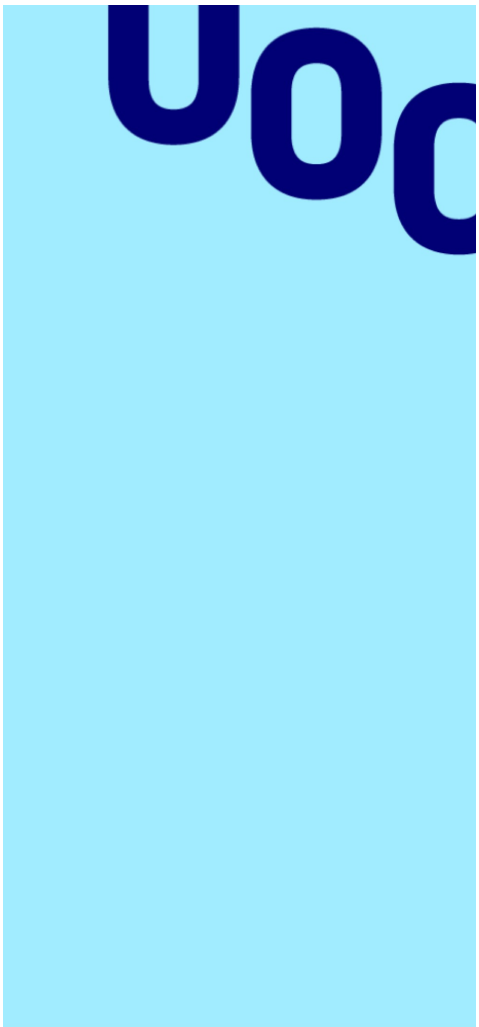
Arquitectura multiagente para la evaluación del impacto energético de Large Language Models (LLMs) en función de técnicas de prompting y longitud de contexto



Pablo Marcos Parra

Area 2: NLP and Visual Analytics

Máster en Ciencia de Datos

Nombre del tutor:
Josep-Anton Mir Tutusaus

Junio 2026

Universitat Oberta de Catalunya

Ficha Del Trabajo Final

Título del trabajo:	Arquitectura multiagente para la evaluación del impacto energético de Large Language Models (LLMs) en función de técnicas de prompting y longitud de contexto
Nombre del autor/a:	Pablo Marcos Parra
Nombre del tutor:	Josep-Anton Mir Tutusaus
Fecha de entrega:	Junio 2026
Titulación o programa:	Máster en Ciencia de Datos
Área del trabajo final:	Area 2: NLP and Visual Analytics
Idioma del trabajo:	Castellano
Palabras clave:	Large Language Models, Green AI, Prompt Engineering, Huella de carbono, Procesamiento de Lenguaje Natural, Eficiencia energética

Resumen del trabajo

El uso masivo de sistemas basados en Modelos de Lenguaje Grande (Large Language Model (LLM)) está generando una preocupación creciente debido a su alto consumo energético e impacto medioambiental, especialmente en la fase de inferencia con entradas largas o complejas. La ingeniería de prompts ha estado centrada hasta ahora en maximizar la calidad de la respuesta, ignorando el coste computacional que hay detrás.

Este proyecto propone analizar empíricamente cómo la técnica de prompting y su longitud influyen en el consumo de energía. Para ello, se desarrollará un sistema multiagente autónomo que ejecute diversas tareas relacionadas con texto tales como generación de código o resumen de textos, mida su huella de carbono mediante herramientas de telemetría y evalúe su calidad utilizando un juez sintético imparcial (LLM-as-a-judge).

Abstract

The massive use of systems based on Large Language Models (LLM) is raising growing concern due to their high energy consumption and environmental impact, especially during the inference phase with long or complex inputs. Prompt engineering has so far focused on maximizing response quality while ignoring the computational cost behind it.

This project proposes an empirical analysis of how prompting techniques and prompt length influence energy consumption. To this end, an autonomous multi-agent system will be developed to execute various text-related tasks such as code generation or text summarization, measure their carbon footprint using telemetry tools, and evaluate their quality using an unbiased synthetic judge (LLM-as-a-judge).

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Contexto y justificación del trabajo

El presente trabajo se enmarca en la necesidad creciente de adoptar prácticas tecnológicas sostenibles frente la emergencia climática. A medida que la Inteligencia Artificial Generativa (GenAI) se integra en tareas cotidianas y es adoptada por las personas alrededor de todo el mundo, la carga computacional en los centros de datos se dispara, incrementando la huella de carbono global. Según estimaciones recientes, la inferencia de modelos de IA representa más del 90 % del consumo energético total de un modelo que se encuentra en producción y una sola consulta a un LLM puede consumir aproximadamente 3 Wh, lo que supone casi diez veces más que una búsqueda web convencional [1]

Desde una perspectiva ingenieril, no existen demasiadas herramientas orientadas a medir la eficiencia energética de forma granular en la capa que gestiona la lógica de la tarea, es decir, el prompt con las instrucciones. Estudios recientes están empezando a demostrar que la elección de la técnica de prompting puede alterar el consumo energético de inferencia y que hay técnicas complejas, como *chain-of-thought*, que imponen una penalización energética mayor que otros enfoques más sencillos como *zero-shot* [2]. Sin embargo, existe una revisión sistemática realizada por Verdecchia et al. [3] sobre 98 estudios de Green AI que reveló que solo 17 de ellos se centraban en la fase de inferencia, lo que refuerza la necesidad de la línea de investigación que se pretende abordar en este trabajo.

Este proyecto es de interés para la sociedad porque aporta transparencia a los modelos de IA generativa, facilitando la identificación de ineficiencias y por tanto reduciendo los costes operativos y medioambientales de las soluciones software que se basen en esta tecnología, basando el trabajo en los principios de Green AI [4]. Además conecta directamente con la línea de investigación emergente denominada "Green Prompting" [5] que propone que el diseño de prompts es un camino viable para reducir la huella de carbono de sistemas basados en LLMs.

1.2. Motivación

La motivación principal para realizar este Trabajo Final de Máster (TFM) se debe a la intersección entre mi interés profesional como AI Data Scientist, donde trabajo en mi día a día con sistemas y soluciones basados en GenAI, y mi preocupación por el impacto medioambiental de estas tecnologías.

A un nivel aún más personal, este proyecto me parece también un reto estimulante que me permite consolidar mis conocimientos en este campo tan puntero y que en el año 2026 está en un pico de popularidad enorme.

1.3. Objetivos del trabajo

1.3.1. Objetivo principal

Diseñar e implementar un sistema multiagente que evalúe y compare empíricamente el impacto energético y la calidad de respuesta de distintas técnicas de prompting en LLMs, identificando posibles patrones de redacción de prompts sostenibles.

1.3.2. Subobjetivos

- Analizar cómo varía el consumo energético por inferencia (medido en Wh) al escalar la complejidad de un prompt usando técnicas de prompting diferentes (*zero-shot, few-shot, chain-of-thought*, entre otras).
- Comparar el comportamiento energético de LLMs diferentes ante un mismo tipo de prompting, tanto en modelos de código abierto ejecutados en local como en modelos propietarios accesibles mediante Application Programming Interface (API).
- Desarrollar un agente IA capaz de identificar el tipo de tarea a realizar y generar múltiples versiones de un mismo prompt aplicando distintas técnicas de ingeniería.
- Desarrollar un agente que genere una rúbrica específica para medir la calidad del resultado de resolución de una tarea.
- Desarrollar un módulo de ejecución de prompts que capture datos de consumo usando la herramienta CodeCarbon [6] para medir el consumo de modelos ejecutados en local y la herramienta EcoLogits [7] para medir el consumo de modelos disponibilizados mediante APIs.
- Desarrollar un agente evaluador basado en modelos de lenguaje (LLM-as-a-Judge [8]) que guiado por las rúbricas dinámicas generadas pueda puntuar numéricamente la calidad de los textos.
- Desarrollar un módulo que sintetice los resultados obtenidos para generar informes automatizados.

1.4. Impacto en sostenibilidad, ético-social y de diversidad

Dimensión de sostenibilidad

Este TFM aborda los siguientes ODS en el apartado de sostenibilidad:

- **ODS 9** (Industry, innovation and infrastructure)
- **ODS 12** (Responsible consumption and production)
- **ODS 13** (Climate action)

La sostenibilidad medioambiental es el eje central de este proyecto, ya que nace de la preocupación por el alto consumo de los LLMs en la fase de inferencia y tiene como objetivo generar evidencia que permita identificar técnicas de prompting más energéticamente eficientes (Green Prompting).

Los impactos positivos principales son la cuantificación del coste en emisiones de CO_2eq de distintas estrategias de prompting y la generación de una guía práctica para reducir la huella de carbono que permite cerrar la brecha investigadora que hay en un campo innovador y en desarrolo como es el Green AI.

Como impacto negativo, el propio proceso experimental implica la ejecución de múltiples inferencias sobre modelos de diferentes escalas, lo que genera un consumo energético. Para mitigarlo, se diseñarán experimentos sin repeticiones excesivas o ejecuciones innecesarias, reutilizando resultados intermedios y documentando en todo momento el coste ambiental.

Dimensión ético-social

Este TFM aborda los siguientes ODS en el apartado de ética:

- **ODS 8** (Decent work and economic growth)
- **ODS 16** (Peace, justice and strong institutions)

El proyecto aportará transparencia al consumo energético real de la utilización de los LLMs, un dato que suele ser ignorado por las organizaciones, por lo que se podrán tomar decisiones más responsables. Además, las herramientas utilizadas son de código abierto, por lo que los resultados son reproducibles y accesibles para todo el mundo, independientemente de su capacidad económica.

Como punto negativo a valorar, el uso de LLM-as-a-judge como evaluador de calidad podría introducir sesgos preexistentes en los modelos utilizados por lo que para mitigar esto, se verificarán que las rúbricas son transparentes y equitativas.

Dimensión de diversidad, género y derechos humanos

Este TFM aborda los siguientes ODS en el apartado de diversidad y derechos humanos:

- **ODS 10** (Reduce inequalities)

El caracter técnico del proyecto limita el impacto directo en esta dimensión. Sin embargo, la optimización del consumo energético que se plantea puede contribuir indirectamente a reducir la brecha entre personas y organizaciones con recursos, dispares, ya que reducir el consumo implicaría también menor coste por lo que el uso de estas tecnologías de GenAI sería más accesible para organizaciones con recursos limitados.

1.5. Enfoque y método seguido

Estrategia de investigación

El proyecto adoptará la metodología de *Design and Creation* que se centra en la creación y evaluación de un artefacto innovador (en este caso, un sistema multiagente basado en LLMs) diseñado para resolver un problema empírico [9]).

Se aplicarán técnicas de generación de datos cuantitativas, extrayendo métricas de calidad de la respuesta de las diferentes técnicas de prompting (del 1 al 10) y el consumo energético (vatios-hora y gCO_2eq).

Las principales herramientas a usar incluirán Python, frameworks de orquestación de IA (`LangGraph`), librerías de telemetría especializadas (`CodeCarbon`, `EcoLogits`) y librerías de análisis de datos (`Pandas`, `Matplotlib`).

Metodología de desarrollo

Para la implementación del proyecto, se utilizará un paradigma de programación ágil e iterativa. Esta metodología permitirá implementar las llamadas a los LLMs correspondientes, los módulos necesarios de Python y validar cada componente antes de integrarlo en el sistema completo.

A través de iteraciones cortas, se desarrollará primero el núcleo de la telemetría (interactuando con la infraestructura de ejecución local y las APIs remotas), posteriormente la capa de orquestación y lógica (generación de rúbricas y prompts) y finalmente la capa de análisis y evaluación.

1.6. Planificación del trabajo

El proyecto se desarrollará en el periodo comprendido entre el 1 de marzo y el 24 de mayo de 2026, organizándose las tareas de la siguiente manera:

1. **Fase 1: Estado del arte (1 marzo - 22 marzo)**

   Revisión bibliográfica (metodologías, Green AI, LLM-as-a-Judge, LLMs, arquitecturas multiagentes. Selección definitiva de frameworks y herramientas.

2. **Fase 2: Diseño e implementación (23 marzo - 10 mayo)**

   2.1 Diseño de la arquitectura multiagente (1 semana)

   2.2 Desarrollo del módulo Python de ejecución de prompts y medición (1 semana). Depende de 2.1

   2.3 Desarrollo del agente de identificación de tareas y creación de prompts basados en técnicas de prompting (1 semana). Depende de 2.1

   2.4 Desarrollo del agente de generación de rúbricas (1 semana). Depende de 2.1

   2.5 Desarrollo del agente evaluador (1 semana). Depende de 2.2, 2.3 y 2.4

   2.6 Desarrollo del módulo Python de análisis de resultados (1 semana). Depende de 2.5

   2.7 Realización de pruebas de integración (1 semana). Depende de 2.6

3. **Fase 3: Redacción preliminar de la memoria (11 mayo-17 mayo)**

   Estructuración del documento de TFM, volcado del marco teórico, descripción metodológica e introducción de datos y resultados del desarrollo y pruebas.

   Dependencia: fase 2.

4. **Fase 4: Redacción final de la memoria (18 mayo - 24 mayo)**

   Discusión de resultados, elaboración de conclusiones, revisión de formato y bibliografía y entrega de la memoria del TFM.

   Dependencia: fase 3.

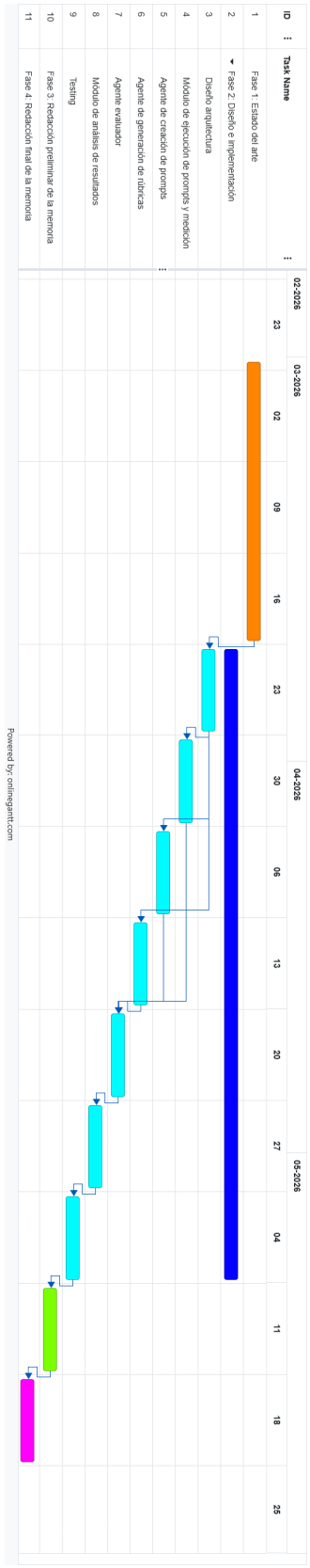


Figura 1.1: Diagrama de Gantt con la planificación

1.7. Breve sumario de productos obtenidos

A continuación se enumeran los productos generados durante el desarrollo de este TFM:

- **Memoria del trabajo**: el presente documento, que recoge el marco teórico, la metodología usada y los resultados obtenidos.
- **Sistema multiagente de medición**: implementación completa en Python (`src/`) de la arquitectura multiagente basada en LangGraph que se describe en el capítulo 3, incluyendo los agentes descritos en los objetivos `Router`, `Rubric Generator`, `Judge` principal y `Validation Judge`, así como los wrappers de invocación para los cinco modelos evaluados (`GPT, Google, Mistral, Qwen y LLaMa`).
- **Módulo de telemetría energética**: capa de medición que integra las herramientas `CodeCarbon`, NVML (`pynvml`) y `Ecologits`. Estas herramientas miden o estiman el consumo energético tanto de los modelos propietarios como de los modelos abiertos de ejecución local.
- **Pipeline de preprocesamiento**: se adjuntan scripts deterministas (`scripts/01_select_instances.py`, `scripts/02_generate_prompts.py`, `scripts/03_generate_rubrics.py`) que materializan y persisten en memoria (`artifacts/`) las instancias, prompts y rúbricas usadas.
- **Artefactos experimentales**: los resultados del preprocesamiento, donde se han persistido 15 instancias (5 por tarea medida), 180 variantes de prompt (12 por instancia), 3 rúbricas calibradas con humano y 9 ejemplos few-shot revisadas manualmente.
- **Base de datos SQLite** con las 2700 inferencias evaluadas y sus correspondientes datos asociados, incluyendo puntuaciones del juez principal y del juez de validación y el consumo del meta-sistema y de las inferencias (ver apéndice A)
- **Pipeline de análisis estadístico** (`scripts/07_analyze_results.py`) con los modelos usados incluyendo modelos lineales mixtos, tests de Wilcoxon, tamaños de efectos de Cohen y correlaciones de Spearman y Kappa.
- **Visualizaciones** generadas automáticamente con Matplotlib: boxplots de consumo y calidad, heatmap de consumo por cada modelo y técnica y gráfico de consumo del meta-sistema.
- **Repositorio público en GitHub** (`https://github.com/lepablito/TFM_UOC_PabloMarcosParra`) con el código, artefactos, documentación adicional y base de datos.

1.8. Breve descripción de los otros capítulos de la memoria

Esta memoria se estructura en seis capítulos y dos apéndices:

- **Capítulo 2 - Estado del arte**: revisión bibliográfica del estado actual de la temática a trabajar, incluyendo Green AI, consumo energético de LLMs y el paradigma LLM-as-a-Judge.
- **Capítulo 3 - Materiales y métodos**: descripción del diseño factorial mixto usado (3 técnicas de prompting $\times$ 2 longitudes de prompt $\times$ 5 modelos LLM $\times$ 3 tipos de tarea $\times$ 5 instancias por tarea $\times$ 3 repeticiones $=$ 2 700 inferencias), datasets usados (XSum, HumanEval, GSM8K), arquitectura multiagente basada en LangGraph, protocolo de evaluación del LLM-as-a-Judge con calibración humano-juez, validación inter-juez, telemetría y plan de análisis estadístico.
- **Capítulo 4 - Resultados**: presentación de los hallazgos empíricos: cobertura del experimento, modelo lineal mixto sobre la energía y sobre la puntuación, contrastes pareados de Wilcoxon, tamaños de efectos de Cohen, validación del juez sintético contra un juez de validación y contra métricas objetivas, análisis del sesgo de verbosidad, latencia y throughput de los modelos y consumo del meta-sistema.
- **Capítulo 5 - Valoración económica**: desglose de los costes de las APIs comerciales usadas, el consumo energético y el coste en hardware.
- **Capítulo 6 - Conclusiones y trabajos futuros**: síntesis de los hallazgos principales, incluyendo valoración de los objetivos y propuesta de líneas de trabajo futuras.
- **Apéndice A - Base de datos SQLite**: diagrama y esquema completo de las cinco tablas que persisten el experimento.
- **Apéndice B - Resultados del modelo de efectos mixtos**: tablas con los coeficientes completos de los cuatro modelos lineales mixtos ajustados.

1.9. Declaración sobre el uso de inteligencia artificial generativa

En este TFM se ha usado Inteligencia Artificial generativa como apoyo en las fases de diseño, implementación y experimentación.

Fase de diseño

Partiendo de unos esqueletos iniciales (ficheros Markdown) definidos por el autor con el diseño del proyecto, donde se han definido cada una de las partes y componentes a implementar del proyecto, se ha usado **Claude Code** (modelo Opus 4.7) para iterar sobre estos documentos de diseño para refinarlos e ir solucionando problemas metodológicos o de diseño que fueran apareciendo durante el desarrollo e implementación. Dichas iteraciones se han realizado bajo la supervisión y aprobación del autor, validando siempre su concordancia con los objetivos planteados. Los resultados de esta fase de diseño se encuentran documentados en la carpeta `docs/`

del proyecto, donde se incluyen además anotaciones operacionales que se han tomado durante la ejecución de los experimentos.

Fase de implementación

En esta fase se ha usado **Claude Code** (modelo Opus 4.7) y **OpenCode** (modelo local Qwen 2.5 Coder 14B) como asistentes de programación. Han servido de apoyo en la escritura, depuración y refactorización del código Python, acelerando la implementación de *boilerplate* de código.

Claude Code ha sido usado durante la implementación del código principal del proyecto (`src/`) mientras que OpenCode ha sido usado durante la implementación de los scripts de ejecución (`scripts/`).

Ambas herramientas han estados bajo supervisión contínua y cada función generada se ha revisado, probado e integrado manualmente por el autor en el repositorio.

Fase de experimentación

Los resultados experimentales no se han generado ni alterado mediante IA y su interpretación son obra del autor.

Sin embargo, y debido a la propia naturaleza del proyecto y a la definición metodológica realizada en el capítulo 3), se ha usado **Claude Sonnet 4.6 y Claude Haiku 4.5** para la generación de algunos de los artefactos experimentales usados, tales como la generación de ejemplos *few-shot* y *Chain-of-Thought (CoT)* y la generación de rúbricas.

Capítulo 2

Estado del arte

En este capítulo se hará la revisión bibliográfica de las áreas de conocimiento que se utilizan en este TFM. El objetivo es fundamentar con evidencias científicas sólidas y probadas la línea de investigación que se aborda e identificar las brechas que justifican la existencia de este proyecto. El estado del arte se organiza en torno a seis ejes temáticos que se encuentran estrechamente relacionados.

Se partirá de los fundamentos teóricos y del contexto medioambiental que motivan el proyecto, se profundizará en las tecnologías en las que se construye el sistema (modelos, técnicas de prompting, herramientas de medición) y se concluirá con los paradigmas de evaluación y orquestación de la arqutectura multiagente propuesta.

2.1. Green AI y huella de carbono de la IA

La preocupación por el impacto medioambiental de la IA apareción en el entorno académico cuando Strubell et al. [10] cuantificaron por primera vez el coste energético del entrenamiento de modelos de Natural Language Processing (NLP). Este estudio reveló que el entrenamiento de un modelo basado en la arquitectura Transformer podía emitir una cantidad elevada de CO_2, lo que situó la eficiencia energética como una cuestión importante a tratar en el mundo de la investigación de estos modelos.

El concepto Green AI fue formalizado por Schwartz et al. [4], quienes distinguieron entre dos paradigmas contrapuestos: la denominada *Red AI*, orientada a maximizar la precisión y calidad de las respuestas del modelo sin considerar el coste computacional, y el propio paradigma *Green AI*, que propone tratar la eficiencia de los modelos como un criterio de evaluación comparable al rendimiento. Los autores abogan por generar reportes sistemáticos del coste computacional de los experimentos realizados con estos modelos para fomentar la investigación responsable y sostenible.

Cuantificación de la huella de carbono en modelos de gran escala

A medida que los modelos basados en arquitectura Transformer fueron creciendo en tamaño, diversos estudios intentaron cuantificar su huella de carbono real. Patterson et al. [11] calcularon la energía consumida y las emisiones de carbono que se generaron durante el entrenamiento de modelos como `T5` o `GPT-3` entre otros, demostrando que la combinación de la arquitectura, la localización del centro de datos usado y el tipo de procesador en esta fase de desarrollo del modelo podían afectar a la huella de carbono, llegando a tener un impacto variable de entre 100 y 1 000 veces según los parámetros.

Luccioni et al. [12] llevaron a cabo el primer análisis completo del ciclo de vida de carbono de un LLM completo, incluyendo entrenamiento e inferencia. El modelo elegido fue `BLOOM`, un modelo de 176 mil millones de parámetros. Estimaron unas emisiones totales de 50,5 toneladas de CO_2eq, con 24,7 toneladas de CO_2eq del total que fueron emitidas durante el entrenamiento. Durante la fase de despliegue e inferencia, estimaron emisiones de 19,1 kilogramos de CO_2eq por cada día de exposición de la API de inferencia.

Coste energético de la inferencia

Los estudios iniciales parecían centrarse en la fase de entrenamiento, sin embargo, algunos estudios posteriores han evidenciado que la fase de inferencia representa la mayor parte del consumo energético total cuando se despliega en producción un modelo de estas características. De Vries [1] estimó ese consumo y proyectó un escenario donde servidores y centros de datos de IA podrían demandar entre 85 y 134 TWh anuales para el año 2027.

En otro trabajo de Luccioni et al. [13], se aportó la primera comparación sistemática del coste energético de la inferencia entre distintos tipos de modelos, concluyendo que los modelos generativos de propósito general consumen más energía que los sistemas especializados en una tarea concreta. Esta disparidad entre sistemas, justifica la necesidad de evaluar no solo la calidad de las respuestas sino también su eficiencia energética, que es el objetivo central de este TFM.

Sin embargo, el foco del estudio del consumo sigue siendo la fase de entrenamiento, como también se ha destacado en el apartado de justificación y contexto. Verdecchia et al. [3] señaló esta brecha crítica entre los estudios que verificaban el coste del entrenamiento del modelo y los estudios que estudiaban el coste de la inferencia, siendo estos últimos los más escasos.

2.2. Arquitectura Transformer y evolución de los LLMs

La arquitectura Transformer, propuesta en el paper *'Attention is all you need'* [14], constituyó un cambio de paradigma en el campo del NLP. Frente a las redes neuronales recurrentes (LSTM, GRU, RNN) que procesaban los textos de forma secuencial, lo que limitaba mucho el volumen de datos usados en entrenamiento y la calidad de la respuesta posterior. La arquitectura Transformer presentó el mecanismo de autoatención (*self-attention mechanism*), permitiendo

capturar dependencias en el contexto a largo plazo de manera paralela. Esta innovación mejoró la calidad de las representaciones lingüísticas de las palabras a nivel computacional y aceleró mucho el entrenamiento al aprovechar el paralelismo que ofrecen las Unidades de Procesamiento de Gráficos (GPU) modernas.

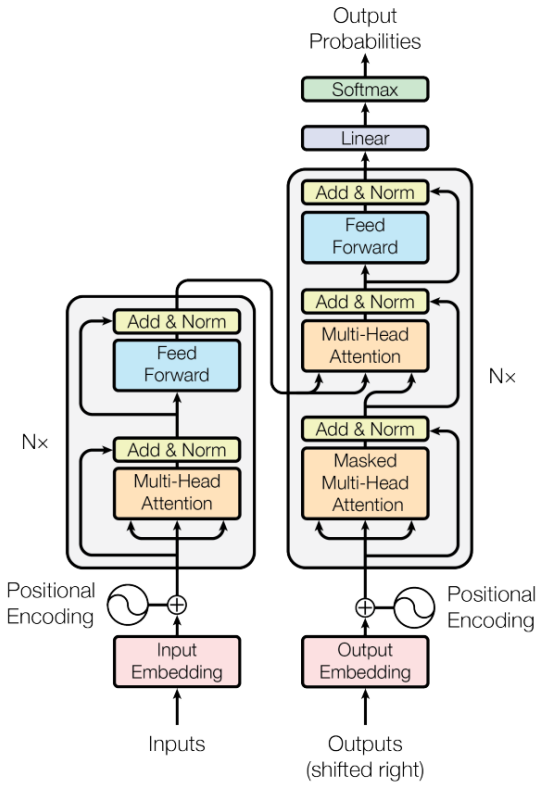


Figura 2.1: Arquitectura Transformer

La arquitectura se compone de una estructura codificador/decodificador, cada uno formado por capas de atención *multihead* y redes feed-forward. A partir de la estructura Transformer se derivaron dos familias de modelos:

- **Modelos basados en el codificador**: modelos orientados a tareas de comprensión del lenguaje, como `BERT` [15].
- **Modelos basados en el decodificador**: modelos orientados a la generación de texto, como los modelos `GPT` [16]

Los modelos basados en el decodificador son los que tienen mayor relevancia para este proyecto, ya que las técnicas de prompting se aplican sobre modelos generativos de este tipo.

Leyes de escalado y eficiencia

El escalado de los LLMs en cuanto a tamaño y capacidades no es arbitrario, sino que sigue una relación empírica predecible. Kaplan et al. [17] establecieron las primeras leyes de escalado demostrando que el rendimiento de un modelo de lenguaje sigue una relación de ley potencial respecto al número de parámetros, el tamaño del conjunto de datos y el presupuesto dedicado al cómputo. De estas relaciones se puede asumir que existe una asignación óptima en cuanto el tamaño del modelo y el volumen de datos usados en el entrenamiento, sin incurrir en ineficiencias económicas y computacionales.

Este primer hallazgo fue luego refinado en el artículo de presentación del modelo conocido como `Chinchilla` [18], donde se demostraba que los modelos existentes estaban subentrenados. Es decir, para un presupuesto de cómputo dado, el rendimiento óptimo se alcanza escalando de forma igual el número de parámetros y el número de tokens de entrenamiento. El modelo `Chinchilla` (70B parámetros) superó al modelo `Gopher` (280B parámetros) usando un mismo presupuesto. Esto implica que modelos más pequeños pero mejor entrenados pueden ofrecer rendimiento equivalente con menor coste de inferencia.

Arquitectura densa vs. Mixture of Experts (MoE)

Los modelos inicialmente creados se basaban en una arquitectura densa, donde todos los parámetros del modelo se activan en cada paso de inferencia, lo cual es bastante costoso a nivel computacional. Fedus et al. [19] presentaron Switch Transformers, donde se define la arquitectura *MoE* donde se selecciona el camino de parámetros más adecuado siguiendo la selección de un camino experto por token. Esta nueva arquitectura permite escalar modelos grandes con un coste computacional equivalente al de modelos mucho más pequeños logrando acelerar el entrenamiento y la inferencia.

Jiang et al.[20] introdujeron el modelo `Mixtral 8x7B`, un modelo MoE disperso que accede a 47B de parámetros pero solo usa 13B activos por token. `Mixtral` igualó en rendimiento al modelo de referencia de aquel entonces que era `GPT-3.5` con cinco veces menos parámetros activos, demostrando así que esta arquitectura MoE son relevantes para mejorar la relación rendimiento/coste en la fase de inferencia. Esta distinción de arquitecturas de modelos constituye por tanto una posible variable en el diseño que en el proyecto actual se evalúa.

Modelos de código abierto vs. modelos propietarios

El ecosistema actual de LLMs se divide entre modelos propietarios accesibles mediante API y modelos de código abierto ejecutables en infraestructura local.

Esta distinción tiene implicaciones directas para la medición energética. Mientras que en los modelos locales se puede medir directamente el consumo a partir de los componentes hardware del entorno local, los modelos propietarios requieren de estimaciones indirectas basadas en el número de tokens usados.

Modelos propietarios

En el ámbito propietario, el informe técnico de `GPT-4` (OpenAI)[21] describió un modelo multimodal que alcanza un nivel humano en ciertas tareas y benchmarks profesionales, aunque sin revelar detalles técnicos.

A fecha 22 de marzo de 2026, estos son los modelos ofrecidos por los principales proveedores de modelos propietarios:

- **GPT-5.4 (OpenAI)**[22]
- **Claude Sonnet 4.6, Opus 4.6, Haiku 4.5 (Anthropic)**[23]
- **Gemini 3 y Gemini 2.5 (Google)**[24]

Modelos de código abierto

En el ámbito abierto, Touvron et al.[25] demostraron con `LLaMa` que modelos entrenados exclusivamente con datos públicos pueden competir con modelos propietarios mayores, ya que por ejemplo `LLaMa 13B` superaba a `GPT-3` (175B parámetros) en muchos de los benchmarks evaluados.

Para este proyecto cabe considerar los siguientes modelos, clasificándolos por su escala y tamaño de parámetros:

- **Modelos de escala pequeña (1B-8B de parámetros)**: son modelos ligeros, frecuentes en despliegues con recursos limitados. Algunos de los modelos considerados son `LLaMa 3.2 1B/3B`, `LLaMa 3.1 8B`, `Mistral 7B` o `Qwen 2.5 7B`.
- **Modelos de escala media (14B-32B de parámetros)**: modelos que representan un punto intermedio en la curva rendimiento-eficiencia. Algunos de los modelos considerados son `Phi-4 14B`, `Mistral Small 3.1 24B` o `Qwen 2.5 14B/32B`.
- **Modelos de escala grande (más de 70B de parámetros)**: modelos de gran capacidad. Aquí se consideran `LLaMa 3.3 70B` o `Qwen 2.5 72B`.

2.3. Ingeniería de prompts y coste computacional

Técnicas fundamentales de prompting

La ingeniería de prompts o *prompt engineering* se ha considerado como el principal mecanismo de interacción y manejo de resultados de los LLMs sin necesidad de modificar los pesos del modelo o realizar procesos de afinado. Brown et al.[16] demostraron con `GPT-3` que los modelos grandes pueden realizar tareas complejas mediante aprendizaje en contexto (*in-context learning*), presentando 3 paradigmas de prompting fundamentales:

- **Zero-shot**: donde el modelo recibe únicamente la instrucción de la tarea.
- **One-shot**: donde se proporciona un único ejemplo resuelto.
- **Few-shots**: donde se incluyen entre dos y cinco ejemplos como contexto previo.

Más adelante se introdujo la técnica **CoT**[26], en la que se solicita al modelo que genere razonamientos intermedios paso a paso antes de emitir una respuesta final. Este enfoque mejoró drásticamente el rendimiento de los modelos en tareas de razonamiento lógico y aritmético. Sin embargo, a nivel energético, CoT implica una generación mayor de tokens de salida, lo que incrementa el consumo de inferencia según el número de pasos de razonamiento generados.

Posteriormente, Kojima et al. [27] demostraron que una simple extensión del prompt básico con la frase *'Let's think step by step'*, lo que se denomina **zero-shot-CoT**, permite activar capacidades de razonamiento multipaso sin necesidad de proporcionar ejemplos. En su estudio, la precisión en el benchmark `MultiArith` pasó del 17,7 % al 78,8 % con esa modificación mínima del prompt. Esto es relevante en el proyecto actual, ya que ilustra cómo una pequeña alteración en la formulación del prompt puede desencadenar un mayor o menor consumo energético.

Impacto energético de las técnicas de prompting

La relación entre la técnica de prompting empleada y el consumo energético ha comenzado a estudiarse recientemente. Wilhelm et al. [2], como se comentó anteriormente, demostraron empíricamente esta relación. Demostraron que técnicas como CoT imponen penalizaciones energéticas mayores que enfoques más simples. En el estudio se plantea la adopción de una métrica de energía por token como criterio de evaluación en los benchmark de LLMs.

Rubei et al. [28] llevaron a cabo un estudio empírico centrado específicamente en el impacto de la ingeniería de prompts sobre el consumo energético, demonstrando que el uso de prompts estructurados con etiquetas como XML pueden reducir el consumo hasta en un 99 % en configuraciones *one-shot* y un 83 % en configuraciones *few-shot* sin comprometer la calidad de la respuesta generada. Esto es particularmente relevante en este proyecto ya que nos ofrece nuevas posibilidades de prompting, incluyendo XML, JSON o técnicas más modernas y enfocadas a modelos de lenguaje como Token-Oriented Object Notation (TOON) [29].

2.4. Herramientas de medición energética para Inteligencia Artificial

Estado actual de las herramientas de medición disponibles

La medición del consumo energético y las emisiones de carbono de los sistemas de aprendizaje automático e IA ha dado lugar a un ecosistema amplio de herramientas de software especializadas. Jay et al.[30] realizaron una comparación empírica de nueve de estas herramientas, entre ellas: `CodeCarbon`, `CarbonTracker`, `Experiment-Impact-Tracker`, `Green-Algorithms`

y `MLCO2`; validando cada una de ellas contra mediciones obtenidas mediante un vatímetro físico conectado a los equipos. Demostró que las estimaciones mediante software pueden desviarse respecto a las mediciones físicas, lo que constituye en este TFM una consideración metodológica importante a la hora de medir la eficiencia energética.

Henderson et al.[31] introdujeron `experiment-impact-tracker` (usado por Jay et al. en su estudio) que era un marco de código abierto para el seguimiento en tiempo real del consumo energético y las emisiones de carbono. Los autores propusieron además la inclusión de las métricas generadas en las publicaciones científicas y demostraron que la región del proveedor de servicios en la nube para trabajar con el modelo puede afectar también a las emisiones. Anthony et al. [32] presentaron `CarbonTracker`, otra herramienta de código abierto que monitoriza los primeros epochs del entrenamiento de los modelos para extrapolar el consumo total, usando para ello los drivers nativos de Intel (Running Average Power Limit (RAPL)) y NVIDIA (NVML).

CodeCarbon (medición en ejecución local)

`CodeCarbon` es la herramienta de referencia en la comunidad de Green AI para la medición de la huella de carbono [33]. Desarrollada por MILA, BCG GAMMA, Haverford College y CometML bajo licencia MIT, su función principal es medir el consumo eléctrico de Central Processing Unit (CPU), GPU y Random Access Memory (RAM) durante la ejecución de código Python, convirtiendo la energía consumida en CO_2 equivalentes.

La medición de potencia sigue una estrategia jerárquica en la que se intenta usar métodos alternativos en caso de que los principales fallen. Intenta primero acceder a los contadores Intel RAPL, que es el método más preciso disponible vía software. Si RAPL no está disponible, recurre a una base de datos interna con el consumo promedio de más de 2000 modelos de procesadores. Para GPUs, la medición se realiza mediante NVIDIA NVML, lo que hace que solo sean compatibles las tarjetas gráficas de marca NVIDIA. `Code Carbon` ha sido usado ampliamente en estudios de referencia como el mencionado de Rubei et al. [28]. Sin embargo, presenta una limitación importante de cara al presente proyecto: no puede medir el consumo de llamadas a APIs en la nube ya que la computación se realiza en servidores remotos.

EcoLogits (medición de inferencia vía API)

`EcoLogits` [7] es una librería de código abierto (licencia MPL-2.0) desarrollada por GenAI Impact. Esta librería cubre precisamente la brecha que `CodeCarbon` no logra cubrir: estimación del impacto medioambientalde las llamadas vía API de LLMs comerciales como los mencionados previamente de OpenAI, Anthropic o Google, entre otros.

Ecologits modela la energía de GPU por token mediante regresión lineal ajustada a los benchmarcks de Hugging Face LLM Perf Leaderboard. La herramienta reporta cuatro métricas: energía (kWh), emisiones de carbono ($kgCO_2eq$), potencial de agotamiento abiótico ($kgSb_{eq}$ y energía primaria (MJ), distinguiendo entre la fase de uso e inferencia y el carbono correspondiente

a la fabricación del hardware.

2.5. Paradigma LLM-as-a-judge frente a benchmarking tradicional

Benchmarks tradionales y sus limitaciones

La evaluación del rendimiento de los LLMs de ha apoyado históricamente en benchmarks basados en pruebas estandarizadas. Hendrycks et al. [34] introdujeron MMLU (*Massive Multitask Language Understanding*), un benchmark que cubre 57 áreas temáticas y que es uno de los tests de referencia para medir el conocimiento y la capacidad de razonamiento de los modelos. Liang et al. [35] propusieron HELM (*Holistic Evaluation of Language Models*), un marco de evaluación que extendió la perspectiva de evaluación a siete métricas: precisión, calibración, robustez, equidad, toxidad, eficiencia y sesgo.

Sin embargo, estos benchmarks presentan limitaciones importantes a la hora de evaluar la calidad de generación de texto abierto. Hay métricas automáticas clásicas como BLEU o ROUGE [36] que miden la coincidencia léxica con textos humanos de referencia, pero que no son capaces de medir la coherencia semántica, el contexto o la creatividad. Aparecieron métricas que parecían solucionar esto como METEOR, pero aún así continuó la búsqueda de métodos de evaluación más sofisticados, entre los que destaca el que se usará en este TFM: LLM-as-a-Judge.

Paradigma LLM-as-a-Judge

Zheng et al. [8] fueron los que formalizaron el paradigma LLM-as-a-Judge demostrando que modelos como `GPT-4` pueden actuar como evaluadores automatizados de la calidad de respuestas generadas por otros modelos de lenguaje. Mediante el uso del benchmark MT-Bench y la plataforma Chatbot Arena, los autores demostraron que las puntuaciones asignadas por el juez `GPT-4` tenía una alta concordancia con las preferencias de los evaluadores humanos. Chiang et al. [37] ampliaron luego este análisis con un estudio estadístico basado en más de 240 000 votos de preferencia humana.

Liu et al.[38] introdujeron G-Eval, un marco metodológico que usa `GPT-4` con un prompting de tipo CoT y un paradigma de rellenado de formularios para evaluar la calidad del texto generado. El enfoque de G-Eval alcanzó una correlación de 0.514 en juicios humanos en tareas de sumarización, superando a las métricas clásicas BLEU y ROUGE.

Evaluación basada en rúbricas y el modelo Prometheus

Uno de los estudios que resulta especialmente relevante para el marco de este proyecto es el estudio de Kim et al. [39] quienes presentaron `Prometheus`, un modelo de código abierto de 13B de parámetros entrenado con 1 000 rúbricas granulares y 100 000 respuestas evaluadas. Este modelo alcanzó una correlación de 0.897 con las respuestas dadas con los evaluadores humanos

cuando se le daban rúbricas personalizadas, demostrando que la evaluación basada en rúbricas dinámicas (el enfoque dado en este TFM) puede lograr una calidad de evaluación cercana a las preferencias humanas.

Algunos estudios recientes han analizado la fiabilidad de esos sistemas. Li et al. [40] ofrecieron un análisis exhaustivo del paradigma LLM-as-a-Judge desde varias perspectivas: funcionalidad, metodología, aplicaciones, metaevaluación y limitaciones. Gu et al. [41] se centraron más específicamente en el estudio de los sesgos que podrían afectar a estos jueces sintéticos, como el sesgo de verbosidad (preferencia por respuestas largas) o el sesgo de autopreferencia (puntuar más favorablemente respuestas generadas por el propio modelo) indicando la necesidad de mitigar estos sesgos.

2.6. Sistemas multiagente basados en LLMs

Los agentes autónomos basados en LLMs representan una forma de trabajo en auge en el año 2026. Es un paradigma en el que los modelos de lenguaje ya no solo generan texto, sino que planifican, razonan y ejecutan acciones de forma iterativa. Yao et al. [42] plantearon el marco metodológico `ReAct` (*Reasoning and Acting*), que intercala trazas de razonamiento del modelo con acciones específicas de la tarea a realizar, permitiendo a los LLMs crear planes dinámicos y adaptables e interactuar con herramientas externas que añaden nuevas capacidades. La mayoría de agentes modernos se basan en este marco, incluído el framework LangGraph.

Wang et al. [43] ofrecieron una revisión del campo de estudio, proponiendo un marco estándar y unificado para la construcción de estos agentes, que comprende tres módulos:

- **Perfil**: definición del rol y competencias del agente.
- **Memoria**: almacenamiento de experiencias previas e información del contexto.
- **Planificación**: descomposición de tareas y selección de acciones.

Qu et al. [44] añaden a esta perspectiva una revisión de la cuestión del aprendizaje de herramientas, donde se cubrirían las cuatro fases del flujo de trabajo:

1. **Planificación de la tarea.**
2. **Selección de la herramienta.**
3. **Invocación de la herramienta.**
4. **Generación de la respuesta.**

Arquitectura multiagente

La extensión de la arquitectura de un solo agente a un sistema multiagente introduce algunas necesidades de diseño relacionadas con la coordinación y la comunicación de los agentes. Guo et al. [45] presentaron una revisión centrada en los sistemas multiagentes basados en LLMs, cubriendo aspectos como el perfilado, los métodos de comunicación y los mecanismos de control de capacidades y herramientas.

Masterman et al. [46] analizaron arquitecturas emergentes en los agentes IA, señalando patrones de liderazgo vertical (jerarquía donde un agente padre/coordinador dirige a los demás) y horizontal (donde los agentes colaboran de forma descentralizada).

Herramientas de orquestación

Existen múltiples frameworks que permiten construir la arquitectura multiagente, orquestando el funcionamiento de estos agentes:

1. **LangGraph**[47]: ofrece una orquestación basada en grafos dirigidos (que pueden ser cíclicos). Los nodos de este grafo representan a los agentes o funciones que se deben realizar y las aristas definen el flujo de datos. Hay un controlador centralizado que controla el contexto global del flujo. Permite también la integración de interacción con humanos (*human-in-the-loop*).

2. **AutoGen**[48]: desarrollado por Microsoft, introduce el concepto de 'agentes conversables' con dos subcategorías principales: `AssistantAgent` (agente potenciado por LLM) y `UserProxyAgent` (*human-in-the-loop* o ejecución de código estático). Soporta flujos estáticos y dinámicos y actualmente usa una arquitectura asíncrona basada en eventos.

3. **MetaGPT**[49]: se usa para procesos en el ciclo del desarrollo del sofware, donde cada agente tiene un rol (Product Manager, Architect, QA...). Los agentes producen entregables (documentación técnica, diseños, código...) en lugar de conversación libre.

4. **CrewAI**[50]: ofrece una abstracción de los agentes basada en cuatro bloques: `Agent`, `Task`, `Tool` y `Crew` (*human-in-the-loop*). Soporta orquestación de agentes secuencial y jerárquica (agentes padre-hijo).

Para este proyecto, los frameworks que más encajan con el diseño establecido serían LangGraph y AutoGen, y si bien cierto que este último ofrece agentes modulares y flexibilidad en el patrón de comunicación entre agentes, se ha seleccionado LangGraph ya que es un framework gratuito, sencillo y que también cubre todas las necesidades del esquema planteado.

Capítulo 3

Materiales y métodos

Este capítulo describe el diseño experimental y metodológico que se ha empleado para cuantificar empíricamente el impacto energético de tres técnicas de prompting (*Zero-Shot, Few-Shot y Chain-of-Thought*[26] sobre cinco modelos de lenguaje distintos, tres tareas representativas de generación de texto (resumen abstractivo, generación de código y razonamiento matemático) y dos factores de presentación de código (longitud del prompt de input y formato sintáctico del mismo). Este capítulo incluye el diseño experimental, los datasets y procedimientos de selección de instancias realizados, así como una descripción de los modelos evaluados, la arquitectura del sistema multiagente que orquesta el experimento y el protocolo de evaluación de calidad de respuesta mediante un juez sintético basado en rúbricas (*LLM-as-a-Judge* [8]). También se comentará el aparato de telemetría energética y el plan de análisis estadístico planteado.

3.1. Diseño experimental

El estudio adopta un **diseño factorial mixto**, donde hay tres factores manipulados *within-subjects* (técnica de prompting, longitud de input y formato del prompt) y dos factores de bloqueo *between-subjects* (modelo y tarea, factores). Los tres factores *within* aseguran que cada instancia del experimento recibe todas las combinaciones de tratamiento posibles, lo que permite obtener resultados estadísticos más relevantes a la hora de detectar efectos diferenciales al reducir la varianza entre instancias.[51]

Los factores manipulados son los siguientes:

- **Técnica**: niveles *Zero-shot, Few-shot, Chain-of-Thought*.
- **Longitud**: por cada tarea, input *largo* o *corto*.
- **Formato**: texto natural vs. etiquetas, es decir, formato *plano* vs. *estructurado mediante XML*

Por tanto se producen $3 \times 2 \times 2 = 12$ celdas experimentales por cada combinación de modelo, tarea e instancia.

En cuanto a los factores de bloqueo se han fijado cinco modelos y tres tareas. Además se han fijado un número de instancias por tarea y de repeticiones que operan como factores aleatorios anidados:

- **Modelo**: 5 modelos de lenguaje que se detallarán más adelante
- **Tarea**: XSum (resumen de textos), HumanEval (generación de código), GSM8K (razonamiento)
- **Instancia**: 5 instancias, ejemplos de tarea, por cada tarea.
- **Repetición**: 3 por cada celda experimental.

Por tanto, el **volumen experimental** final sería de $12 \times 5 \times 3 \times 5 \times 3 = \mathbf{2\ 700}$ inferencias evaluadas. A este volumen posteriormente se añaden también las invocaciones del agente Router para pre-procesado de prompts, del agente Generador de Rúbricas, las 2 700 evaluaciones del Juez principal y las evaluaciones del juez de validación.

Para garantizar la reproducibilidad del experimento se ha fijado una semilla global (`SEED = 42`) que se utiliza a lo largo de toda el pipeline. También se ha fijado a cero la temperatura de los modelos para reducir la variabilidad de la respuesta, pero al estar trabajando con modelos de lenguaje es muy probable que las respuestas al prompting inicial varíen.

3.2. Datasets y selección de instancias

El estudio se ha apoyado en tres datasets muy conocidos y utilizados en la literatura de evaluación y benchmarking de LLMs. Cada dataset seleccionado cubre un aspecto cognitivo de los modelos. Además para poder contrastar los resultados del *LLM-as-a-Judge* se han usado algunas métricas clásicas en NLP para verificar la calidad de respuesta del juez:

- **XSUM**[52]: resumen de un solo párrafo. Es una tarea generativa de comprensión textual. En este caso como métricas de contraste contra el juez se ha usado `ROUGE-L` [36] (solapamiento léxico mediante subsecuencia común más larga) y `BERTScore` [53] (similaridad semática usando embeddings contextuales).
- **HumanEval**[54]: generación de código Python a partir de *docstrings* con tests unitarios canónicos asociados a cada instancia. La métrica usada en este caso es *pass@1* (descrito también en el paper original de HumanEval) ejecutando los tests unitarios aislados con timeout.
- **GSM8K**[55]: mide el razonamiento matemático paso a paso de aritmética escolar. En este caso la métrica de control ha sido únicamente el contraste entre la respuesta canónica y el último número emitido por el modelo (recogido e indentificado mediante expresiones regulares).

Selección de instancias

De cada dataset se han seleccionado cinco instancias mediante el script `scripts/01_select_instances.py`, usando la seed previamente señalada. La lógica de selección se encuentra en `src/data/instance_selection.py`, donde se aplica un muestreo estratificado para la selección sobre los buckets definidos:

- **XSUM**. Se ha aplicado un filtro previo de inputs de entre 200 y 600 palabras para asegurar que la versión corta del prompt sea natural. Hay tres buckets según la longitud (corto, medio, largo) y filtro de diversidad temática para evitar cinco instancias/artículos del mismo tema.
- **HumanEval**: Se han dividido los buckets según la dificultad estimada como longitud de la solución canónica dada (fácil - 100, medio - entre 100 y 300, difícil - más de 300 caracteres).
- **GSM8K**: Buckets divididos según el número de pasos del razonamiento canónico en el campo `answer` del dataset (fácil - 2 pasos, medio - 3 o 4 pasos, difícil - más de 5 pasos).

Gestión de la longitud

La condición de la longitud no se manipula simplemente truncando o expandiendo el texto, sino inyectando contexto irrelevante coherente con la tarea planteada.

La versión corta corresponde con el input nativo del dataset, mientras que la versión larga se construye envolviendo el input original en un párrafo de contexto irrelevante y no informativo generado por el agente Router. Se ha buscado en el input largo multiplicar aproximadamente $\times 3$ la longitud inicial con objetivos especícos por tarea: $\sim$3 000 tokens para XSum, $\sim$1 000 tokens en HumanEval y $\sim$600 en GSM8K.

Conjunto de ejemplos few-shot

Para las técnicas de *few-shot* y *CoT* se necesitan ejemplos demostrativos.

Se han preparado en una fase previa y almacenado en `artifacts/few_shot_examples/tarea.json`

- **GSM8K**: los *CoT* se toman directamente del split train del dataset original, siguiendo la metodología descrita en Wei et al. [26].
- **XSum y HumanEval**: las trazas *CoT* se generan con `Claude Sonnet 4.6` a partir del input y la respuesta esperada y **se revisan manualmente** antes de ser fijadas. Para HumanEval, los ejemplos se han cogido del dataset MBPP[56] en lugar del split train de HumanEval para evitar contaminación entre conjuntos.

3.3. Modelos evaluados

Se evalúan cinco modelos heterogéneos en proveedor, escala y forma de despliegue (Cuadro 3.1). La heterogeneidad es intencional ya que se busca ver cómo responde cada uno de los modelos a los efectos planteados.

Cuadro 3.1: Modelos evaluados con identificadores SDK fijados.

Modelo	Proveedor	Despliegue	Telemetría
GPT-5 mini [22]	OpenAI	API remota	EcoLogits [7]
Gemini 2.5 Flash [24]	Google	API remota	EcoLogits
Llama 3.2 3B [57]	Meta	Local (Ollama)[58]	CodeCarbon[6] + NVML
Mistral 7B [59]	Mistral AI	Local (Ollama)	CodeCarbon + NVML
Qwen 2.5 14B [60]	Alibaba	Local (Ollama)	CodeCarbon + NVML

Los tres modelos locales se ejecutan en el runtime de Ollama en formato *GGUF* cuantizado a 4 bits (`Q4_K_M`). La cuantización ofrece un compromiso conservador entre calidad/fidelidad de la respuesta y memoria [61] y permite que los modelos seleccionados quepan holgadamente en el hardware utilizado para este TFM (NVIDIA RTX 5070 Ti 16GB VRAM). Estos modelos se cargan y se descargan uno cada vez para evitar contaminación cruzada en la medición de la telemetría.

Parámetros de generación

Se han usado estos parámetros de generación en los cinco modelos:

- **temperature = 0.0**
- **top_p = 1.0**: siempre que el modelo lo acepte como parámetro junto `temperature`.
- **seed = 42**: siempre que el modelo lo soporte como parámetro.
- **max_tokens**: se ha limitado/truncado el tamaño de la respuesta para evitar gastos elevados (256 tokens para XSum, 1024 tokens para HumanEval y 512 para GSM8K). En ningún caso del experimento se ha llegado al truncamiento.
- **stop_sequences**: usado para HumanEval para evitar generación más allá del cuerpo del código pedido. Se han omitido para los modelos de OpenAI, ya que rechazan este parámetro.

3.4. Arquitectura del sistema multiagente

El experimento se orquesta mediante una arquitectura multiagente implementada con LangGraph [47], que gestiona cinco roles especializados: `Router`, `Rubric Generator`, invocación de modelos candidatos, `Judge` principal y `Validation Judge`. Cada rol tiene asignado un *system prompt* que define su comportamiento, la herramienta de *structured output* `tool_use`[62],

política de reintentos (*fallbacks*) y verificación de salida. La coordinación entre agentes se realiza mediante un estado compartido y persistido en una base de datos `SQLite`.

Separación temporal pre-procesado y experimento

Las invocaciones de algunos agentes se realizan de forma asíncrona en dos tiempos:

1. **Pre-procesado** (solo una vez al comienzo): el `Router` produce las 12 variantes del prompt para cada instancia y el `Rubric Generator` materializa la rúbrica. Ambos artefactos generados se persisten en disco (`artifacts/prompts` y `artifacts/rubrics`)

2. **Experimento**: el grafo de `LangGraph` carga los artefactos del paso anterior y ejecuta las inferencias evaluadas con la métrica de telemetría correspondiente y disparando la respuesta del `Judge` en cada ejecución.

Al hacerlo en estas dos fases se restringe la variabilidad de la generación de prompts y rúbricas al preprocesamiento, fijando el contenido generado, de forma que las diferencias observadas en el experimento son atrbuibles al experimento en sí y no a la regeneración del prompt.

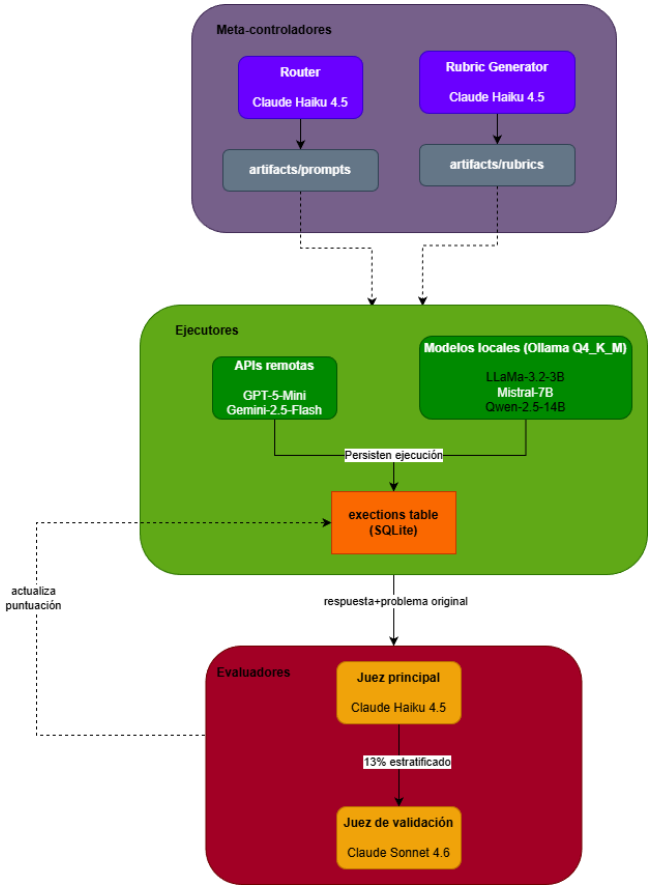


Figura 3.1: Arquitectura multiagente propuesta

Grafo LangGraph

El grafo se invoca una vez por cada combinación de tarea, instancia y repetición, siendo 45 invocaciones por cada una de ellas en el experimento principal. Cada invocación procesa las 60 ejecuciones evaluadas correspondientes a la instancia (12 celdas experimentales × 5 modelos). Los nodos del grafo (`src/graph/nodes.py`) son:

1. `init_node`: carga los prompts y rúbrica desde el disco y construye dos colas. Una cola para inferencia de API remotas y otra para inferencia en modelos locales.

2. `dispatch_router`: una función condicional que, mientras haya tareas API pendientes en la cola, lanza de forma concurrente (mediante semáforos) las llamadas a la API. Al vaciarse la cola de API procesa los modelos locales de uno en uno. Finalmente invoca la evaluación del juez en paralelo.

3. `api_fanout_node`, `api_inference_node`, `local_next_node`,`local_inference_node`, `collect_node`: los nodos de inferencia (*wrappers*) que envuelven la llamada al modelo aplicando la telemetría adecuada y persisten el resultado en `SQLite`.

4. `judge_fanout_node`, `judge_evaluation_node`: evaluación conb el juez de cada respuesta aplicando la rúbrica correspondiente.

5. `persist_node`: marca la invocación como completada y termina el grafo (`END`)

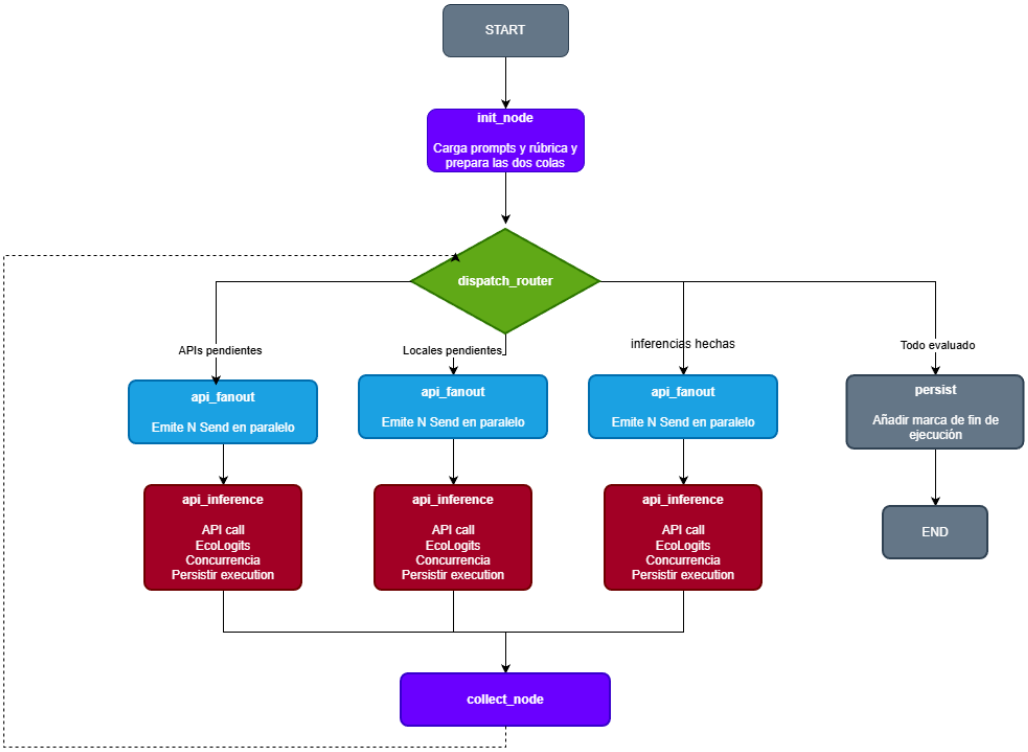


Figura 3.2: Flujo de LangGraph definido

Para asegurar poder recuperar ejecuciones en caso de fallo intermedio, se compila el grafo usando un *checkpointer* `SqliteSaver`[47].

En el caso de la concurrencia se ha usado para las invocaciones de API usando *threading* para invocar concurrentemente los modelos, siguiendo los límites establecidos en `src/graph/concurrency.py`.

Los resultados finalmente se persisten en una base de datos `SQLite` de cinco tablas (definidas en `src/data/persistence.py` y documentadas en el apéndice A):

- `experiments`: una fila por experimento.
- `executions`: una fila por inferencia evaluada con telemetría y juicio.
- `metasystem_executions`: consumo del meta-sistema (elementos que no forman parte de la inferencia principal como `Router` o `Judge`)
- `failures`: incidentes registrados durante el experimento.
- `system_baselines`: medidas de línea base del sistema para calcular el impacto real de los modelos vs el estado inicial del equipo usado.

3.5. Técnicas de prompting

Las tres técnicas de prompting se materializan mediante plantillas deterministas combinadas con el agente `Router`. Cada técnica está representada por un prompt canónico reutilizable, pero que se adapte al input específico:

- **Plantilla Zero-shot**[16]: el prompt contiene únicamente la descripción de la tarea y el input de la instancia.
- **Plantilla Few-shot**[16]: el prompt incluye tres ejemplos demostrativos seguidos del input real.
- **Plantilla Chain-of-Thought**[26]: el prompt incluye los mismos tres ejemplos pero acompañados de su traza de razonamiento y seguido finalmente del input final con la instrucción explícita de razonar paso a paso.

Por instancia en el `Router` se generan 3 técnicas $\times$ 2 longitudes $\times$ 2 formatos $=$ 12 variantes. Para minimizar la varianza introducida por el LLM y reducir costes de ejecución, solo se han generado las variantes *Largo-Plano* invocando al modelo de lenguaje (`Claude Haiku 4.5`). Las demás variantes derivan deterministamente usando las plantillas nativas en el caso del texto plano y envolviendo el contenido plano con las etiquetas semánticas correspondientesen el caso de las variantes XML.

Cabe mencionar que se ha optado por el formato XML como referencia para los prompts estructurados, pero también hay otros como JSON (ampliamente utilizado en aplicaciones reales

por su soporte nativo en los SDK de los proveedores [62]) o TOON [29], ya mencionado anteriormente, y que recientemente se ha vuelto popular al estar diseñada especificamente para el ahorro de tokens y por tanto de costes. La comparación sistemática de los tres formatos queda fuera del alcance de este TFM y se deja señalado como línea de trabajo futura en la sección 6.4

En el caso de la versión *Largo-Texto Plano*, el meta-prompt usado (`src/agents/rubric_templates/router_meta_prompts.py`) envuelve el input original con isntrucciones estrictas que no deben alterar la respuesta. Ha ocurrido que el filtro de seguridad de la API de Anthropic ha bloqueado algunos inputs. Para este caso se han fijado *fallbacks* como usar una variante conservadora con eliminación de posibles palabras restringidas o usar un relleno fijo con frases prefijadas, quedando registrado en base de datos qué método se ha usado.

3.6. Evaluación de calidad: LLM-as-a-Judge

La evaluación de calidad sigue el paradigma *LLM-as-a-Judge* [8] complementado por métricas objetivas por tarea y por una validación cruzada inter-juez con un modelo más potente (Claude Sonnet).

Calibración previa humano-juez

Cada tarea tiene prefijada una estructura fija de criterios, pesos y descripciones aplicable a todas las instancias de una tarea. Dichos criterios y tareas (`artifacts/rubrics`) se han calibrado con un humano de forma manual, aplicando unas 15-50 respuestas por tarea, calidad esperada y técnica. Un humano puntúa cada respuesta con la misma rúbrica y posteriormente se calcula la concordancia entre el humano y el juez en las mismas pruebas.

Dicha concordancia se ha verificado con dos métricas:

- **Correlación de Spearman**[63]: permite ver si el juez ordena las respuestas con el humano (puntuación final sobre 100 puntos). Spearman absorbe el sesgo de calibración como discrepancia. Se ha fijado un umbral de aceptación de $\rho \geq 0{,}6$, correspondiente a una zona de correlación moderada-alta según la guía de uso de Mukaka [64], ampliamente usada como referencia.
- **Cohen's Kappa cuadrático ponderado**[65]: permite ver en cada criterio si el juez da los mismos valores que el humano (escala del 1-5 por criterio). Penaliza desacuerdos grands más que los pequeños. Se ha fijado un umbral de aceptación de $\kappa \geq 0{,}4$, correspondiente a un nivel 'moderado' en la escala canónica de Landis y Koch [66], referencia estándar en la interpretación de este valor.

Validación cruzada inter-juez

Tras completar las 2 700 evaluaciones del juez principal, se ha ejecutado una segunda pasada con un modelo más potente (`Claude Sonnet 4.6` [23]) sobre una muestra estratificada por cada factos de 360 ejecuciones. La elección de este modelo como juez validador y de `Claude Haiku`

4.5 como juez principal sirve para evitar la aparición del sesgo de auto-preferencia que puede aparecer cuando unos modelos de una misma familia se evalúan entre sí, ya que los modelos evaluados son familias distintas.

Métricas objetivas

Como se ha comentado anteriormente en la sección 3.2, se han usado métricas objetivas para evaluar la validez del juez y que se han persistido al aplicarse en la tabla `executions.objective_metric_value`.

Se ha aplicado la correlación de Spearman entre `judge_score_total` y `objective_metric_value` por tarea.

Sesgo de verbosidad

Como verificación adicional se ha calculado la correlación de Spearman entre `response_tokens` y `judge_score_total` desglosada por técnica. Una correlación alta y positiva indicaría que el juez premia respuestas más verbosas; un valor moderado o bajo sería evidencia de que este sesgo conocido del paradigma *LLM-as-a-Judge*, ha sido mitigado correctamente con los criterios empleados.

3.7. Telemetría y medición de consumo

En el caso de la medición energética, se parte del problema de que no se dispone de la información de consumo real de los modelos remotos. Por tanto y como se comentó anteriormente en la sección 2.4 se ha usado:

- **CodeCarbon + NVML** para los modelos locales. Los tres modelos Ollama se han medido usando `CodeCarbon`, midiendo el factor de emisión de la red eléctrica española con una tasa de muestreo de 1s. En paralelo, el monitor NVML (librería`pynvml`) mide cada 0.5 segundos la temperatura de la GPU, el consumo instantáneo en vatios y la memoria usada. Estas medidas se han persistido en la base de datos y se ha usado `gpu_temp_avg_c` como covariable de control térmico en el modelo estadístico, ya que no se ha metido espaciado temporal en las ejecuciones por lo puede provocarse un calentamiento de la GPU. Debido al sistema operativo usado (Windows 11) la medición directa del consumo de CPU no es accesible por lo que se recurre a estimación basada en marca y modelo de CPU.
- **Ecologits** para modelos remotos: `EcoLogits` inyecta en la respuesta de la API una estimación probabilística de impactos ambientales basado en el análisis del ciclo de vida del modelo.

También se ha medido el consumo del meta-sistema (`Router`, `Rubric Generator`, `Principal Judge`, `Validation Judge`) usando `Ecologits` (para modelos Anthropic). Se ha persistido en

la tabla `metasystem_executions` y se ha estudiado anteriormente reportando el **ratio de consumo del sistema de inferencia vs meta-sistema**.

Para que todas estas mediciones sean metodológicamente correctas, se mide antes de cada sesión una línea base con el equipo usado en estado *idle*. Durante los experimentos se han cerrado todas las aplicaciones excepto Ollama.

3.8. Análisis estadístico

El análisis se estructura en seis bloques que responden a algunas de las preguntas planteadas. Se han ejecutado todos vía `scripts/07_analyze_results.py` usando la base de datos `SQLite` consolidada tras el experimento.

3.8.1. Modelo principal: efectos mixtos

Para cada variable dependiente se ajusta un modelo lineal mixto (Linear Mixed Model (LMM)) [67, 68] usando `statsmodels.MixedLM` [69] para ver si hay efecto:

```
log(energy_wh) ~ technique * length * format
                + model + task
                + gpu_temp_avg_c
                + (1 | instance_id)
```

donde `technique * length * format` expande a todos los efectos principales y sus interacciones de hasta tercer orden, `model` y `task` controlan la varianza atribuible al bloqueo, `gpu_temp_avg_c` actúa como covariable térmica (presente solo en modelos locales) y `(1 | instance_id)` modela un intercept aleatorio por instancia.

La elección de un LMM en lugar de un ANOVA clásico responde a la estructura jerárquica experimental que se ha aplicado en la metodología. El experimento combina, como se ha explicado anteriormente, tres factores manipulados *within-subjects* (técnica $\times$ longitud $\times$ formato) con dos factores de bloqueo *between-subjects* (modelo, tarea) y un factor fijo anidado (las instancias son una muestra). Tratar las instancias como factor fijo, como haría el ANOVA clásico, exageraría la potencia del análisis al ignorar la posible variabilidad inter-instancia [51]. Por tanto, el uso de LMM para el análisis estadístico de datos experimentales con medidas repetidas anidadas es una práctica estándar en el modelado de este tipo de diseños, ya que permite medir de forma adecuada la dependencia entre observaciones procedentes de un mismo modelo.

El modelo ajusta por separado las inferencias medidas por `EcoLogits` y las medidas con `CodeCarbon`.

Como nota metodológica, al realizar los cuatro ajustes (energía/calidad $\times$ API/local), el modelo reporta una covarianza singular para el intercept aleatorio por instancia de cero o cercana

a cero. Lo que implica que no hay variabilidad inter-instancia sistemática por lo que un modelo Ordinary Least Squares (OLS) podría usarse también, pero se mantiene LMM porque encaja con el diseño planteado y cambiarlo a posteriori podría considerarse una forma de manipulación de resultados.

3.8.2. Test entre técnicas

Para contrastar diferencias dos a dos entre las tres técnicas de prompting se ha usado un **test de rangos con signo de Wilcoxon pareado** [70] (pareado porque se pueden emparajar observaciones por tarea, instancias, repetición, longitud, formato y modelo; no paramétrico para evitar tener que asumir normalidad en las diferencias). Las tres comparaciones por familia se ajustan por **corrección de Bonferroni**[71] (corrección más conservadora porque tres comparaciones por familia son pocas).

3.8.3. Tamaños de efecto

Para medir la magnitud del efecto, se ha usado Cohen's d [72] con desviación típica agrupada (*pooled SD*). Se ha elegido este test porque con $n = 2700$ cualquier diferencia trivial daría un p-valor muy pequeño, lo que sería engañoso ya que sería debido al tamaño muestral. Cohen's d cuantifica magnitud independiente de n.

En este caso se ha usado la interpretación canónica de Cohen: $|d| < 0{,}2$ negligible, 0,2–0,5 pequeño, 0,5–0,8 medio, $\geq 0{,}8$ grande.

3.8.4. Análisis del meta-sistema

Para medir el propio sistema de medición, como se ha mencionado anteriormente, se han medido y reportado los consumos absolutos del meta-sistema

Visualizaciones complementarias

El pipeline diseñado para el análisis genera cuatro figuras usando `Matplotlib`:

- **Boxplot de consumo por (técnica $\times$ longitud $\times$ formato) y por modelo.**
- **Boxplot de calidad de respuesta por (técnica $\times$ longitud $\times$ formato) y por modelo.**
- **Heatmap de consumo medio modelo $\times$ técnica**
- **Barras de ovehead (consumo) por componente del meta-sistema**

3.9. Limitaciones metodológicas

Algunas de las limitaciones se han ido señalando a lo largo del capítulo, pero algunas limitaciones adicionales que se han identificado han sido:

- **Determinismo imperfecto**: a pesar de fijas la temperatura de los modelos 0 y fijar la seed, los modelos en GPU pueden mostrar variaciones y no-determinismo en las operaciones de punto flotante.
- **Thinking y Reasoning models**: los modelos `Gemini` y `GPT-5` usan razonamiento interno cuyo consumo no puede ser medido por `Ecologits`. En el caso de `Gemini` se ha podido desactivar con un parámetro llamado `thinking_budget`, pero para la familia de OpenAI no ha podido desactivarse. Esto se puede observar en los resultados obtenidos.
- **Evaluación humana limitada**: en el caso de la evaluación de la rúbrica, un solo operador (el propio alumno) ha realizado la evaluación sobre 15 prompts (5 prompts por cada instancia de tarea). Que sea un solo operador y un número tan bajo de prompts evaluados, hace que pueda existir un sesgo subyacente en la evaluación. Con un número más alto de prompts evaluados y un número mayor de evaluadores se podría mitigar y obtener un resultado estadístico aún más robusto.
- **Número limitado de ejecuciones**: se han ejecutado y evaluado 2 700 prompts, lo que es un número suficiente para obtener un análisis estadístico relevante. Si se hubieran realizado más repeticiones por cada prompt o se hubiera cogido un mayor número de instancias el resultado estadístico podría ser más robusto, pero el coste habría sido muchísimo más elevado (ver capítulo 5).

Capítulo 4

Resultados y discusión

En este capítulo se presentan los resultados obtenidos tras aplicar la metodología del capítulo anterior.

4.1. Cobertura del experimento

El experimento principal se ejecutó entre el día 5 de mayo y 6 de mayo de 2026 completando las 2 700 inferencias evaluadas previstas, todas ellas con la puntuación del juez principal correspondiente. La validación inter-juez con `Sonnet 4.6` se realizó sobre 360 celdas (cobertura efectiva de ∼13 %) de los cuales 359 produjeron pares válidos para el análisis estadístico.

Se registraron 182 fallos durante el experimento, todos recuperados mediante la implementación del parche `scripts/05b_patch_missing_cells.py`. La tasa neta de fallos no recuperados finalmente fue del 0 %. Todos los fallos fueron por la incompatibilidad de la limitación `stop_sequences` con a API de `GPT-5` en HumanEval.

La duración total fue de ∼9 horas en una sola sesión, con un promedio de ∼297 inferencias por hora.

4.2. Consumo energético por configuración

Distribución empírica

La figura 4.1 presenta la distribución de la energía consumida (`energy_wh`) por (técnica × longitud × formato), y representando cada modelo. El eje vertical está, como se ha comentado anteriormente, en escala logarítimica. La figura 4.2 es un heatmap que resume las medias de las evaluaciones por modelo y técnica.

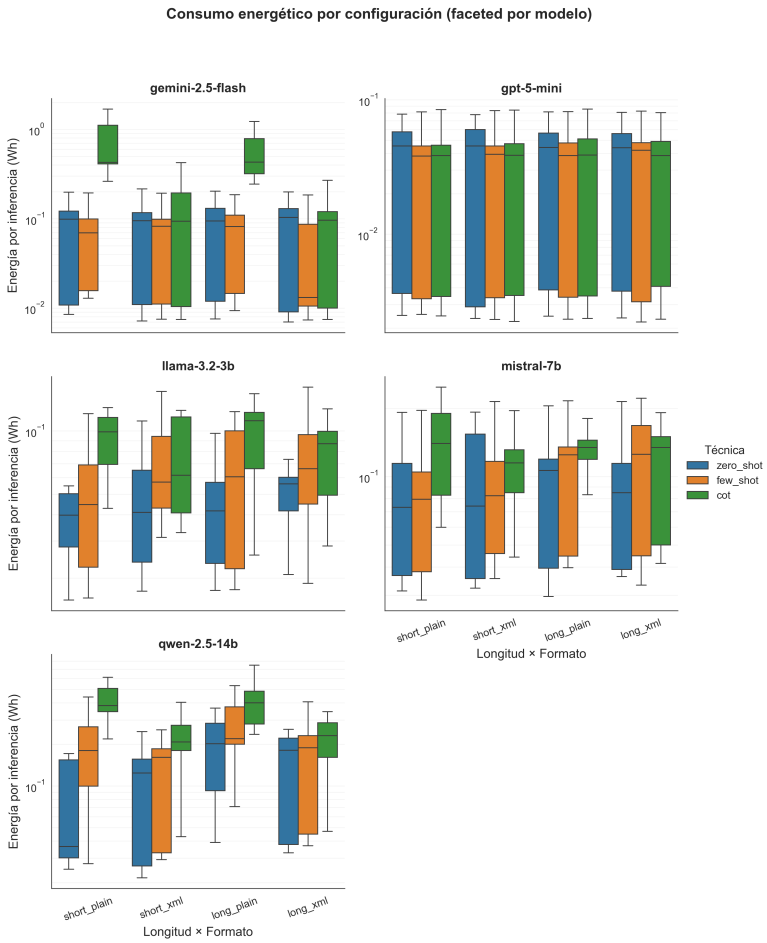


Figura 4.1: Boxplot con la distribución de consumo energético (en escala logarítmica) por configuración y modelo.

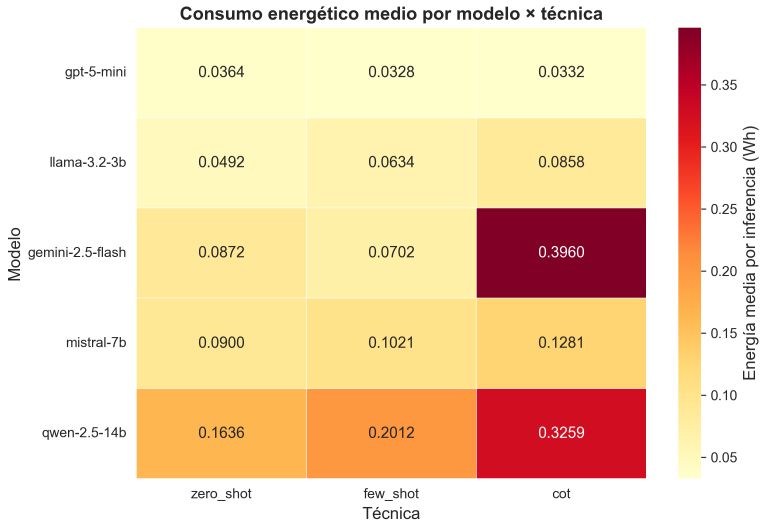


Figura 4.2: Heatmap de energía media por inferencia (Wh) por modelo y técnica

Ser observan tres patrones en estos gráficos:

- **Heterogeneidad entre modelos de varios órdenes de magnitud**: `GPT-5-Mini` consume entre 0.03 y 0.04 Wh por inferencia independientemente de la técnica, mientras que `Gemini-2.5-Flash` con CoT alcanza los 0.40 Wh ($\sim \times 12$ veces mayor) y `Qwen-2.5-14B` con CoT también alcanza los 0.33 Wh.
- **Coste creciente con el razonamiento explícito**: en todos los modelos locales el consumo aumenta con la técnica de la forma **Zero-shot** $<$ **Few-Shot** $<$ **CoT**, con el mayor incremento en CoT atribuible a un volumen mayor en los tokens de salida (152 tokens de longitud de respuesta CoT vs 74 en *Zero-shot*).
- **La excepción de GPT-5**: en `GPT-5-Mini` el consumo es prácticamente el mismo entre técnica, lo que sugiere lo que se ha comentado anteriormente de que posee razonamiento interno que absorbe la diferencia que otros modelos arrojan.

4.2.1. Modelo principal: efectos mixtos sobre consumo

El modelo lineal mixto se ha ajustado sobre `log(energy_wh)` ajustado por separado a APIs ($n = 1080$) y locales ($n = 1620$) converge en ambos casos. Los detalles completos se encuentran en el apéndice B

Los efectos principales *technique, length y format* son significativos a nivel $p < 0{,}001$, así como las interacciones de segundo y tercer orden (`technique:length` y `technique:length:format`).

Análisis de modelos invocados via API

En el caso de los modelos `Gemini-2.5-Flash` y `GPT-5-mini`

- **Zero-shot consume un $\sim$33.5 % de lo que consume CoT**
- **Few-shot consume un $\sim$31.4 % de lo que consume CoT**
- **El formato XML reduce un 29.8 % frente a formato texto plano en contextos largos**
- **GPT-5-mini consume alrededor de un 26 % de Gemini-2.5-Flash para una misma celda**

Análisis de modelos invocados via local

En el caso de los modelos `Qwen-2.5-14B`, `LLaMa-3.2-3B` y `Mistral-7B`:

- **Zero-shot consume el 50 % de CoT**.
- **Few-shot consume el 64.6 % de CoT** .
- **Qwen-2.5-14B consume $\times$2.57 veces lo que consume LLaMa-3.2-3B**, lo que es consistente con la diferencia proporcional de parámetros.

- **El efecto de la temperatura de la GPU en las ejecuciones afecta**. Por cada grado adicional de temperatura el consumo aumenta un 6.9 %.

4.2.2. Test de Wilcoxon y tamaño de efectos

Los resultados de Wilcoxon pareado con corrección de Bonferroni para las tres comparaciones técnica por técnica confirma que las tres técnicas son significativamente distintas en consumo energético.

Cuadro 4.1: Tests de Wilcoxon pareado con corrección de Bonferroni sobre log(energy_wh). Pareo por (tarea, instancia, repetición, longitud, formato, modelo); $n = 900$ pares por comparación.

Comparación	p_{Bonf}	Signif.
Zero-Shot *vs.* CoT	$4{,}16 \cdot 10^{-64}$	***
Zero-Shot *vs.* Few-Shot	$8{,}10 \cdot 10^{-08}$	***
CoT *vs.* Few-Shot	$4{,}83 \cdot 10^{-66}$	***

Al aplicar Cohen's d sobre la variable `log(energy_wh)`, se observa que *Zero-shot* vs *CoT* tiene $d = -0{,}485$ (pequeño, tirando a medio) y que *Zero-shot* vs *Few-shot* $d = -0{,}071$ (muy pequeño).

Es decir, el coste energético adicional de *Few-shot* sobre *Zero-shot* es prácticamente inexistente, mientras que *CoT* introduce un incremento sustancial.

4.2.3. Discusión

Los hallazgos parecen alinearse con los reportados por Patterson et al. [11] y Strubell et al. [10], que documentan diferencias de varios órdenes de magnitud en consumo energético entre modelos de tamaños distintos y estrategias de inferencia.

La diferencia de consumo entre *CoT* y *zero-shot* es coherente con los incrementos de consumo asociados a técnicas de prompting más sofisticadas reportada por Rubei et al. [28] en sus mediciones.

4.3. Calidad de las respuestas

Distribución empírica

En la figura 4.3 presenta la distribución de la puntuación del juez por modelo y técnica.

Los puntuajes medios por técnicas (agregando sobre todas las celdas y modelos son: *Zero-Shot* 49.0, *Few-Shot* 52.4 y *CoT* 53.1 sobre 100. Las diferencias absolutas son moderadas pero consistentes con los hallazgos de la literatura: las técnicas con ejemplos demostrativos [16] y razonamiento explícito [26] mejoran el rendimiento sobre *Zero-shot*.

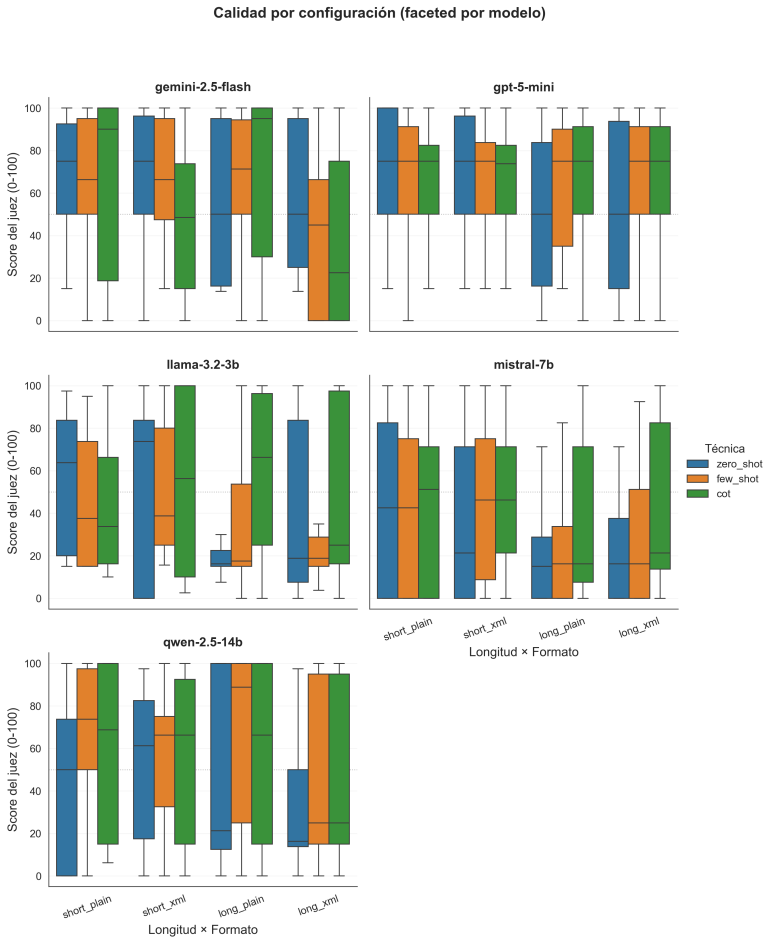


Figura 4.3: Distribución de la puntuación normalizada del juez (0-100) por configuración y modelo

4.3.1. Modelo principal: efectos mixtos sobre calidad de respuesta

Usando el modelo ajustado disponible en el apéndice B para `judge_score_total` (escala 0-100) como variable dependiente se observa:

- En invocación vía API, **Few-shot es estadísticamente indistinguible de CoT** (2 puntos de diferencia, p-valor no significativo). En cambio, *Zero-shot* pierde casi 20 puntos frente a *CoT*.
- En modelos locales, sí hay diferencia entre *Few-Shot* y *CoT*. **En modelos locales Few-shot pierde casi 9 puntos frente a CoT** con p-valor significativo a nivel 95. *Zero-shot* pierde de forma parecida.

- **El formato XML reduce sistemáticamente la calidad**: -13.4 puntos para modelos vía API, -9.6 para modelos locales.
- En el caso de los modelos: `GPT-5-mini` puntúa 6.6 puntos por encima de `Gemini-2.5-Flash` (p-valor significativo) y `Qwen-2.5-14B` puntúa 7.8 puntos por encima de `LLaMa-3.2-3B`.
- Como cabría suponer, la temperatura de la GPU no es un factor relevante en la calidad del output.

4.3.2. Test de Wilcoxon y tamaño de efectos

Los contrastes pareados sobre `judge_score_total` (Cuadro 4.2) muestran un patrón distinto al del consumo: *Zero-shot* difiere significativamente de *CoT* y *Few-shot*, pero **CoT y Few-shot son indistinguibles** estadísticamente hablando.

Cohen's *d* confirma que la magnitud es trivial en el caso de *Zero-shot* contra las otras técnicas.

Cuadro 4.2: Tests post-hoc sobre `judge_score_total`; $n = 900$ pares por comparación.

Comparación	p_{Bonf}	Sig.
Zero-Shot *vs.* CoT	0,000819	***
Zero-Shot *vs.* Few-Shot	0,0155	*
CoT *vs.* Few-Shot	1,00	n.s.

La lectura de estos resultados es clara, CoT ofrece una mejora de calidad estadísticamente significativa pero prácticamente despreciable a costa de un incremento energético sustancial. *Few-shot* mejora la calidad sobre Zero-shot con coste energético insignificante, por lo que sería la elección más eficiente.

4.3.3. Discusión

La equivalencia entre *few-shot* y *CoT* en calidad contrasta con el supuesto generalizado en la comunidad de prompting de que el razonamiento explícito mejora sistemáticamente las respuestas [26]. Otros estudios apuntan en la misma dirección que nuestro hallazgo como Kojima et al. [27] que muestran que **la mejora del razonamiento paso a paso depende del modelo y de la tarea y no es una mejora universal**.

4.4. Validez del LLM-as-a-Judge

Sobre las 359 celdas con resultados válidos se ha aplicado la correlación de Spearman [63] entre los totales de Haiku y Sonnet, obteniendo un $\rho = +0{,}948$ con p-valor muy cercano a

cero. La concordancia en el ordenamiento de las respuestas en ambos jueces es prácticamente perfecta.

El análisis criterio a criterio mediante Cohen's kappa cuadrático ponderado [65] muestra valores $\kappa \geq 0{,}74$ en lso catorce criterios principales establecidos, con la excepción de `gsm8k.conciseness` ($\kappa = 0{,}187$). Esta divergencia se puede interpretar como una indeterminación del criterio de concisión en GSM8K: respuestas matemáticamente correctas pueden ser percibidas como concisas o verbosas según la preferencia del juez, sin tener en cuenta el contenido.

4.4.1. Juez vs. métrica objetiva

Cuadro 4.3: Correlación de Spearman entre la puntuación del juez sintético y métricas objetivas no basadas en LLM. $n = 900$ por tarea.

Tarea	**Métrica objetiva**	ρ	p	
GSM8K	exact_match (48,2 % pass)	+0,868	$7{,}8 \cdot 10^{-276}$	***
HumanEval	pass@1 (36,1 % pass)	+0,600	$5{,}6 \cdot 10^{-89}$	***
XSum	ROUGE-L (avg 0,162) [36]	+0,467	$6{,}6 \cdot 10^{-50}$	***
XSum	BERTScore F1 (avg 0,869) [53]	+0,516	$2{,}7 \cdot 10^{-62}$	***

En el cuadro 4.3 se puede ver la correlación de Spearman entre la puntuación del juez y la métrica objetica para cada tarea, obteniendo una correlación positiva y moderada-alta en todas las tareas con p-valores que rechazan la hipótesis nula con un margen muy amplio, validando el resultado.

La correlación más alta corresponde a GSM8K, donde la métrica objetiva de coincidencia del número final está fuertemente alineada con el peso del resultado en la rúbrica (35 % de la nota). En XSum, la diferencia entre `ROUGE-L` y `BERTScore` ilustra que la similaridad semántica parece capturar mejor la calidad del resumen que el solapamiento léxico.

4.4.2. Sesgo de verbosidad

Cuadro 4.4: Sesgo de verbosidad: correlación de Spearman entre número de *tokens* de la respuesta y la puntuación del juez, por técnica.

Técnica	*tokens* medios	ρ	p_{Bonf}
Zero-Shot	74,1	+0,418	$5{,}8 \cdot 10^{-39}$
Few-Shot	81,0	+0,377	$2{,}9 \cdot 10^{-31}$
CoT	152,5	+0,152	$1{,}4 \cdot 10^{-05}$

La correlación de Spearman entre `response_tokens` y `judge_score_total` desglosada por técnica de la tabla 4.4 es positiva y significativa en las tres técnicas, pero notablemente más baja en CoT que en las otras dos técnicas.

Se puede intepretar como que en Zero-shot y Few-shot las respuestas más desarrolladas tienden a ser más completas y mejor puntuadas. En CoT, donde la longitud está forzada por la instrucción explícita de razonamiento paso a paso, la correlación baja.

Esto puede implicar de forma no directa que el juez **no recompensa verbosidad de forma automática**, sino contenido sustantivo, es decir, cuando la longitud es obligatoria deja de correlacionar con calidad.

4.4.3. Discusión

Este hallazgo dontrasta con la literatura existente sobre este sesgo de verbosidad en la evaluación de LLM. Saito et al. [73] documentan que los LLMs utilizados como evaluadores tienden a preferir respuestas más largas independientemente de su calidad, sesgo que también identifica Zheng et al. [8] en el benchmark MT-Bench. En este TFM el sesgo parece ser parcialmente mitigado por la inclusión de un criterio explícito de *conciseness* con peso entre el 10 % y 15 % en las rúbricas de las tres tareas. Sin embargo, la correlación residual en *zero-shot* y *few-shot* podría reflejar que no ha sido mitigado completamente.

4.5. Latencia y throughput

Se han medido también las métricas de Time-To-First-Token (TTFT) y velocidad de generación de tokens (tokens/s) por cada modelo como se muestra en las tablas 4.5 y 4.6. Cada modelo tiene 540 mediciones y se han señalado la mediana y el percentil 90 %.

Cuadro 4.5: TTFT por modelo. Mediana y percentil 90.

Modelo	TTFT p_{50} (s)	TTFT p_{90} (s)
mistral-7b	0,185	0,591
llama-3.2-3b	0,195	0,408
qwen-2.5-14b	0,509	1,342
gemini-2.5-flash	0,522	0,825
gpt-5-mini	0,576	0,954

Cuadro 4.6: *Throughput* de generación (*tokens*/s).

Modelo	TPS p_{50}	TPS p_{90}
llama-3.2-3b	146,4	200,1
mistral-7b	98,0	138,6
gemini-2.5-flash	72,2	156,3
gpt-5-mini	44,2	101,0
qwen-2.5-14b	9,5	11,0

En las tablas 4.5 y 4.6 se observan algunos hallazgos interesantes:

- **Los modelos locales baten a las APIs en TTFT**: especialmente los modelos pequeños de Mistral y LLaMa. La llamada de red es mayor que el tiempo de *prefill* de un modelo pequeño en GPU local.
- **Qwen-2.5-14B tiene un throughput muy limitado**: 9-11 tokens/s independientemente de la técnica, muy por debajo del resto de modelos. Combinado con su consumo medio, es una opción no muy recomendable para escenarios sensibles al coste.

4.6. Consumo del meta-sistema

En la figura 4.4 se resume el consumo total del meta-sistema desglosado por componente, con barras de error correspondientes a los brackets low/high de la estimación de `EcoLogits`.

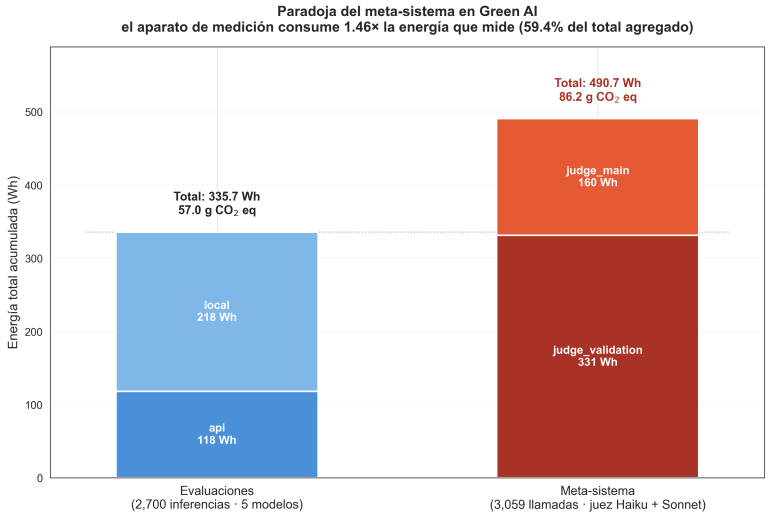


Figura 4.4: Consumo energético del meta-sistema por componente (Wh)

- **Evaluaciones** (2 700 inferencias): 335.73 Wh $\equiv$ 69.52 gCO_2eq
- **Meta-sistema** (Haiku y Sonnet): 490.73 Wh $\equiv$ 101.62 gCO_2eq

- **Ratio meta/evals: 1.46**. El meta-sistema consume un 59.38 % del total agregado.

Si se entra en el desglose interno del meta-sistema se revela el origen del consumo:

- `judge_main` (`Claude Haiku 4.5`, 2 700 llamadas): 159.57 Wh, coste por llamada ∼0.059Wh
- `judge_validation` (`Claude Sonnet 4.6`, 360 llamadas): 331.16 Wh, coste por llamada ∼0.920 Wh.

Sonnet es 15 veces más costoso por llamada que Haiku. Este es el resultado clave: el muestreo de validación (∼13 % de las ejecuciones) representan el 67 % del coste energético del meta-sistema a pesar de ser un menor número de inferencias. El `Router` y el `Rubric Generator`, que se han registrado bajo otro experimento de pre-procesado diferente, añaden ∼1 Wh de consumo; despreciable frente al coste de los jueces.

4.6.1. Discusión

El ratio de overhad del meta-sistema de 1.46 que se ha obtenido en este estudio implica que la medición energética consume más que la propia ejecución del sistema medido. La literatura sobre validación de jueces sintéticos como MT-Bench [8], se ha centrado en la calidad de la concordancia entre el juez automático y la evaluación humana, pero no cuantifica el coste energético del proceso de evaluación, por lo que no se dispone de un valor de referencia directo con el que contrastar el ratio obtenido.

El factor diferencial en este trabajo, es la inclusión de la segunda capa de validación con el inter-juez con el juez más capaz (`Sonnet 4.6`). Esta decisión metodológica prioriza la robustez de la validez del juicio sobre la eficiencia energética. Este *trade-off* entre rigor metodológico y consumo del meta-sistema es propio de los experimentos del campo de *Green AI* [4, 3] y es una de las paradojas operativas del estudio: consume más la medida del sistema que el propio sistema.

Capítulo 5

Valoración económica

En este capítulo se desglosarán los costes asociados al desarrollo y ejecución del experimento.

APIs comerciales

El uso de los modelos Claude, Gemini y GPT generan costes por cada token consumido. Para completar el experimento realizado en este TFM estos han sido los consumos reales:

Cuadro 5.1: Coste de las APIs comerciales del experimento.

Componente	Modelo	In (MTok)	Out (MTok)	Coste (€)
Router (preprocesado)	Haiku 4.5	0,006	0,007	∼ 0,52
Generador de Rúbricas	Haiku 4.5	0,019	0,003	∼ 0,49
Juez principal	Haiku 4.5	5,904	1,518	∼ 11,28
Juez de validación	Sonnet 4.6	0,769	0,176	∼ 5,18
GPT-5 mini (evaluación)	OpenAI	0,619	0,025	∼ 0,40
Gemini 2.5 Flash (evaluación)	Google	0,654	0,059	∼ 0,25
Subtotal APIs				∼ 18,12

Cabe destacar que el juez principal (`Claude Haiku 4.5`, 2 753 llamadas) y el juez de validación (`Claude Sonnet 4.6`, 360 llamadas) concentran ∼98 % del coste de API. Lo cual indica que el meta-sistema es lo más costoso.

Energía eléctrica

La energía total consumida durante el experimento ha sido 335.73 Wh en las inferencias + 495.19 Wh en el meta-sistema. Si se aplica la media residencial española de 0.1433€/kWh el día 6 de mayo de 2026 [74]:

$$0{,}831\ \text{kWh} \times 0{,}1433\,\frac{€}{\text{kWh}} \approx 0{,}119 €$$

Hardware

Se ha usado una GPU NVIDIA RTX 5070Ti (coste aproximado de 900€). Si se estima una vida útil de cinco años, el experimento que ha tenido ∼9 horas de cómputo, supone un valor de amortización muy bajo que se puede omitir frente al resto de costes.

Capítulo 6

Conclusiones y trabajos futuros

El código, artefactos y datos utilizados en este TFM se pueden encontrar en el repositorio público de GitHub: `https://github.com/lepablito/TFM_UOC_PabloMarcosParra`

6.1. Conclusiones principales

A partir de los resultados del capítulo 4 se pueden extraer algunas conclusiones:

- **Existe un trade-off cuantificable entre técnica de prompting y consumo energético**, siendo su magnitud muy dependiente del modelo.
- **La mejora de calidad por aplicar CoT o Few-shot sobre Zero-shot es estadísticamente significativa**. El supuesto habitual de que el razonamiento explícito mejora sistemáticamente las respuestas es desbancado, ya que **la calidad de respuesta de CoT y Few-shot es estadísticamente indistinguible**.
- **El equilibrio entre coste y calidad lo gana Few-shot**. Ofrece una respuesta similar a *CoT* con un coste energético equivalente al de *Zero-shot*.
- **La respuesta del juez sintético es fiable**. Como se ha visto en el capítulo anterior, supera las tres capas de validación planteadas (validación inter-juez, validación contra métricas objetivas y validación de mitigación de sesgo de verbosidad).
- **Hay una paradoja en el consumo energético del meta-sistema versus las evaluaciones**. Medir el consumo gasta más energía que la medida. Además el coste del juez de validación ha sido relativamente desproporcionado, lo que motiva una posible investigación futura de estrategias más eficientes.
- **Los modelos pequeños locales parecen tener ventaja en TTFT**

6.2. Logro de objetivos

Se han conseguido todos los objetivos previstos inicialmente en la sección 1.3, tanto el objetivo principal como los subobjetivos.

6.3. Seguimiento de planificación y metodología

Se ha seguido la planificación definida en la sección 1.6 y la metodología propuesta.

6.4. Líneas de trabajo futuras

Quedan abiertas las siguientes líneas de trabajo como continuación:

1. **Nuevo formato adicional (TOON)**: añadir un tercer formato sintáctico TOON frente a texto plano y XML. TOON es un formato estructurado pensado para la reducción de consumo de tokens y por tanto de costes y consumo energético, por lo que tendría un gran potencial como referencia de *Green AI*.

2. **Ampliar el conjunto de instancias por tarea**: aumentar el número de instancias de 5 a 10-15 fortalecería la generalización a nivel instancia. El volumen del experimento principal y, por tanto, del coste se incrementaría proporcionalmente.

3. **Incluir otras técnicas avanzadas**: incluir algunas técnicas más complejas que CoT como *Self-Consistency* o *Tree-of-Thoughts* para mapear nuevas técnicas.

4. **Buscar estrategias de validación más eficientes**: el juez de validación con `Claude Sonnet 4.6` ha sido muy costoso energética y económicamente, habría que evaluar alternativas.

5. **Replicación con otros modelos adicionales**, tanto comerciales como locales. En el caso de los modelos comerciales podría estar interesante estudiar la capacidad y coste de modelos *high-end* como `Gemini-2.5-Pro` o `GPT-5.4`.

6. **Estudio del coste del razonamiento interno**: los modelos modernos, como la familia `GPT-5` tienen razonamiento interno implícito, cuyo valor de consumo de tokens no se ve reflejado en las medidas reportadas por el SDK. Una línea futura podría replicar el experimento con modelos que permiten manipular el nivel de pensamiento como `Gemini` o `Claude` para cuantificar ese coste oculto de razonamiento interno y compararlo con CoT explícito.

Glosario

API Application Programming Interface. 10, 12, 16, 19, 21, 24, 33, 34, 35, 36, 40, 42, 44, 45, 48

CoT Chain-of-Thought. 17, 23, 25, 30, 42, 43, 44, 45, 47, 52, 53

CPU Central Processing Unit. 24, 36

GenAI Inteligencia Artificial Generativa. 9, 10, 12

GPU Unidades de Procesamiento de Gráficos. 20, 24, 36, 39, 43, 45, 48

IA Inteligencia Artificial. 5, 9, 10, 12, 17, 18, 23, 27

LLM Large Language Model. 3, 4, 5, 9, 10, 11, 12, 13, 16, 19, 21, 22, 23, 24, 25, 26, 27, 29, 34, 47

LMM Linear Mixed Model. 37, 38

meta-sistema Conjunto de componentes basados en el uso de Large Language Models que orquestan el experimento sin ser objeto de evaluación. 5, 15, 36, 37, 38, 49

MoE Mixture of Experts. 21

NLP Natural Language Processing. 18, 19, 29

OLS Ordinary Least Squares. 38

RAM Random Access Memory. 24

RAPL Running Average Power Limit. 24

SDK Soqftware Development Kit. 8, 31, 35, 53

TFM Trabajo Final de Máster. 10, 11, 12, 13, 16, 18, 19, 24, 25, 26, 31, 35, 47, 50, 52

TOON Token-Oriented Object Notation. 23, 35, 53

TTFT Time-To-First-Token. 47, 48, 52

Bibliografía

[1] Alex De Vries. «The growing energy footprint of artificial intelligence». En: *Joule* 7.10 (2023), págs. 2191-2194.

[2] Patrick Wilhelm, Thorsten Wittkopp y Odej Kao. «Beyond test-time compute strategies: Advocating energy-per-token in llm inference». En: *Proceedings of the 5th Workshop on Machine Learning and Systems*. 2025, págs. 208-215.

[3] Roberto Verdecchia, June Sallou y Luís Cruz. «A systematic review of Green AI». En: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 13.4 (2023), e1507.

[4] Roy Schwartz et al. «Green ai». En: *Communications of the ACM* 63.12 (2020), págs. 54-63.

[5] Marta Adamska et al. «Green prompting». En: *arXiv preprint arXiv:2503.10666* (2025).

[6] Alexandre Lacoste et al. «Quantifying the carbon emissions of machine learning». En: *arXiv preprint arXiv:1910.09700* (2019).

[7] Samuel Rincé y Adrien Banse. «Ecologits: Evaluating the environmental impacts of generative AI». En: *Journal of Open Source Software* 10.111 (2025), pág. 7471.

[8] Lianmin Zheng et al. «Judging llm-as-a-judge with mt-bench and chatbot arena». En: *Advances in neural information processing systems* 36 (2023), págs. 46595-46623.

[9] Briony J Oates, Rachel McLean y Marie Griffiths. «Researching information systems and computing». En: (2022).

[10] Emma Strubell, Ananya Ganesh y Andrew McCallum. «Energy and policy considerations for deep learning in NLP». En: *Proceedings of the 57th annual meeting of the association for computational linguistics*. 2019, págs. 3645-3650.

[11] David Patterson et al. «Carbon emissions and large neural network training». En: *arXiv preprint arXiv:2104.10350* (2021).

[12] Alexandra Sasha Luccioni, Sylvain Viguier y Anne-Laure Ligozat. «Estimating the carbon footprint of bloom, a 176b parameter language model». En: *Journal of machine learning research* 24.253 (2023), págs. 1-15.

[13] Sasha Luccioni, Yacine Jernite y Emma Strubell. «Power hungry processing: Watts driving the cost of AI deployment?» En: *Proceedings of the 2024 ACM conference on fairness, accountability, and transparency*. 2024, págs. 85-99.

[14] Ashish Vaswani et al. «Attention is all you need». En: *Advances in neural information processing systems* 30 (2017).

[15] Jacob Devlin et al. «Bert: Pre-training of deep bidirectional transformers for language understanding». En: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 2019, págs. 4171-4186.

[16] Tom Brown et al. «Language models are few-shot learners». En: *Advances in neural information processing systems* 33 (2020), págs. 1877-1901.

[17] Jared Kaplan et al. «Scaling laws for neural language models». En: *arXiv preprint arXiv:2001.08361* (2020).

[18] Jordan Hoffmann et al. «Training compute-optimal large language models». En: *arXiv preprint arXiv:2203.15556* 10 (2022).

[19] William Fedus, Barret Zoph y Noam Shazeer. «Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity». En: *Journal of Machine Learning Research* 23.120 (2022), págs. 1-39.

[20] Albert Q Jiang et al. «Mixtral of experts». En: *arXiv preprint arXiv:2401.04088* (2024).

[21] Josh Achiam et al. «Gpt-4 technical report». En: *arXiv preprint arXiv:2303.08774* (2023).

[22] OpenAI. *Modelos GPT*. `https://developers.openai.com/api/docs/models` [Accedido: 22/03/2026]. 2026.

[23] Anthropic. *Modelos Claude*. `https://platform.claude.com/docs/en/about-claude/models/overview` [Accedido: 22/03/2026]. 2026.

[24] Google. *Modelos Gemini API*. `https://ai.google.dev/gemini-api/docs/models` [Accedido: 22/03/2026]. 2026.

[25] Hugo Touvron et al. «Llama: Open and efficient foundation language models». En: *arXiv preprint arXiv:2302.13971* (2023).

[26] Jason Wei et al. «Chain-of-thought prompting elicits reasoning in large language models». En: *Advances in neural information processing systems* 35 (2022), págs. 24824-24837.

[27] Takeshi Kojima et al. «Large language models are zero-shot reasoners». En: *Advances in neural information processing systems* 35 (2022), págs. 22199-22213.

[28] Riccardo Rubei et al. «Prompt engineering and its implications on the energy consumption of Large Language Models». En: *arXiv preprint arXiv:2501.05899* (2025).

[29] Johann Schopplich. *Token-Oriented Object Notation (TOON)*. `https://github.com/toon-format/toon` [Accedido: 22/03/2026]. 2026.

[30] Mathilde Jay et al. «An experimental comparison of software-based power meters: focus on CPU and GPU». En: *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE. 2023, págs. 106-118.

[31] Peter Henderson et al. «Towards the systematic reporting of the energy and carbon footprints of machine learning». En: *Journal of machine learning research* 21.248 (2020), págs. 1-43.

[32] Lasse F Wolff Anthony, Benjamin Kanding y Raghavendra Selvan. «Carbontracker: Tracking and predicting the carbon footprint of training deep learning models». En: *arXiv preprint arXiv:2007.03051* (2020).

[33] Codecarbon. *Codecarbon*. `https://codecarbon.io/`, [Accedido: 18/05/2026]. 2024.

[34] Dan Hendrycks et al. «Measuring massive multitask language understanding». En: *arXiv preprint arXiv:2009.03300* (2020).

[35] Percy Liang et al. «Holistic evaluation of language models». En: *arXiv preprint arXiv:2211.09110* (2022).

[36] Chin-Yew Lin. «ROUGE: A Package for Automatic Evaluation of Summaries». En: *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*. Barcelona, Spain: Association for Computational Linguistics, 2004, págs. 74-81.

[37] Wei-Lin Chiang et al. «Chatbot arena: An open platform for evaluating llms by human preference». En: *Forty-first International Conference on Machine Learning*. 2024.

[38] Yang Liu et al. «G-eval: NLG evaluation using gpt-4 with better human alignment». En: *Proceedings of the 2023 conference on empirical methods in natural language processing*. 2023, págs. 2511-2522.

[39] Seungone Kim et al. «Prometheus: Inducing fine-grained evaluation capability in language models». En: *The Twelfth International Conference on Learning Representations*. 2023.

[40] Haitao Li et al. «Llms-as-judges: a comprehensive survey on llm-based evaluation methods». En: *arXiv preprint arXiv:2412.05579* (2024).

[41] Jiawei Gu et al. «A survey on llm-as-a-judge». En: *The Innovation* (2024).

[42] Shunyu Yao et al. «React: Synergizing reasoning and acting in language models». En: *The eleventh international conference on learning representations*. 2022.

[43] Lei Wang et al. «A survey on large language model based autonomous agents». En: *Frontiers of Computer Science* 18.6 (2024), pág. 186345.

[44] Changle Qu et al. «Tool learning with large language models: A survey». En: *Frontiers of Computer Science* 19.8 (2025), pág. 198343.

[45] Taicheng Guo et al. «Large language model based multi-agents: A survey of progress and challenges». En: *arXiv preprint arXiv:2402.01680* (2024).

[46] Tula Masterman et al. *The Landscape of Emerging AI Agent Architectures for Reasoning, Planning, and Tool Calling: A Survey*. 2024.

[47] LangChain, Inc. «LangGraph: Build Resilient Language Agents as Graphs». En: *Documentation* (2024). `https://langchain-ai.github.io/langgraph/`, [Accedido: 17/05/2026].

[48] Qingyun Wu et al. «AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation». En: *Proceedings of the Conference on Language Modeling (COLM)*. 2024.

[49] Sirui Hong et al. «MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework». En: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2024.

[50] CrewAI Inc. *CrewAI: Framework for Orchestrating Role-Playing, Autonomous AI Agents*. `https://github.com/crewAIInc/crewAI`, [Accedido: 17/05/2026]. 2024.

[51] Scott E. Maxwell y Harold D. Delaney. *Designing Experiments and Analyzing Data: A Model Comparison Perspective*. 2nd. Mahwah, NJ: Lawrence Erlbaum Associates, 2004.

[52] Shashi Narayan, Shay B. Cohen y Mirella Lapata. «Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization». En: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Brussels, Belgium: Association for Computational Linguistics, 2018, págs. 1797-1807.

[53] Tianyi Zhang et al. «BERTScore: Evaluating Text Generation with BERT». En: *International Conference on Learning Representations (ICLR)*. 2020.

[54] Mark Chen et al. «Evaluating Large Language Models Trained on Code». En: *arXiv preprint arXiv:2107.03374* (2021).

[55] Karl Cobbe et al. «Training Verifiers to Solve Math Word Problems». En: *arXiv preprint arXiv:2110.14168* (2021).

[56] Jacob Austin et al. «Program Synthesis with Large Language Models». En: *arXiv preprint arXiv:2108.07732* (2021).

[57] Aaron Grattafiori et al. «The Llama 3 Herd of Models». En: *arXiv preprint arXiv:2407.21783* (2024).

[58] Ollama. *Ollama: Get Up and Running with Large Language Models Locally*. `https://ollama.com/`, [Accedido: 10/05/2026]. 2024.

[59] Albert Q. Jiang et al. «Mistral 7B». En: *arXiv preprint arXiv:2310.06825* (2023).

[60] An Yang et al. «Qwen2.5 Technical Report». En: *arXiv preprint arXiv:2412.15115* (2024).

[61] Georgi Gerganov y llama.cpp contributors. *llama.cpp: LLM Inference in C/C++*. `https://github.com/ggerganov/llama.cpp`, [Accedido: 10/05/2026]. 2023.

[62] Anthropic. *Tool Use with Claude - Anthropic Documentation*. `https://platform.claude.com/docs/en/agents-and-tools/tool-use/overview` [Accedido: 17/05/2026]. 2024.

[63] Charles Spearman. «The Proof and Measurement of Association between Two Things». En: *The American Journal of Psychology* 15.1 (1904), págs. 72-101.

[64] Mavuto M. Mukaka. «A guide to appropriate use of correlation coefficient in medical research». En: *Malawi Medical Journal* 24.3 (2012), págs. 69-71.

[65] Jacob Cohen. «Weighted Kappa: Nominal Scale Agreement with Provision for Scaled Disagreement or Partial Credit». En: *Psychological Bulletin* 70.4 (1968), págs. 213-220.

[66] J. Richard Landis y Gary G. Koch. «The Measurement of Observer Agreement for Categorical Data». En: *Biometrics* 33.1 (1977), págs. 159-174.

[67] José C. Pinheiro y Douglas M. Bates. *Mixed-Effects Models in S and S-PLUS*. New York: Springer, 2000.

[68] Douglas Bates et al. «Fitting Linear Mixed-Effects Models Using lme4». En: *Journal of Statistical Software* 67.1 (2015), págs. 1-48.

[69] Skipper Seabold y Josef Perktold. «Statsmodels: Econometric and Statistical Modeling with Python». En: *Proceedings of the 9th Python in Science Conference (SciPy 2010)*. 2010, págs. 92-96.

[70] Frank Wilcoxon. «Individual Comparisons by Ranking Methods». En: *Biometrics Bulletin* 1.6 (1945), págs. 80-83.

[71] Olive Jean Dunn. «Multiple Comparisons among Means». En: *Journal of the American Statistical Association* 56.293 (1961), págs. 52-64.

[72] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. 2nd. Hillsdale, NJ: Lawrence Erlbaum Associates, 1988.

[73] Keita Saito et al. «Verbosity Bias in Preference Labeling by Large Language Models». En: *arXiv preprint arXiv:2310.10076* (2023).

[74] Fiva Energía. *Consumo medio (kWh) España 6 de mayo 2026*. `https://www.fiva.es/precio-luz/07-mayo-2026` [Accedido: 10/05/2026]. 2026.

Apéndice A

Base de datos SQLite

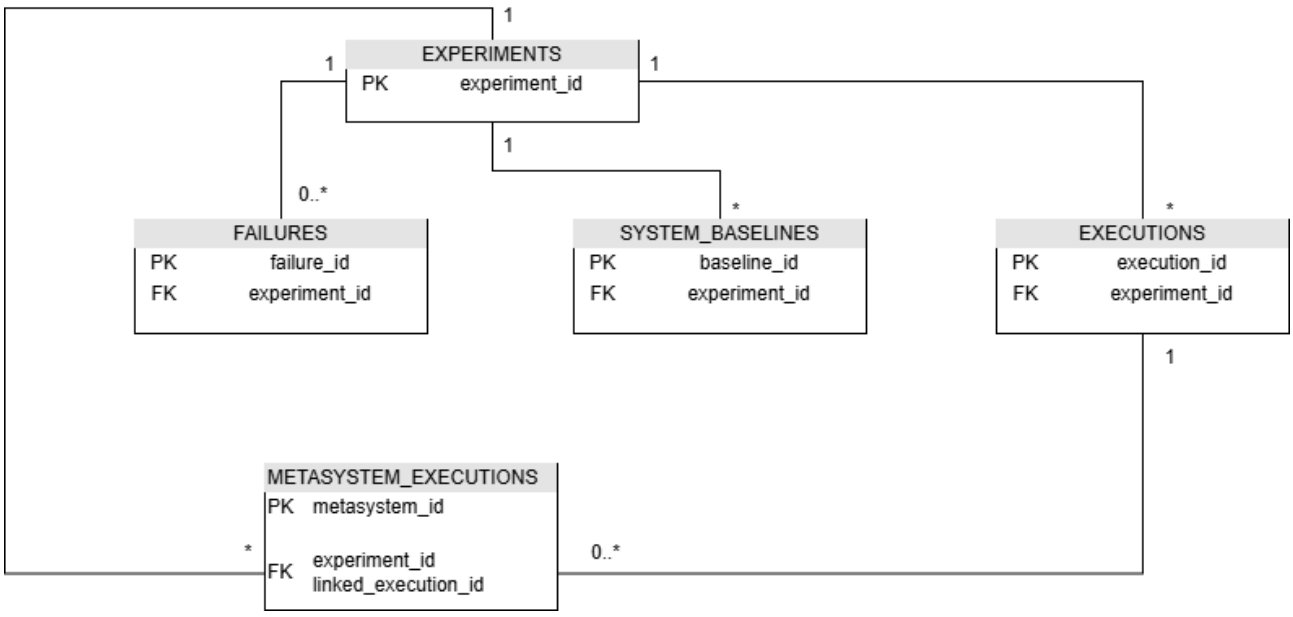


Figura A.1: Diagrama resumido de la base de datos propuesta

Esquema completo

```
"""
CREATE TABLE IF NOT EXISTS experiments (
    experiment_id      TEXT PRIMARY KEY,
    started_at         TIMESTAMP NOT NULL,
    finished_at        TIMESTAMP,
    config_hash        TEXT NOT NULL,
    git_commit_hash    TEXT NOT NULL,
    seed               INTEGER NOT NULL,
    operator           TEXT,
```

```
    notes               TEXT,
    hardware_info       JSON
)
""",
"""
CREATE TABLE IF NOT EXISTS executions (
    execution_id            TEXT PRIMARY KEY,
    experiment_id           TEXT NOT NULL,
    task                    TEXT NOT NULL,
    instance_id             TEXT NOT NULL,
    repetition              INTEGER NOT NULL,
    technique               TEXT NOT NULL,
    length                  TEXT NOT NULL,
    format                  TEXT NOT NULL,
    model                   TEXT NOT NULL,
    model_type              TEXT NOT NULL,
    prompt                  TEXT NOT NULL,
    prompt_tokens           INTEGER NOT NULL,
    response                TEXT NOT NULL,
    response_tokens         INTEGER NOT NULL,
    truncated               BOOLEAN NOT NULL DEFAULT 0,
    latency_seconds                 REAL NOT NULL,
    time_to_first_token_seconds     REAL,
    tokens_per_second               REAL,
    energy_wh               REAL NOT NULL,
    energy_wh_low           REAL,
    energy_wh_high          REAL,
    co2_eq_g                REAL NOT NULL,
    measurement_tool        TEXT NOT NULL,
    measurement_mode        TEXT,
    gpu_temp_start_c        REAL,
    gpu_temp_end_c          REAL,
    gpu_temp_avg_c          REAL,
    gpu_power_avg_w         REAL,
    gpu_memory_used_mb      INTEGER,
    cpu_usage_avg_percent   REAL,
    judge_score_total               REAL,
    judge_scores_by_criterion       JSON,
    judge_justifications            JSON,
    judge_global_comment            TEXT,
    judge_validated_by_sonnet       BOOLEAN DEFAULT 0,
    sonnet_score_total              REAL,
    sonnet_scores_by_criterion      JSON,
```

```
    objective_metric_name          TEXT,
    objective_metric_value         REAL,
    timestamp_start        TIMESTAMP NOT NULL,
    timestamp_end          TIMESTAMP NOT NULL,
    FOREIGN KEY (experiment_id) REFERENCES experiments(experiment_id)
)
""",
"CREATE INDEX IF NOT EXISTS idx_executions_cell
ON executions(task, technique, length, format, model)",
"CREATE INDEX IF NOT EXISTS idx_executions_instance
ON executions(task, instance_id)",
"""
CREATE TABLE IF NOT EXISTS metasystem_executions (
    metasystem_id          TEXT PRIMARY KEY,
    experiment_id          TEXT NOT NULL,
    component              TEXT NOT NULL,
    model_used             TEXT NOT NULL,
    input_tokens           INTEGER NOT NULL,
    output_tokens          INTEGER NOT NULL,
    latency_seconds        REAL NOT NULL,
    energy_wh_low          REAL NOT NULL,
    energy_wh_mean         REAL NOT NULL,
    energy_wh_high         REAL NOT NULL,
    co2_eq_g_mean          REAL NOT NULL,
    linked_execution_id    TEXT,
    timestamp_start        TIMESTAMP NOT NULL,
    timestamp_end          TIMESTAMP NOT NULL,
    FOREIGN KEY (experiment_id) REFERENCES experiments(experiment_id),
    FOREIGN KEY (linked_execution_id) REFERENCES executions(execution_id)
)
""",
"CREATE INDEX IF NOT EXISTS idx_metasystem_component
ON metasystem_executions(component)",
"""
CREATE TABLE IF NOT EXISTS failures (
    failure_id             TEXT PRIMARY KEY,
    experiment_id          TEXT NOT NULL,
    task                   TEXT,
    instance_id            TEXT,
    repetition             INTEGER,
    technique              TEXT,
    length                 TEXT,
    format                 TEXT,
```

```
        model                   TEXT,
        component               TEXT,
        failure_type            TEXT NOT NULL,
        error_message           TEXT,
        retry_count             INTEGER NOT NULL DEFAULT 0,
        final_action            TEXT NOT NULL,
        timestamp               TIMESTAMP NOT NULL,
        FOREIGN KEY (experiment_id) REFERENCES experiments(experiment_id)
    )
    """,
    """
    CREATE TABLE IF NOT EXISTS system_baselines (
        baseline_id             TEXT PRIMARY KEY,
        experiment_id           TEXT NOT NULL,
        duration_seconds        INTEGER NOT NULL,
        avg_power_w             REAL NOT NULL,
        energy_wh               REAL NOT NULL,
        gpu_temp_avg_c          REAL,
        cpu_usage_avg_percent   REAL,
        timestamp_start         TIMESTAMP NOT NULL,
        timestamp_end           TIMESTAMP NOT NULL,
        FOREIGN KEY (experiment_id) REFERENCES experiments(experiment_id)
    )
    """,
)
```

Apéndice B

Resultados del modelo de efectos mixtos

Cuadro B.1: Modelo lineal mixto: log(energy_wh) - API

Término	Coef.	SE	z	$P > \|z\|$	[0.025	0.975]
Intercept	0.000	–	–	–	–	–
technique[T.few_shot]	-1.157	0.120	-9.636	0.000	-1.393	-0.922
technique[T.zero_shot]	-1.094	0.120	-9.107	0.000	-1.329	-0.858
length[T.short]	0.064	0.120	0.532	0.595	-0.172	0.299
format[T.xml]	-1.211	0.120	-10.085	0.000	-1.446	-0.976
model[T.gpt-5-mini]	-1.343	0.049	-27.394	0.000	-1.439	-1.247
task[T.humaneval]	0.000	–	–	–	–	–
task[T.xsum]	0.000	–	–	–	–	–
technique[T.few_shot]:length[T.short]	-0.051	0.170	-0.298	0.766	-0.383	0.282
technique[T.zero_shot]:length[T.short]	-0.055	0.170	-0.323	0.747	-0.388	0.278
technique[T.few_shot]:format[T.xml]	0.883	0.170	5.198	0.000	0.550	1.216
technique[T.zero_shot]:format[T.xml]	1.178	0.170	6.934	0.000	0.845	1.510
length[T.short]:format[T.xml]	-0.002	0.170	-0.009	0.993	-0.334	0.331
technique[T.few_shot]:length[T.short]:format[T.xml]	0.258	0.240	1.076	0.282	-0.212	0.729
technique[T.zero_shot]:length[T.short]:format[T.xml]	-0.010	0.240	-0.042	0.967	-0.481	0.461
Group Var	0.000	–	–	–	–	–

Fórmula: `log_energy_wh ~ technique * length * format + model + task`.
Agrupamiento: `instance_id` ($n = 1080$, 15 grupos, tamaño 72). REML; convergido. Escala = 0,6489.

Cuadro B.2: Modelo lineal mixto: log(energy_wh) - API

Término	Coef.	SE	z	$P > \|z\|$	[0.025	0.975]
Intercept	0.000	–	–	–	–	–
technique[T.few_shot]	-1.157	0.120	-9.636	0.000	-1.393	-0.922
technique[T.zero_shot]	-1.094	0.120	-9.107	0.000	-1.329	-0.858
length[T.short]	0.064	0.120	0.532	0.595	-0.172	0.299
format[T.xml]	-1.211	0.120	-10.085	0.000	-1.446	-0.976
model[T.gpt-5-mini]	-1.343	0.049	-27.394	0.000	-1.439	-1.247
task[T.humaneval]	0.000	–	–	–	–	–
task[T.xsum]	0.000	–	–	–	–	–
technique[T.few_shot]:length[T.short]	-0.051	0.170	-0.298	0.766	-0.383	0.282
technique[T.zero_shot]:length[T.short]	-0.055	0.170	-0.323	0.747	-0.388	0.278
technique[T.few_shot]:format[T.xml]	0.883	0.170	5.198	0.000	0.550	1.216
technique[T.zero_shot]:format[T.xml]	1.178	0.170	6.934	0.000	0.845	1.510
length[T.short]:format[T.xml]	-0.002	0.170	-0.009	0.993	-0.334	0.331
technique[T.few_shot]:length[T.short]:format[T.xml]	0.258	0.240	1.076	0.282	-0.212	0.729
technique[T.zero_shot]:length[T.short]:format[T.xml]	-0.010	0.240	-0.042	0.967	-0.481	0.461
Group Var	0.000	–	–	–	–	–

Fórmula: `log_energy_wh ~ technique * length * format + model + task`.
Agrupamiento: `instance_id` ($n = 1080$, 15 grupos, tamaño 72). REML; convergido. Escala = 0,6489.

Cuadro B.3: Modelo lineal mixto: `judge_score_total` - API

Término	Coef.	SE	z	$P > \|z\|$	[0.025	0.975]
Intercept	0.000	–	–	–	–	–
technique[T.few_shot]	-2.002	3.523	-0.568	0.570	-8.907	4.903
technique[T.zero_shot]	-11.229	3.523	-3.187	0.001	-18.134	-4.324
length[T.short]	-0.281	3.523	-0.080	0.937	-7.186	6.625
format[T.xml]	-13.444	3.523	-3.816	0.000	-20.350	-6.539
model[T.gpt-5-mini]	6.641	1.438	4.618	0.000	3.822	9.460
task[T.humaneval]	0.000	–	–	–	–	–
task[T.xsum]	0.000	–	–	–	–	–
technique[T.few_shot]:length[T.short]	3.481	4.982	0.699	0.485	-6.284	13.246
technique[T.zero_shot]:length[T.short]	16.051	4.982	3.222	0.001	6.286	25.817
technique[T.few_shot]:format[T.xml]	1.113	4.982	0.223	0.823	-8.653	10.878
technique[T.zero_shot]:format[T.xml]	15.226	4.982	3.056	0.002	5.461	24.992
length[T.short]:format[T.xml]	3.634	4.982	0.729	0.466	-6.131	13.400
technique[T.few_shot]:length[T.short]:format[T.xml]	7.832	7.046	1.112	0.266	-5.978	21.642
technique[T.zero_shot]:length[T.short]:format[T.xml]	-6.555	7.046	-0.930	0.352	-20.365	7.255
Group Var	0.000	–	–	–	–	–

Fórmula: `judge_score_total ~ technique * length * format + model + task`.
Agrupamiento: `instance_id` ($n = 1080$, 15 grupos, tamaño 72). REML; convergido. Escala = 558,54.

Cuadro B.4: Modelo lineal mixto: `judge_score_total` - LOCAL

Término	Coef.	SE	z	$P > \|z\|$	[0.025	0.975]
Intercept	-0.000	n.d.	n.d.	1.000	n.d.	n.d.
technique[T.few_shot]	-9.188	4.080	-2.252	0.024	-17.185	-1.190
technique[T.zero_shot]	-20.128	4.079	-4.934	0.000	-28.123	-12.133
length[T.short]	-4.137	4.078	-1.014	0.310	-12.129	3.855
format[T.xml]	-9.604	4.079	-2.354	0.019	-17.598	-1.609
model[T.mistral-7b]	-5.964	3.205	-1.861	0.063	-12.245	0.316
model[T.qwen-2.5-14b]	7.843	2.132	3.679	0.000	3.665	12.022
task[T.humaneval]	-0.000	n.d.	n.d.	1.000	n.d.	n.d.
task[T.xsum]	-0.000	n.d.	n.d.	1.000	n.d.	n.d.
technique[T.few_shot]:length[T.short]	13.716	5.768	2.378	0.017	2.411	25.021
technique[T.zero_shot]:length[T.short]	18.937	5.767	3.283	0.001	7.633	30.240
technique[T.few_shot]:format[T.xml]	3.780	5.771	0.655	0.512	-7.531	15.090
technique[T.zero_shot]:format[T.xml]	11.467	5.772	1.987	0.047	0.155	22.780
length[T.short]:format[T.xml]	14.801	5.766	2.567	0.010	3.501	26.102
technique[T.few_shot]:length[T.short]:format[T.xml]	-9.191	8.155	-1.127	0.260	-25.174	6.791
technique[T.zero_shot]:length[T.short]:format[T.xml]	-15.402	8.155	-1.889	0.059	-31.385	0.582
gpu_temp_avg_c	-0.313	0.415	-0.753	0.451	-1.127	0.501
Group Var	0.000	–	–	–	–	–

Fórmula: `judge_score_total ~ technique * length * format + model + task + gpu_temp_avg_c`.
Agrupamiento: `instance_id` ($n = 1620$, 15 grupos, tamaño 108). REML; convergido. Escala = 1121,93.