible verificar la normalidad. En E-A, el contraste con F1 fijo usa un criterio de reloj definido después de medir, y E-C, el escenario que muestra la ventaja de la clasificación, se construyó con el kernel de cómputo más sensible al reloj del catálogo, de una familia que el modelo vio en el entrenamiento, dentro de una aplicación compuesta. La sobrecarga total del proceso de CPU, de 6 a 14 W, no tiene una causa aislada, aunque la cadencia de lectura e inferencia es la candidata más probable; separar sus componentes exigiría ablaciones (proceso ocioso, muestreo, construcción de variables, inferencia y actuación) que no se realizaron.

En cuanto a la plataforma, la conclusión sobre CPU se limita a la frecuencia de los seis núcleos delegados de un zócalo en esta asignación, con el turbo desactivado y sin actuación sobre el *uncore*, y no se generaliza a la CPU completa, a otros controladores de frecuencia ni a la ocupación de todos los núcleos. La base se midió con los dos gobernadores que `intel_pstate` expone en esta plataforma, *performance* y *powersave*, con el turbo desactivado salvo en la serie de ocho bloques

de E-B bajo *powersave*; *ondemand* y *schedutil* no están disponibles con ese controlador. En los seis escenarios medidos con ambos gobernadores, *powersave* no produjo una diferencia apreciable en la base ni cambió las conclusiones sobre el agente. Todas las mediciones provienen de un único nodo, y las aplicaciones de familias vistas reutilizan kernels del entrenamiento, lo que restringe la extrapolación a otras aplicaciones y plataformas.

6.2 Trabajo Futuro

El paso siguiente en GPU es ampliar las familias externas, en especial las de fases *memory-bound* que hoy llevan al agente a abstenerse, y evaluar una sola vez el modelo y el umbral ya congelados. Convendría también repetir el contraste entre la base, el agente y F1 fijo en aplicaciones externas que tengan fases de cómputo sensibles al reloj, para estimar la ventaja de conmutar por fase fuera de un escenario construido. Para mejorar la respuesta en fases breves, habría que medir por separado la latencia de detección, la de recolección y la de actuación, y estudiar una reevaluación controlada de las abstenciones.

En la observación, CUPTI Activity (NVIDIA, 2026a) podría aportar fronteras de fase para construir etiquetas independientes, siempre que antes se mida su intrusión, mientras NVML se mantiene como entrada ligera del agente. Las métricas de actividad de DCGM (`fp64_active`, `dram_active`), usadas por Ali et al. (2023), podrían además reemplazar la utilización de NVML por una aproximación más directa al régimen Roofline.

En CPU, la salida *revisar* tendría utilidad si el agente midiera la etiqueta física en una fracción pequeña de intervalos. Conviene ampliar el catálogo con familias alejadas de los perfiles ya cubiertos y reservar algunas para una prueba externa sellada. Antes de revisar la política de CPU, sin embargo, es necesario reducir el costo de observación.

Referencias Bibliográficas

Akiba, T., Sano, S., Yanase, T., Ohta, T., y Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. En *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 2623-2631). https://doi.org/10.1145/3292500.3330701

Ali, G., Side, M., Bhalachandra, S., Wright, N. J., y Chen, Y. (2023). An automated and portable method for selecting an optimal GPU frequency. *Future Generation Computer Systems*, *149*, 71-88. https://doi.org/10.1016/j.future.2023.07.011

Antepara, O., Oliker, L., y Williams, S. (2025). Roofline analysis of tightly-coupled CPU-GPU superchips: A study on MI300A and GH200. En *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1205-1216). https://doi.org/10.1145/3731599.3767497

Antici, F., Bartolini, A., Kiziltan, Z., Babaoglu, O., y Kodama, Y. (2024). MCBound: An online framework to characterize and classify memory/Compute-bound HPC jobs. En *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1-15). https://doi.org/10.1109/SC41406.2024.00062

Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., Simon, H. D., Venkatakrishnan, V., y Weeratunga, S. K. (1991). The NAS parallel benchmarks. *The International Journal of Supercomputer Applications*, *5*(3), 63-73. https://doi.org/10.1177/109434209100500306

Beamer, S., Asanović, K., y Patterson, D. (2015). *The GAP benchmark suite* [Preprint]. arXiv. https://arxiv.org/abs/1508.03619

Beckingsale, D. A., Burmark, J., Hornung, R., Jones, H., Killian, W., Kunen, A. J., Pearce, O., Robinson, P., Ryujin, B. S., y Scogland, T. R. W. (2019). RAJA: Portable performance for large-scale scientific applications. En *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)* (pp. 71-81). https://doi.org/10.1109/P3HPC49587.201

9.00012

Bergstra, J. S., Bardenet, R., Bengio, Y., y Kégl, B. (2011). Algorithms for hyper-parameter optimization. En *Advances in Neural Information Processing Systems* (Vol. 24, pp. 2546-2554).

Breiman, L. (2001). Random forests. *Machine Learning*, *45*, 5-32. https://doi.org/10.1023/A:1010933404324

Brodersen, K. H., Ong, C. S., Stephan, K. E., y Buhmann, J. M. (2010). The balanced accuracy and its posterior distribution. En *2010 20th International Conference on Pattern Recognition* (pp. 3121-3124). https://doi.org/10.1109/ICPR.2010.764

Calore, E., Gabbana, A., Schifano, S. F., y Tripiccione, R. (2017). Evaluation of DVFS techniques on modern HPC processors and accelerators for energy-aware applications. *Concurrency and Computation: Practice and Experience*, *29*(12), e4143. https://doi.org/10.1002/cpe.4143

Cawley, G. C., y Talbot, N. L. C. (2010). On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, *11*, 2079-2107.

Cesarini, D., Bartolini, A., Bonfà, P., Cavazzoni, C., y Benini, L. (2021). COUNTDOWN: A run-time library for performance-neutral energy saving in MPI applications. *IEEE Transactions on Computers*, *70*(5), 682-695. https://doi.org/10.1109/TC.2020.2995269

Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., y Skadron, K. (2009). Rodinia: A benchmark suite for heterogeneous computing. En *2009 IEEE International Symposium on Workload Characterization (IISWC)* (pp. 44-54). https://doi.org/10.1109/IISWC.2009.5306797

Chen, T., y Guestrin, C. (2016). XGBoost: A scalable tree boosting system. En *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785-794). https://doi.org/10.1145/2939672.2939785

Chen, Y., Davis, T. A., Hager, W. W., y Rajamanickam, S. (2008). Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software*, *35*(3), 22:1-22:14. https://doi.org/10.1145/1391989.1391995

Chicco, D., y Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics*, *21*(1), 6. https:

//doi.org/10.1186/s12864-019-6413-7

Corbalan, J., Alonso, L., Aneas, J., y Brochard, L. (2020). Energy optimization and analysis with EAR. En *2020 IEEE International Conference on Cluster Computing (CLUSTER)* (pp. 464-472). ht tps://doi.org/10.1109/CLUSTER49012.2020.00067

Costa, M. T., Georgiadou, A., White, J. B., III, Alvarez, B. V., Polo, J., Shin, W., Navaux, P. O. A., Messer, B., y Lorenzon, A. F. (2025). Characterizing the impact of GPU power management on an Exascale system. En *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25)* (pp. 1524-1533). https://doi.org/10.1145/3731599.3767702

David, H., Gorbatov, E., Hanebutte, U. R., Khanna, R., y Le, C. (2010). RAPL: Memory power estimation and capping. En *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)* (pp. 189-194). https://doi.org/10.1145/1840845.1840883

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1-30.

Eastep, J., Sylvester, S., Cantalupo, C., Geltz, B., Ardanaz, F., Al-Rawi, A., Livingston, K., Keceli, F., Maiterth, M., y Jana, S. (2017). Global extensible open power manager: A vehicle for HPC community collaboration on co-designed energy management solutions. En *High performance computing (ISC High Performance 2017)* (pp. 394-412). Springer International Publishing. ht tps://doi.org/10.1007/978-3-319-58667-0_21

Efron, B., y Tibshirani, R. J. (1993). *An introduction to the bootstrap*. Chapman & Hall.

Geurts, P., Ernst, D., y Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, *63*(1), 3-42. https://doi.org/10.1007/s10994-006-6226-1

Gonzalez, R., Gordon, B. M., y Horowitz, M. A. (1997). Supply and threshold voltage scaling for low power CMOS. *IEEE Journal of Solid-State Circuits*, *32*(8), 1210-1216. https://doi.org/10.1109/4.604077

Goto, K., y van de Geijn, R. A. (2008). Anatomy of high-performance matrix multiplication. *ACM*

Transactions on Mathematical Software, *34*(3), Artículo 12. https://doi.org/10.1145/1356052.1356053

Gregg, B. (2020). *Systems performance: Enterprise and the cloud* (2.ª ed.). Addison-Wesley.

Guerreiro, J., Ilic, A., Roma, N., y Tomás, P. (2019). DVFS-aware application classification to improve GPGPUs energy efficiency. *Parallel Computing*, *83*, 93-117. https://doi.org/10.1016/j.parco.2018.02.001

Hebbar, R., y Milenković, A. (2022). PMU-events-driven DVFS techniques for improving energy efficiency of modern processors. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, *7*(1), 1-31. https://doi.org/10.1145/3538645

Heroux, M. A., Doerfler, D. W., Crozier, P. S., Willenbring, J. M., Edwards, H. C., Williams, A., Rajan, M., Keiter, E. R., Thornquist, H. K., y Numrich, R. W. (2009). *Improving performance via mini-applications* (Informe N.º SAND2009-5574). Sandia National Laboratories.

Hollander, M., Wolfe, D. A., y Chicken, E. (2014). *Nonparametric statistical methods* (3.ª ed.). John Wiley & Sons. https://doi.org/10.1002/9781119196037

Intel Corporation. (2021). *3rd gen Intel Xeon processor scalable family, codename Ice Lake, uncore performance monitoring reference manual* (Informe N.º 639778, rev. 1.0). https://cdrdv2-public.intel.com/679093/639778%20ICX%20UPG%20v1.pdf

Intel Corporation. (2025). *Intel Advisor user guide: Analyze CPU Roofline*. Intel Advisor User Guide. Recuperado el 12 de agosto de 2026, de https://cdrdv2-public.intel.com/848588/advisor_user-guide_2025.1-766448-848588.pdf

Intel Corporation. (2026a). *Intel 64 and IA-32 architectures software developer's manual, volume 3B: System programming guide, part 2 (Performance monitoring, NMI watchdog interaction with fixed-function and general-purpose counters)*. Recuperado el 12 de agosto de 2026, de https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

Intel Corporation. (2026b). *perfmon: Performance monitoring events for Intel architecture processors (FP_ARITH_INST_RETIRED event, Ice Lake family)*. Recuperado el 10 de agosto de 2026, de https://github.com/intel/perfmon

James, G., Witten, D., Hastie, T., y Tibshirani, R. (2021). *An introduction to statistical learning: With applications in R* (2.ª ed.). Springer. https://doi.org/10.1007/978-1-0716-1418-1

Kaufman, S., Rosset, S., Perlich, C., y Stitelman, O. (2012). Leakage in data mining: Formulation, detection, and avoidance. *ACM Transactions on Knowledge Discovery from Data*, *6*(4), Artículo 15. https://doi.org/10.1145/2382577.2382579

Kerrisk, M. (2010). *The Linux programming interface: A Linux and UNIX system programming handbook*. No Starch Press.

Laros, J. H., III, Pedretti, K., Kelly, S. M., Shu, W., Ferreira, K., Van Dyke, J., y Vaughan, C. (2013). Energy delay product. En *Energy-efficient high performance computing: Measurement and tuning* (pp. 51-55). Springer. https://doi.org/10.1007/978-1-4471-4492-2_8

Linux Kernel Documentation. (2026a). *CPU performance scaling*. Recuperado el 5 de septiembre de 2026, de https://docs.kernel.org/admin-guide/pm/cpufreq.html

Linux Kernel Documentation. (2026b). *`intel_pstate` CPU performance scaling driver*. Recuperado el 5 de septiembre de 2026, de https://docs.kernel.org/admin-guide/pm/intel_pstate.html

Linux man-pages project. (2026). *perf_event_open(2) - Linux manual page*. Recuperado el 29 de agosto de 2026, de https://man7.org/linux/man-pages/man2/perf_event_open.2.html

Linux perf project. (2026). *Perf-stat(1) measurement overhead documentation*. Recuperado el 20 de septiembre de 2026, de https://man7.org/linux/man-pages/man1/perf-stat.1.html

Littman, L., y Deakin, T. (2025). *Classifying performance bounds using machine learning* [Póster]. SC25: International Conference for High Performance Computing, Networking, Storage and Analysis. https://sc25.supercomputing.org/proceedings/posters/poster_files/post215s2-file3.pdf. Conjunto de datos: https://doi.org/10.5281/zenodo.17194638

Liu, T.-Y., Guo, J., y Huang, B. (2024). Efficient cross-platform multiplexing of hardware performance counters via adaptive grouping. *ACM Transactions on Architecture and Code Optimization*, *21*(1), 1-26. https://doi.org/10.1145/3629525

Mallinson, A. C., Beckingsale, D. A., Gaudin, W. P., Herdman, J. A., Levesque, J. M., y Jarvis, S. A. (2013). CloverLeaf: Preparing hydrodynamics codes for Exascale. En *Proceedings of the Cray*

User Group 2013. https://cug.org/proceedings/cug2013_proceedings/includes/files/pap130.pdf

McCalpin, J. D. (1995, diciembre). Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, 19-25.

McCarty, S. (2015). *Architecting containers part 1: Why understanding user space vs. kernel space matters*. Red Hat Blog. Recuperado el 3 de abril de 2025, de https://www.redhat.com/en/blog/architecting-containers-part-1-why-understanding-user-space-vs-kernel-space-matters

Montgomery, D. C. (2017). *Design and analysis of experiments* (9.ª ed.). Wiley.

National Energy Research Scientific Computing Center. (2026). *Roofline performance model*. NERSC Documentation. Recuperado el 7 de agosto de 2026, de https://docs.nersc.gov/tools/performance/roofline/

NVIDIA. (2026a). *CUDA profiling tools interface (CUPTI)*. NVIDIA Developer Documentation. Recuperado el 18 de septiembre de 2026, de https://docs.nvidia.com/cupti/

NVIDIA. (2026b). *Nsight Compute*. NVIDIA Developer Documentation. Recuperado el 8 de agosto de 2026, de https://docs.nvidia.com/nsight-compute/

NVIDIA. (2026c). *NVIDIA management library (NVML)*. NVIDIA Developer Documentation. Recuperado el 3 de abril de 2026, de https://docs.nvidia.com/deploy/nvml-api/index.html

Plimpton, S. (1995). Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, *117*(1), 1-19. https://doi.org/10.1006/jcph.1995.1039

Roberts, D. R., Bahn, V., Ciuti, S., Boyce, M. S., Elith, J., Guillera-Arroita, G., Hauenstein, S., Lahoz-Monfort, J. J., Schröder, B., Thuiller, W., Warton, D. I., Wintle, B. A., Hartig, F., y Dormann, C. F. (2017). Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. *Ecography*, *40*(8), 913-929. https://doi.org/10.1111/ecog.02881

Simsek, O. S., Piccinali, J.-G., y Ciorba, F. M. (2024). Increasing energy efficiency of astrophysics simulations through GPU frequency scaling. En *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1826-1834). https://doi.org/10.1109/SCW63240.2024.00229

Sokolova, M., y Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, *45*(4), 427-437. https://doi.org/10.1016/j.ipm.2009.03.002

Thamm, P., y Leser, U. (2025). *Strategies to measure energy consumption using RAPL during workflow execution on commodity clusters*. arXiv. https://arxiv.org/abs/2505.09375

Velicka, D., Vysocky, O., y Riha, L. (2025). Methodology for GPU frequency switching latency measurement. En *Proceedings of the 2025 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (pp. 830-839). https://doi.org/10.1109/IPDPSW66978.2025.00133

Wang, Y., Hao, M., He, H., Zhang, W., Tang, Q., Sun, X., y Wang, Z. (2024). DRLCap: Runtime GPU frequency capping with deep reinforcement learning. *IEEE Transactions on Sustainable Computing*, *9*(5), 712-726. https://doi.org/10.1109/TSUSC.2024.3362697

Weaver, V. M. (2015). Self-monitoring overhead of the Linux perf event performance counter interface. En *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (pp. 102-111). https://doi.org/10.1109/ISPASS.2015.7095789

Weaver, V. M., y McKee, S. A. (2008). Can hardware performance counters be trusted? En *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)* (pp. 141-150). https://doi.org/10.1109/IISWC.2008.4636099

Williams, S., Waterman, A., y Patterson, D. (2009). Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, *52*(4), 65-76. https://doi.org/10.1145/1498765.1498785