Para este trabajo usamos el workload **jpeg2k\_dec/jpg2kdec\_testfile.j2k**, que corresponde a una aplicación de **decodificación de imágenes JPEG2000** incluida en gem5.

Elegimos este workload porque nos permitió **evaluar el comportamiento del procesador de una forma equilibrada**, ya que combina operaciones de cómputo y acceso a memoria. Esto nos ayudó a observar cómo afectan los cambios microarquitectónicos tanto al **pipeline del CPU** como a las **memorias caché**.

Además, jpeg2k\_dec tiene varias ventajas frente a otros workloads disponibles:

* Se ejecuta rápido y sin errores, lo que lo hace ideal para correr muchas simulaciones seguidas.
* No depende de librerías externas ni requiere un sistema operativo completo, a diferencia de otros workloads más pesados como H264\_dec o SPEC CPU.
* Su comportamiento es predecible y estable, lo que facilita comparar los resultados entre configuraciones diferentes.

Probamos algunos workloads alternativos, pero los descartamos:

* **H264\_dec** tenía tiempos de simulación mucho más altos.

Por eso, jpeg2k\_dec fue el mejor punto medio: permite ver cambios reales en el rendimiento, el consumo de energía y la eficiencia (EDP), pero con simulaciones manejables en tiempo y recursos.

**EXPERIMENTO 1: 100 simulaciones FUERZA BRUTA**

Para el primer bloque de pruebas, realizamos **100 simulaciones automáticas** utilizando el **simulador gem5** con el modelo del procesador **Cortex-A76**.  
El objetivo fue observar cómo cambia el rendimiento del procesador al modificar varios **parámetros microarquitectónicos clave**, especialmente los relacionados con **memoria caché** y **ancho de pipeline**.

Las simulaciones se automatizaron mediante un script en **Python**, que ejecutaba gem5 con diferentes combinaciones de parámetros y guardaba los resultados (como stats.txt y config.ini) en carpetas separadas.  
Cada simulación se ejecutó con el workload **jpeg2k\_dec**, usando la siguiente línea base de comando:

./build/ARM/gem5.fast scripts/CortexA76\_scripts\_gem5/CortexA76.py \

-c workloads/jpeg2k\_dec/jpg2k\_dec \

-o "-i workloads/jpeg2k\_dec/jpg2kdec\_testfile.j2k -o image.pgm"

### **Parámetros modificados**

Durante este proceso, exploramos las combinaciones de los siguientes parámetros:

* **Tamaño de caché L1 de instrucciones (--l1i\_size)**  
  Valores probados: 32kB, 64kB, 128kB
* **Tamaño de caché L1 de datos (--l1d\_size)**  
  Valores probados: 32kB, 64kB, 128kB
* **Tamaño de caché L2 (--l2\_size)**  
  Valores probados: 128kB, 256kB, 512kB
* **Anchos del pipeline (--fetch\_width, --decode\_width, --commit\_width)**  
  Valores probados: 2, 3, 4

### **Métricas analizadas**

De cada simulación se extrajeron principalmente los siguientes valores del archivo stats.txt:

* system.cpu.ipc → Instrucciones por ciclo (rendimiento).
* system.cpu.cpi → Ciclos por instrucción (latencia).
* simSeconds → Tiempo total simulado.
* hostSeconds → Tiempo real de ejecución en el host.
* system.cpu.numCycles → Número total de ciclos simulados.

Con estos datos se pudo construir una visión inicial del rendimiento sin incluir todavía las métricas de energía (que se calcularon después con McPAT).

### **Resultados esperados**

El objetivo de estas primeras 100 simulaciones **no era realizar una optimización completa**, sino **comprender el comportamiento del procesador** frente a diferentes configuraciones microarquitectónicas.  
En esta fase se trabajó exclusivamente con los resultados generados por **gem5**, **sin integrar aún el análisis energético mediante McPAT**, ya que la prioridad era observar cómo varía el rendimiento puro y el tiempo de simulación con cambios en la arquitectura base.

Por lo tanto, esta etapa se enfocó en identificar **tendencias generales y comportamientos característicos**, más que en encontrar una configuración ideal.  
En particular, se buscaba responder a preguntas como:

* ¿Cómo influye el tamaño de las cachés L1 y L2 sobre el rendimiento (IPC)?
* ¿De qué manera los anchos del *pipeline* (fetch, decode, commit) afectan el CPI y la eficiencia del procesador?
* ¿En qué configuraciones comienzan a aparecer cuellos de botella o mejoras significativas?
* ¿Existen combinaciones que mantengan un buen equilibrio entre rendimiento y estabilidad de simulación?

En esta fase inicial, las simulaciones permitieron obtener una **base comparativa sólida** sobre cómo los cambios individuales en los parámetros afectan el comportamiento del procesador Cortex-A76 dentro del entorno de gem5.  
Este análisis servirá como punto de partida para la siguiente etapa, donde se integrará **McPAT** para estudiar el **consumo de energía y la eficiencia energética (EDP)**, con el fin de identificar las configuraciones óptimas tanto en desempeño como en potencia.

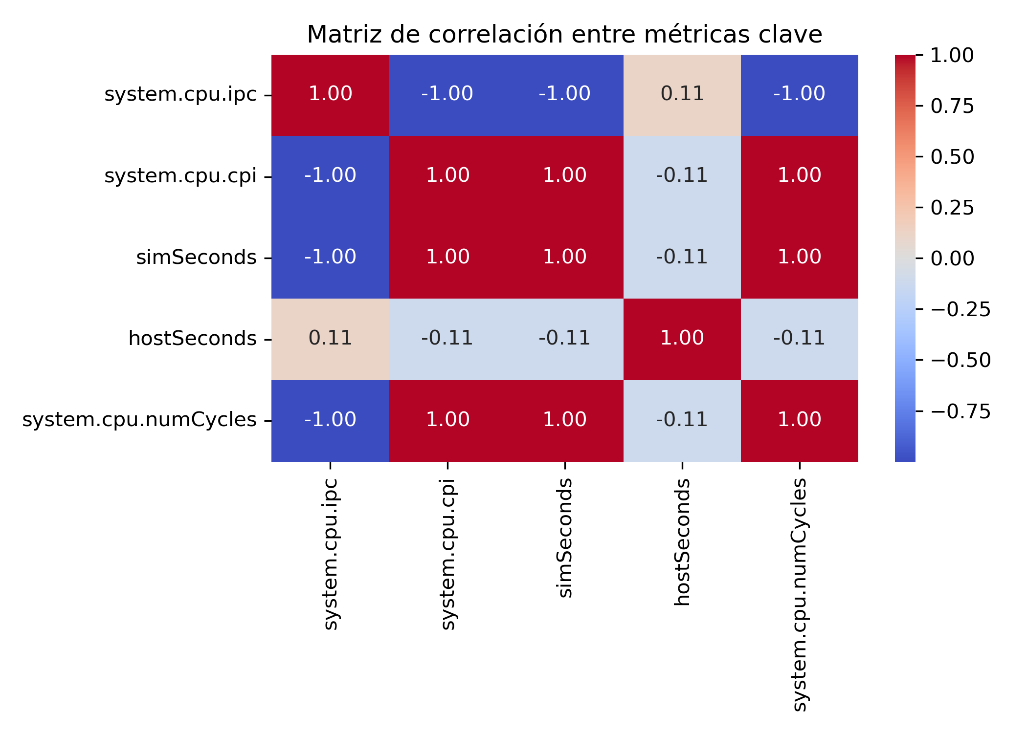
Métricas de correlación

Esta gráfica muestra la correlación estadística entre las métricas principales del procesador obtenidas en las simulaciones:

* system.cpu.ipc (Instrucciones por ciclo)
* system.cpu.cpi (Ciclos por instrucción)
* simInsts (Número total de instrucciones ejecutadas)
* hostSeconds (Tiempo real de ejecución en el host)
* system.cpu.numCycles (Cantidad total de ciclos de CPU)

**Cómo interpretarla:**

* Los valores cercanos a **+1** indican correlación positiva (ambas métricas crecen juntas).
* Los valores cercanos a **-1** indican correlación inversa (cuando una crece, la otra baja).
* Por ejemplo, un valor cercano a **-1 entre IPC y CPI** es esperable, ya que son inversos directos (un aumento en IPC implica una disminución de CPI).
* Este heatmap permite identificar **qué variables tienen relación directa con el rendimiento** y cuáles son independientes.

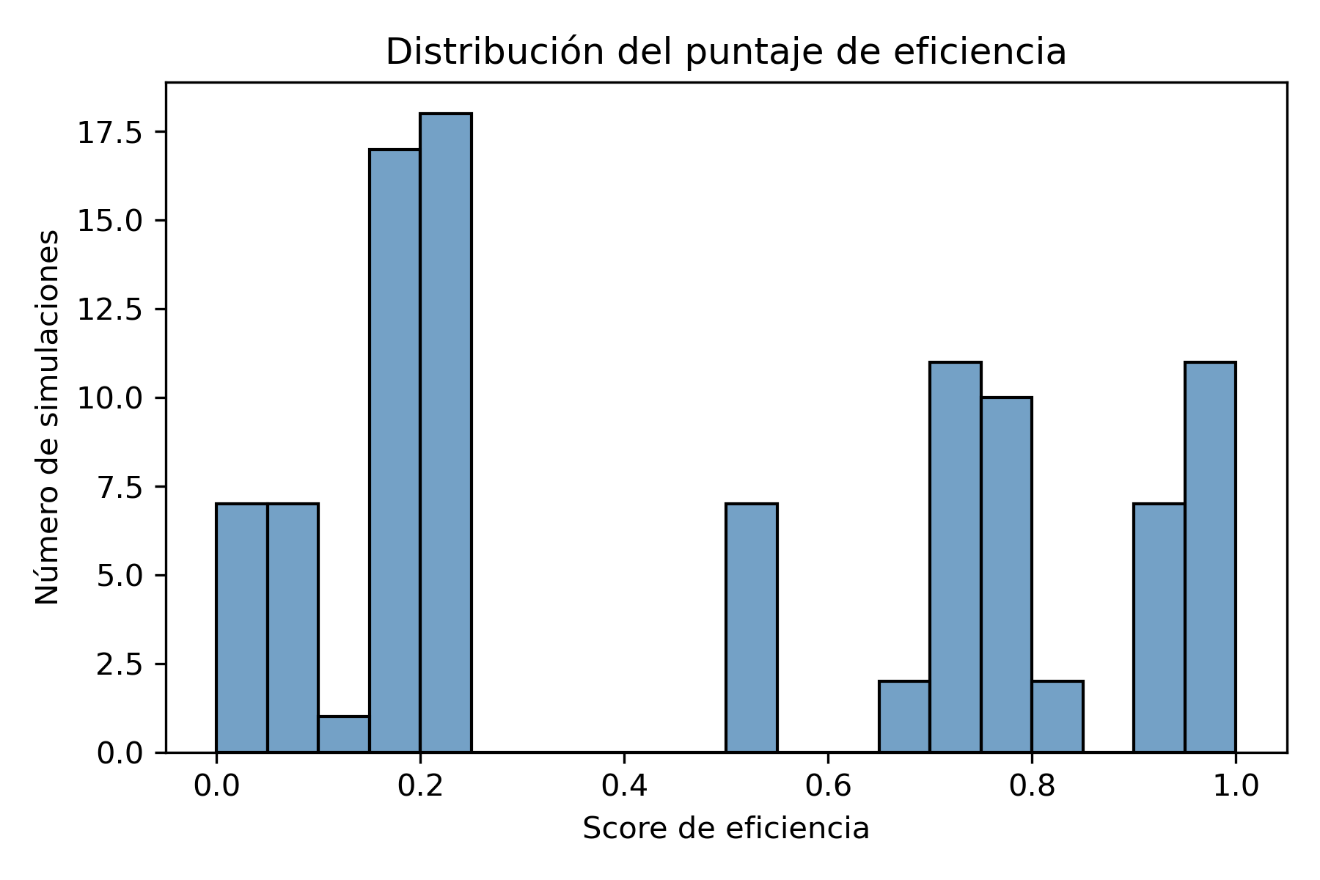


Gráfica: distribucion\_eficiencia

**Qué muestra:**  
Representa la distribución de un puntaje de eficiencia calculado a partir del IPC (o alguna métrica derivada de rendimiento).  
Cada barra representa cuántas simulaciones lograron un nivel de rendimiento similar.

**Cómo interpretarla:**

* Si la gráfica tiene una forma **asimétrica o sesgada**, significa que solo unas pocas configuraciones obtuvieron un alto rendimiento.
* Si la distribución es más **uniforme**, los cambios en los parámetros microarquitectónicos no afectaron significativamente el desempeño.

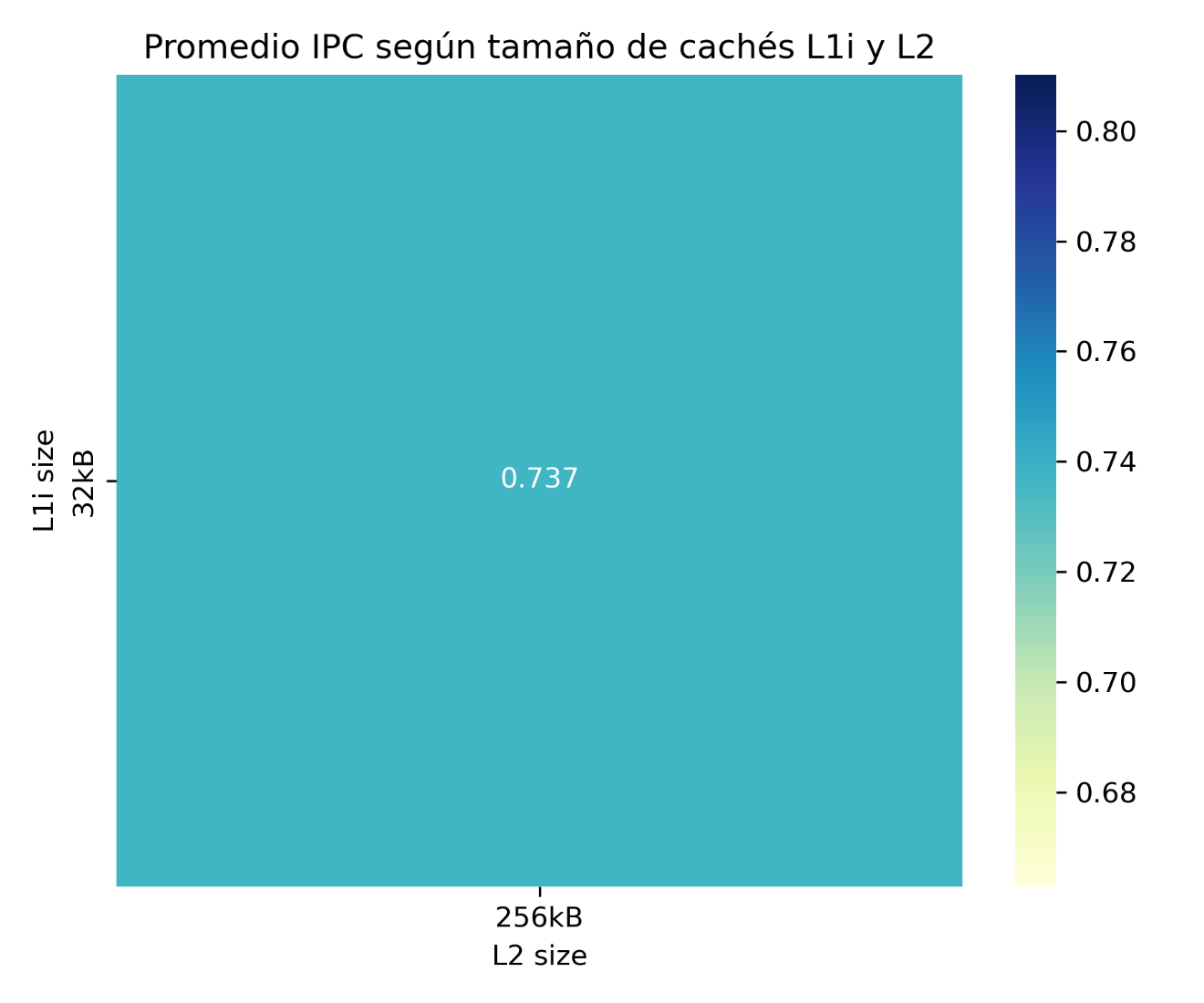


Gráfica: ipc\_heatmap\_caches

**Qué muestra:**  
Representa el **promedio del IPC** obtenido en función del tamaño de las cachés L1 y L2.  
El eje vertical corresponde al tamaño de la caché L1, el eje horizontal al tamaño de la L2, y el color al valor del IPC.

**Cómo interpretarla:**

* Los colores más oscuros indican mayor IPC, es decir, mejor rendimiento.
* Permite visualizar si aumentar el tamaño de caché tiene un impacto real o si llega un punto donde **no hay ganancias significativas** (donde el color deja de cambiar).

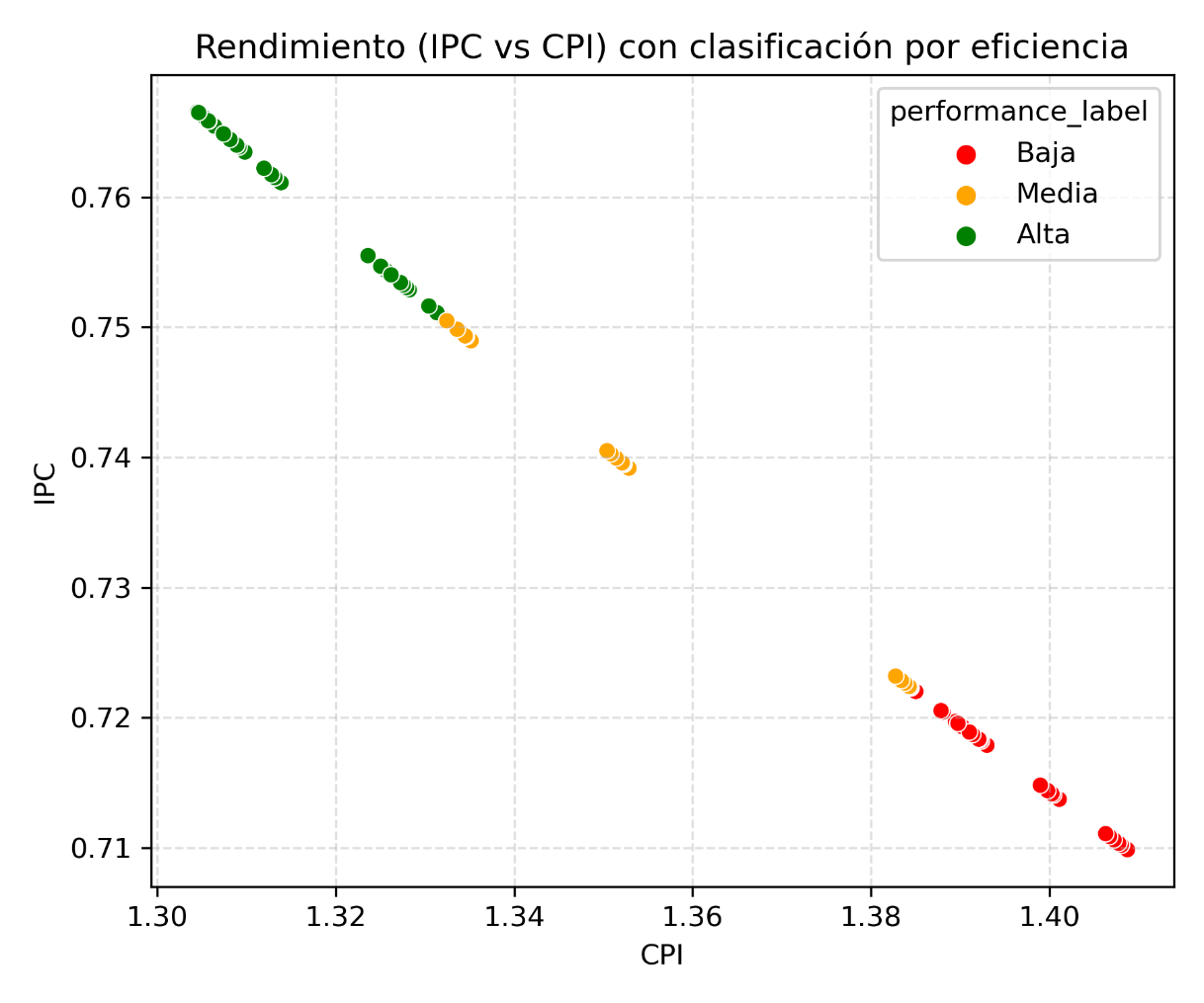


Gráfica: ipc\_vs\_cpi\_etiquetas

**Qué muestra:**  
Relaciona el IPC (rendimiento) con el CPI (eficiencia por instrucción) y clasifica cada simulación según su rendimiento (por ejemplo: bajo, medio, alto).

**Cómo interpretarla:**

* Los puntos más arriba (mayor IPC y menor CPI) corresponden a las configuraciones más rápidas.
* Las etiquetas de color ayudan a visualizar la agrupación de rendimiento.
* Permite verificar la consistencia del modelo: si las etiquetas de alta eficiencia se concentran en un rango específico, el comportamiento del procesador es estable.



Gráfica: ipc\_vs\_fetchwidth

**Qué muestra:**  
El promedio del IPC obtenido para diferentes anchos del pipeline de *fetch* (ancho de captura de instrucciones por ciclo).

**Cómo interpretarla:**

* Permite evaluar cómo cambia el rendimiento al modificar el número de instrucciones que el procesador puede traer por ciclo.
* Un aumento en el fetch\_width debería incrementar el IPC, pero hasta cierto punto — después de un límite, el beneficio se estanca.

## ****EXPERIMENTO 2: Optimización con algoritmo Greedy****

Después de realizar las 100 simulaciones iniciales con fuerza bruta, cuyo propósito era analizar el comportamiento general del procesador ante variaciones microarquitectónicas, se desarrolló un **segundo experimento utilizando un enfoque heurístico basado en el algoritmo Greedy** (codificado en Python).

### **Objetivo del enfoque Greedy**

Mientras el primer experimento buscaba entender tendencias generales (cómo afectaban el tamaño de las cachés, los anchos del pipeline y los parámetros del predictor de saltos), el propósito de este segundo fue **optimizar** el rendimiento y la eficiencia energética **de manera guiada**, reduciendo la cantidad de simulaciones necesarias.

El algoritmo Greedy se diseñó para explorar el espacio de diseño de forma **iterativa**, **eligiendo en cada paso la mejor opción local** según los resultados anteriores. Esto significa que, en cada iteración, el script modifica un solo parámetro microarquitectónico (por ejemplo, el tamaño de la L1, la asociatividad del L2 o el ancho de fetch) y vuelve a ejecutar la simulación con gem5 y McPAT.

A diferencia del método de fuerza bruta, donde todas las combinaciones posibles se prueban sin dirección, el enfoque Greedy prioriza aquellas configuraciones que **mejoran la métrica de eficiencia energética global (EDP)** respecto a la iteración anterior.

### **Integración de gem5 y McPAT**

En este experimento sí se utilizó **McPAT**, una herramienta de estimación de potencia y energía ampliamente usada en investigación en arquitectura de computadores.  
El flujo de trabajo fue el siguiente:

1. **Ejecución de simulación en gem5** con una configuración específica del procesador Cortex-A76.
2. **Generación automática de archivos de salida:** stats.txt (métricas de rendimiento) y config.json (parámetros de simulación).
3. **Conversión de resultados con el script gem5toMcPAT\_cortexA76.py**, el cual toma los dos archivos anteriores y genera un archivo XML (config.xml) compatible con McPAT.
4. **Ejecución de McPAT** sobre el archivo XML para obtener estimaciones de:
   * Energía total consumida.
   * Energía dinámica (RuntimeDynamic).
   * Pérdidas por Leakage.
   * Potencia total y parámetros derivados.
5. Cálculo de la **métrica de eficiencia energética (EDP)** mediante la fórmula:



donde el **CPI (Cycles Per Instruction)** se obtiene directamente del stats.txt generado por gem5.  
Esta métrica combina rendimiento y consumo energético, permitiendo evaluar el equilibrio entre ambos.

### **Funcionamiento del algoritmo**

El algoritmo Greedy desarrollado en Python sigue este esquema:

1. Se define un conjunto inicial de parámetros del procesador y sus posibles valores.
2. En cada iteración, se elige un parámetro y se prueba con un nuevo valor.
3. Se ejecuta gem5 con la nueva configuración.
4. Se calcula el EDP de la simulación con ayuda de McPAT.
5. Si el EDP mejora (es decir, disminuye), el nuevo valor se mantiene; si no, se descarta.
6. El proceso se repite hasta completar un número determinado de iteraciones o hasta que las mejoras sean mínimas.

Todos los resultados de cada iteración se almacenaron en el archivo **history.csv**, que sirve como registro histórico de las decisiones del algoritmo y base de datos para análisis posteriores y generación de gráficas.

Al analizar el comportamiento del **algoritmo Greedy aplicado al diseño microarquitectónico del procesador Cortex-A76**.  
A diferencia del experimento inicial de 100 simulaciones, aquí no se explora todo el espacio de diseño con fuerza bruta, sino que se aplica un enfoque **heurístico (Greedy)** que busca **ajustar un parámetro por iteración** para mejorar el rendimiento o reducir el consumo energético según los resultados previos.

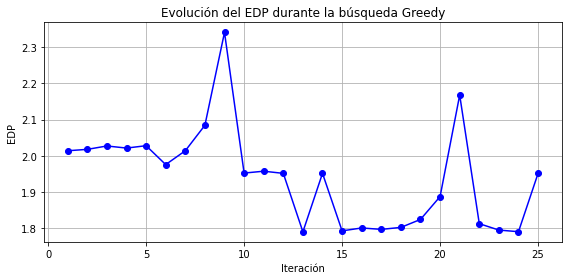
Los datos provienen del archivo history.csv, donde se registró la evolución del algoritmo:

* **Iteración:** número de paso o simulación dentro del proceso Greedy.
* **Parámetro:** elemento del procesador que se modificó (por ejemplo, tamaño de caché, ancho de fetch, tipo de predictor de saltos).
* **Valor probado:** valor concreto asignado al parámetro.
* **EDP (Energy-Delay Product):** métrica compuesta de eficiencia energética (energía × CPI).
* **Energía, Leakage, RuntimeDynamic:** métricas obtenidas de McPAT.
* **CPI:** ciclos por instrucción.
* **Configuración completa:** cadena con todos los valores de la simulación.

## Gráfica 1 — Evolución del EDP durante la búsqueda Greedy

**Objetivo:** visualizar cómo evoluciona la eficiencia energética a lo largo de las iteraciones.

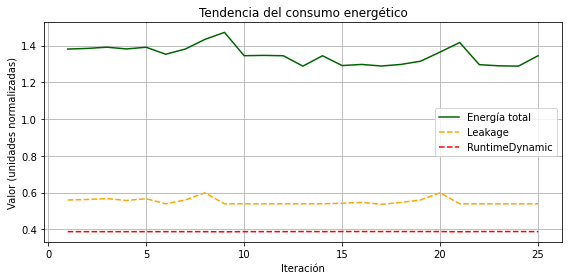
* Un **descenso en el EDP** indica que el algoritmo está encontrando configuraciones **más eficientes** (menos energía por unidad de rendimiento).
* Si el EDP se estabiliza, sugiere que el algoritmo llegó a una **zona óptima local**.



## Gráfica 2 — Tendencia del consumo energético

**Objetivo:** observar cómo cambian los componentes del consumo de energía durante las simulaciones.

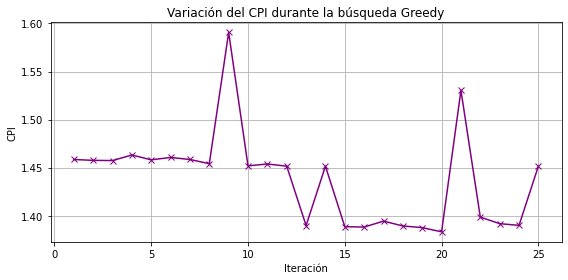
* **Energía total:** suma general del consumo.
* **Leakage:** energía estática (pérdidas por transistores).
* **RuntimeDynamic:** energía dinámica (debido a operaciones activas).



## Gráfica 3 — Variación del CPI durante la búsqueda

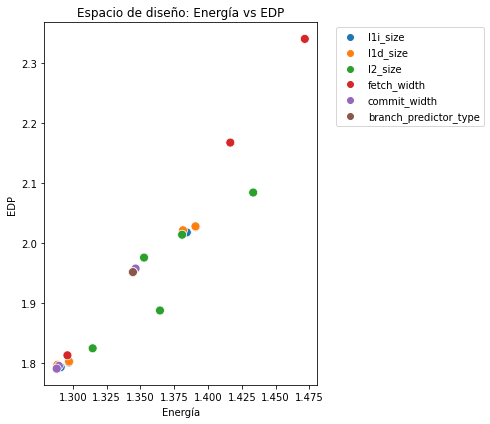
**Objetivo:** mostrar cómo cambia el rendimiento puro (CPI) a lo largo de las iteraciones.

* Un **CPI más bajo** implica **mayor rendimiento**.
* Si el CPI se incrementa, puede haber penalización por configuraciones que priorizan el ahorro energético.



## Gráfica 4 — Espacio de diseño (Energía vs EDP)

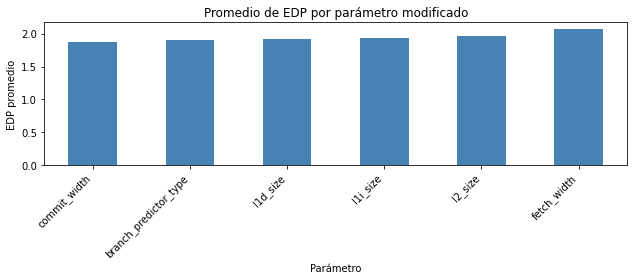
**Objetivo:** visualizar el espacio de exploración de las configuraciones y su impacto.

* Cada punto es una simulación.
* El color indica qué parámetro se modificó. 

## Gráfica 5 — Promedio de EDP por parámetro modificado

**Objetivo:** identificar qué parámetro tiene mayor impacto promedio en la eficiencia energética.

* Se agrupan las simulaciones por tipo de parámetro.
* Un valor de EDP bajo indica que ese parámetro fue especialmente influyente.



## Gráfica 6 — Frontera de Pareto (Energía vs CPI)

**Objetivo:** mostrar el compromiso (trade-off) entre energía consumida y rendimiento.

* Cada punto representa una configuración.
* El color muestra qué parámetro fue modificado.
* La “frontera” ideal son los puntos **más cercanos al origen (baja energía y bajo CPI)**.

