Universidad Nacional Autónoma de México

Ingeniería en computación

Laboratorio de inteligencia artificial

LLM performance & evaluation app

ALUMNO: Molina Véjar Aarón Gael
318333506

Semestre:
2026-I

México,CDMX. octubre 2025

Índice

1. Introducción

La inteligencia artificial generativa, comúnmente abreviada GenAI, avanza día con día con el objetivo de proporcionar nuevas herramientas para dar solución a distintas problemáticas a través de la inteligencia artificial. Los LLMs son una implementación que cada vez tiene mas auge en cualquier entorno que maneje datos, puesto que una gran cantidad de procesos pueden ser automatizados logrando una eficiencia que evoluciona por completo la forma de enfrentar problemas.

En el laboratorio de inteligencia artificial es algo rutinario enfrentarse a distintos problemas que involucran una gran cantidad de datos, los LLM´s son ideales para lograr implementaciones con alto valor agregado, por lo que es primordial comprender las implementaciones de los modelos LLM de manera técnica, analizar las distintas implicaciones tales como costos, rendimiento, alcance y sostenibilidad a largo plazo. Por esta razón es que esta investigación tiene como objetivo implementar LLM´s para comprender todos estos parámetros y así planificar futuros proyectos dentro del laboratorio que involucren el uso de LLM´s.

2. Marco Teórico

2.1. Large Language Models

Un modelo de lenguaje grande (LLM por sus siglas en inglés), es un modelo de lenguaje probabilístico que calculará una distribución sobre un vocabulario, lo que significa que el modelo de lenguaje conoce un conjunto de palabras llamadas vocabulario, asociadas al contexto de una oración, es decir, el modelo originalmente tomará en cuenta el contexto en el que se encuentra y seleccionará un vocabulario de palabras que encuentre con mayor relación a la oración proporcionada, posteriormente seleccionará la palabra con mayor probabilidad de continuar en la oración y de esta manera será capaz de generar texto. Un LLM no deja de ser un modelo de lenguaje, a pesar de que se le agregue un "large"sigue teniendo el mismo funcionamiento que un simple modelo de lenguaje, esta distinción se refiere al numero de parámetros que contiene el modelo, sin embargo aún no se encuentra definido en que punto exacto un modelo de lenguaje se convierte en un LLM, ni en qué punto un LLM deja de serlo. Incluso existen modelos con pocos parámetros que siguen siendo considerados LLM´s, debido a que no existe una distinción clara respecto al número de parámetros. [6]

2.1.1. Modelos en la nube

Los modelos LLM en la nube son los mas conocidos y utilizados por la gran mayoría de los usuarios, dentro de esta categoría nos encontramos con modelos tales como ChatGPT de OpenAI[9], Claude de Anthropic[1], Grok[11], Gemini de Google[3], Mistral[8] etc. Su principal ventaja es sin duda la potencia que te pueden otorgar pues no dependes de nada mas que de un navegador para hacer uso de estos modelos en su formato de chat, para esto basta con crear una cuenta para cada modelo, o bien asociarla con una cuenta como google, y empezar a hacer uso de los modelos mediante

propmts en formato de chat. Sin embargo, estos modelos en su mayoría poseen distintas restricciones de uso que a menudo van asociadas con un costo, estas restricciones se pueden enfocar a distintos aspectos como lo son el numero de prompts disponibles, el numero de tokens de entrada y el numero de tokens de salida (generalmente este es el parámetro más utilizado), la potencia o inteligencia que puede poseer cada modelo, así como funciones adicionales como lo son las búsquedas masivas, modelos de generación de video, audio, etc.

Tabla 1: Comparación de suscripciones estándar de plataformas de IA

Plataforma	**Suscripción estándar**	**Precio Mensual (USD)**
ChatGPT (OpenAI)	Plus	$20
Claude (Anthropic)	Pro	$17
Grok (xAI)	Premium+ (vía X)	$30
Gemini (Google)	AI Pro	$19.99
Mistral	Pro	$14.99

En la tabla 1 podemos ver algunos datos de los planes de suscripción mas famosos en la época actual, estos precios son muy parecidos a excepción de Grok que muestra un precio casi del doble con respecto a la competencia.

Por otro lado, se ofrecen distintos planes de pago para el uso de APIS de integración de cada modelo, Las API´s de integración son utilizadas para incorporar los LLM en proyectos y aplicaciones empresariales, de uso personal o para fines distintos. En este apartado de igual manera encontramos un rango de precios especificados por cada modelo, de tal manera que podemos obtener una referencia respecto a lo que vamos a utilizar en la integración de dichos modelos.

Tabla 2: Precios del Modelo API de Alto Rendimiento por Millón de Tokens (MTok)

Modelo API	**Input ($/1 MTok)**	**Output ($/1 MTok)**
GPT-5	$1.25	$10.00
Opus (Claude) 4.1	$15.00	$75.00
Grok 4-0709	$3.00	$15.00

2.1.2. Estructuras de Costo API Pay-Per-Token (Evaluación para Desarrolladores)

La industria ha estandarizado el precio para el nivel de acceso premium al consumidor en el rango de aproximadamente $20 USD por mes, ejemplificado por ChatGPT Plus y Claude Pro. En el ámbito empresarial, la segmentación del mercado es más marcada, diferenciando claramente a los proveedores de Cognición de Frontera (modelos que ofrecen la inteligencia más avanzada, como Anthropic Opus) de los proveedores de Eficiencia a Escala (modelos diseñados para el menor costo de transacción, como

Cohere R7B o IBM Granite). Los modelos de frontera imponen tarifas API sustancialmente más altas (ej., $75 por millón de tokens de salida para Opus 4.1), mientras que los modelos de eficiencia compiten agresivamente en el mínimo costo por millón de tokens de entrada (MTok), llegando hasta $0.0375 por MTok

Otro vector de costo crítico es el riesgo de licenciamiento. Ciertos modelos, aunque son accesibles o casi-abiertos (Falcon, Gemma), no cobran por token, sino que imponen barreras de entrada legales y de cumplimiento. Estos riesgos, que incluyen obligaciones de regalías (Falcon) [10] o exclusiones estrictas de uso profesional (Gemma) [2], exigen una diligencia debida considerablemente mayor que el simple cálculo de tokens.

OpenAI: Precios Tokenizados de la Serie GPT-5 y GPT-4o

Tabla 3: Precios del Modelo API de OpenAI por Millón de Tokens (MTok)

Modelo API	Input ($/1 MTok)	Output ($/1 MTok)
GPT-5	$1.25	$10.00
GPT-5 mini	$0.25	$2.00
GPT-5 nano	$0.05	$0.40

Anthropic: Token Economics de Claude (Opus, Sonnet, Haiku)

Tabla 4: Precios del Modelo API de Claude por Millón de Tokens (MTok)

Modelo API	Input ($/1 MTok)	Output ($/1 MTok)
Opus 4.1	$15.00	$75.00
Sonnet 4.5	$3.00	$15.00
Haiku	$1.00	$1.00

Cohere: Análisis de Command R, R+ y Servicios de Rerank/Embedding

Tabla 5: Precios del Modelo API de Cohere por Millón de Tokens (MTok)

Modelo API	Input ($/1 MTok)	Output ($/1 MTok)
Command R	$0.15	$0.60
Command R7B	$0.0375	$0.15
Embed	$0.12	–

Alibaba Cloud Qwen: Tarifas Tokenizadas Regionales y Tiered
E. watsonx (IBM): Precios Pay-Per-Token de los Modelos Granite

2.2. Modelos locales

Por otro lado encontramos un mundo de LLMs que parece contrastar por completo con los modelos cloud en cuestión de precios e implementación. Dentro de los modelos

Tabla 6: Estructura de precios de Alibaba Cloud para modelos Qwen

Modelo	**Contexto**	**Input ($/MTok)**	**Output ($/MTok)**
Qwen-Flash	Internacional (Singapur)	0.05	0.40
Qwen-Max	$<$ 32K tokens	1.20	6.00
	$>$ 128K tokens	3.00	15.00

Tabla 7: Precios del Modelo API de watsonx por Millón de Tokens (MTok)

Modelo API	**Input ($/1 MTok)**	**Output ($/1 MTok)**
granite-8b-code-instruct	$0.60	$0.60
granite-4-h-small	$0.06	$0.25
Embed	$0.12	–

on-permise encontramos modelos que suelen ser en su mayoría versiones de los LLMs mas conocidos en forma reducida, pues encontramos un amplio catalogo de LLMs que cuentan con licencias de código abierto como Apache o MIT, lo que significa que somos completamente libres de utilizar estos modelos en cualquier proyecto, sea de investigación, pruebas o con fines de lucro. A pesar de que a simple vista pueda parecer que esta alternativa es la ideal en cualquier escenario hay que tomar en cuenta un punto muy importante y aquí es donde entra en juego el balance en comparación con modelos cloud, y es que estos modelos se instalan de forma local por lo que es necesario contar con toda la infraestructura física necesaria para implementar estos modelos, y esto implica un importante análisis técnico y económico.

El análisis de modelos LLM locales implica los requerimientos técnicos de funcionamiento, así como las especificaciones propias de cada modelo para conocer su calidad y potencia, además del propósito por el que previamente fueron entrenados en algunos casos. Es por eso que, a pesar de que esta clase de modelos ofrecen ventajas muy destacables como la absoluta privacidad de uso, la potencia ajustable de forma libre y una adaptación a aplicaciones o sistemas mucho más personalizable, su implementación requiere de un análisis igual o incluso superior que la de los modelos tipo SaaS, por lo que es primordial conocer y tomar en cuenta cada una de estas implicaciones.

2.2.1. Benchmarks

Para un primer acercamiento con las anteriores métricas se realizaron algunos benchmarks propuestos para modelos locales proponiendo distintos prompts que representan diferentes objetivos, así como diferentes habilidades para demostración de efectividad de cada LLM. Estas evaluaciones son puramente ilustrativas y no pretenden obtener conclusiones definitivas, ya que este proceso será atuomatizado mas adelante con la aplicación de evaluación & performance comparando estos resultados con LLMs de tipo SaaS.

Comencemos definiendo qué es lo que va a ser evaluado en un modelo específico. Para esto, se tomarán en cuenta aspectos técnicos definidos por la documentación de cada modelo:

1. **Tamaño del modelo en disco**.
2. **Tamaño del modelo (Parámetros):** ¿Qué número de parámetros (3B, 7B, 14B, 70B, etc.) posee cada modelo?
   Generalmente, a mayor número de parámetros, mejora la potencia.
3. **Contexto:** ¿Cuánta información es capaz de procesar a la vez? (4k, 8k, 128k, etc.).
4. **Licencia:** Tipo de licencia.

Nota: La cantidad de memoria RAM/VRAM utilizada se estandariza con la computadora de pruebas utilizada para cada modelo:

- 16 GB de memoria RAM.
- 4 GB de memoria VRAM.

Adicionalmente, se agregarán dos métricas que obtendremos a partir de la experimentación con los modelos:

1. **Capacidad y rendimiento:** Evaluación basada en los distintos prompts con los que se realizarán las pruebas.
2. **Integración a proyectos:** Dentro de las pruebas, en la medida de lo posible, se realizarán implementaciones en proyectos reales.

Prompts a utilizar en la fase de pruebas Es de vital importancia definir una serie de prompts que nos permitan evaluar la capacidad y rendimiento de los modelos. Para esto, se deciden cuatro giros de evaluación:

- Razonamiento: Prompt: "Si tengo 5 manzanas, me quitan 2 y me dan otras 3, ¿cuántas tengo?"
- Instrucción compleja: Prompt: .Escribe un correo electrónico para declinar una invitación a una reunión de forma educada, mencionando que tengo un conflicto de horario con otro proyecto. Máximo 3 oraciones."
- Escritura de código: Prompt: .Escribe una función en Python que calcule la secuencia de Fibonacci."

2.2.2. Fase de pruebas en entornos locales

Embeddinggemma Embeddinggema es un modelo inscrustado de código abierto de ultima generación desarrollado por Google basado en Gemma 3. Embeddinggema produce representaciones vectoriales de texto lo que lo hace adecuado para tareas de busqueda y recuperación, incluida la calsificación, la agrupación en clusteres y la búsqueda de similitud semántica.

Tabla 8: Especificaciones Técnicas: Embeddinggema

Característica	**Valor**
Tamaño del modelo en disco	593 MB
Tamaño del modelo (Parámetros)	300M
Ventana de contexto	2k
Licencia	Apache

Tabla 9: Pruebas de Rendimiento: Embeddinggema

Prompt	**Salida / Observaciones**
Razonamiento (Manzanas)	″
Redacción de Correo	″
Código (Fibonacci)	″

Como observamos la tabla 9 se encuentra completamente vacía debido a que este modelo no es un modelo de chat, a diferencia de los modelos más habituales de interacción con el usuario final, este tipo de modelos tienen su aplicación en técnicas avanzadas de entrenamiento de modelos de lenguaje grandes para mejor precisión y enfoque, basadas en bases de conocimiento.

Qwen3:4b La familia Qwen 3 es un conjunto completo de modelos densos y de mezcla de expertos (MoE). El modelo insignia, Qwen3-235B-A22B, logra resultados competitivos en evaluaciones de referencia de codificación, matemáticas, capacidades generales, etc., en comparación con otros modelos de primer nivel como DeepSeek-R1, o1, o3-mini, Grok-3 y Gemini-2.5-Pro. Además, el pequeño modelo de MoE, Qwen3-30B-A3B, supera a QwQ-32B con 10 veces los parámetros activados, e incluso un modelo pequeño como Qwen3-4B puede rivalizar con el rendimiento de Qwen2.5-72B-Instruct.

Tabla 10: Especificaciones Técnicas: Qwen3:4b

Característica	**Valor**
Tamaño del modelo en disco	2.5 GB
Tamaño del modelo (Parámetros)	4B
Ventana de contexto	3k
Licencia	Apache

Tabla 11: Pruebas de Rendimiento: Qwen3:4b

Prompt	**Salida del Modelo**
Razonamiento (Manzanas)	Si tienes 5 manzanas y te quitan 2, entonces tienes $5-2=3$ manzanas. Luego, si te dan otras 3, sumas $3+3=6$. **(20.8s)**
Redacción de Correo	Estimado [Nombre], Agradezco la invitación a la reunión, pero debido a un conflicto de horario con otro proyecto que estoy desarrollando, no podré asistir. Espero poder coordinar una fecha alternativa... Atentamente, [Nombre]. **(139.9s)**
Código (Fibonacci)	`def fibonacci_sequence(n): if n<=0: return [] ...` **(114.4s)**

Granite3.3:2b Granite es un modelo LLM diseñado para una potente capacidad de razonamiento, uso el razonamiento cuando es necesario y cuando no, y una ventana de contexto optimizada. Granite ofrece documentación soportada por IBM, en la que se encuentran distintos recursos como tutoriales de inicio, cookbooks y playground.

Tabla 12: Especificaciones Técnicas: Granite3.3:2b

Característica	**Valor**
Tamaño del modelo en disco	1.5 GB
Tamaño del modelo (Parámetros)	2B
Ventana de contexto	32k
Licencia	Apache

Tabla 13: Pruebas de Rendimiento: Granite3.3:2b

Prompt	**Salida del Modelo**
Razonamiento (Manzanas)	Después de que te quiten 2 manzanas, tendrás $5 - 2 = 3$ manzanas. Luego, si te dan 3 más, tendrás un total de $3+3 = 6$ manzanas.
Redacción de Correo	Subject: Declinación de la Invitación. Dear [Recipiente], Agradezco mucho tu invitación... Sin embargo, lamento informarte que tengo un compromiso inevitable...
Código (Fibonacci)	`def fibonacci(n): if n<=0: return .Error"...` (Implementa algoritmo iterativo eficiente en memoria).

Deepcode:latest A través de una colaboración conjunta entre el equipo de Agentica y Together AI, DeepCoder-14B-Preview es un modelo de razonamiento de código ajustado a partir de Deepseek-R1-Distilled-Qwen-14B a través de RL distribuido. Logra una impresionante precisión del 60,6 % en LiveCodeBench (+8 % de mejora)

Tabla 14: Especificaciones Técnicas: Deepcoder:latest

Característica	**Valor**
Tamaño del modelo en disco	9.0 GB
Tamaño del modelo (Parámetros)	14B
Ventana de contexto	128k
Licencia	MIT License

Tabla 15: Pruebas de Rendimiento: Deepcoder:latest

Prompt	**Salida del Modelo**
Razonamiento (Manzanas)	Claro, resolvamos paso a paso: 1. Inicial: 5 manzanas. 2. Te quitan 2 ($5 - 2 = 3$). 3. Te dan 3 ($3 + 3 = 6$). Respuesta final: 6.
Redacción de Correo	Asunto: Declinación de Invitación. Estimado [Nombre], Lamentablemente no puedo asistir debido a un conflicto de horario... Agradezco su comprensión.
Código (Fibonacci)	`def fibonacci(n): Çalcula el enésimo número...ïf n==0: return 0...` (Incluye docstrings y manejo de casos base).

Cogito Cogito v1 Preview es una familia de modelos de razonamiento híbrido de Deep Cogito que superan a los mejores modelos abiertos disponibles del mismo tamaño,

incluidas las contrapartes de LLaMA, DeepSeek y Qwen en la mayoría de los puntos de referencia estándar.

Tabla 16: Especificaciones Técnicas: Cogito

Característica	**Valor**
Tamaño del modelo en disco	4.9 GB
Tamaño del modelo (Parámetros)	8B
Ventana de contexto	128k
Licencia	MIT License

Tabla 17: Pruebas de Rendimiento: Cogito

Prompt	**Salida del Modelo**
Razonamiento (Manzanas)	Vamos a resolverlo paso a paso: Tenías 5. Te quitaron 2 ($5-2=3$). Luego te dieron 3 nuevas ($3+3=6$). La respuesta es: 6 manzanas.
Redacción de Correo	Estimado/a [Nombre], Agradezco sinceramente la invitación... Lamentablemente no puedo asistir debido a un compromiso previo...
Código (Fibonacci)	Ofreció dos formas: 1. Bucle for (más simple). 2. Recursividad (más elegante pero menos eficiente). `def fibonacci(n): if n<=0: return "Num inválido"...`

2.3. Métricas de evaluación de LLMs

La evaluación del rendimiento de los LLMs en entornos de producción debe ir más allá del rendimiento total, descomponiéndose en distintas métricas fundamentales que impactan directa o indirectamente en la experiencia del usuario:

2.3.1. Métricas de eficiencia

Como primer métrica de evaluación para la implementación de un LLM nos encontramos con la eficiencia, Liang et al. en HELM [7] propone las siguientes distinciones que nos serán útiles:

- Eficiencia de entrenamiento: midiendo el costo en horas de GPU y emiciones de CO2, la métrica clave en nuestro caso será el kWh
- Eficiencia en dólares: para el caso de modelos SaaS la métrica clave serán los tokens vs el precio de la API

2.3.2. Métricas de latencia e inferencia

Liang [7] también propone una forma estandarizada de medir la latencia que vería segúin el tráfico de internet:

Inferencia especializada: se descompone el tiempo de respuesta en dos variables

- Tiempo de codificación
- Tiempo de generación

2.3.3. Métricas de calidad

Además de costo-velocidad, Liang [7] menciona una métrica importante: ¿Qué tan bien responde?

- Exactitud: ¿La respuesta es igual a la esperada?
- Calibración: ¿El modelo sabe cuando está equivocado?
- Toxicidad y sesgos: Métrica importante en entornos críticos.

2.4. Groq

Groq es un desarrollador de inferencia de IA que se centra en agilizar todo el proceso de inteligencia artificial mediante procesadores específicamente diseñados para ejecutar LLMs sobre ellos, denominados como LPU[4], estos procesadores tienen la característica fundamental de agilizar en gran medida procesos de álgebra lineal, pilares en la ejecución de modelos de lenguaje grandes. Las LPU superan en gran medida el coste energético a las GPUs que son comúnmente usadas en el desarrollo e implementación de LLMs, Groq ha diseñado su propio procesador que convierte este proceso en un proceso muy más barato y eficiente, actualmente con una tecnología de 14nm con miras a desarrollar tecnología de 4nm.[4]

Groq se analizó como una herramienta a implementar en este proyecto debido a su capa gratuita que abarca distintos modelos, permitiendo así comparar con un mayor margen de modelos de distintos tamaños, pudiendo así comparar de igual manera con modelos locales que son posibles para su respectiva comparativa y evaluación. A continuación se analizan distintos puntos clave que fueron altamente influyentes en la incoproración de Groq al proyecto.

2.4.1. Conceptos de básicos diseño de las LPU

Las LPU son chips diseñado específicamente para el procesamiento de consultas a LLMs acelerando considerablemente operaciones matriciales, fundamentales para el funcionamiento de un LLM y que posteriormente se convierte en respuestas a prompts, haciendo así más eficiente y barato el uso de modelos de lenguaje grandes, proporcionando la posibilidad de probar distintos modelos sobre esta infraestructura física mediante un servicio SaaS.[4]

El servicio SaaS que proporciona Groq se convierte en su capa de software diseñado para el uso de las LPU para un alto rendimiento y acoplamiento mediante servicios a proyectos y plataformas de distintos alcances, modernizando y agilizando de esta manera el uso de LLMs en proyectos de manera altamente costeable.

Software primero: La arquitectura de Groq se encuentra basada en un enfoque "Software-frist", buscando agilizar el trabajo del desarrollador para maximizar el uso del hardware y lograr colocar el mayor numero de controladores en manos del desarrollador.[4] Esto significa que el desarrollo del sistema Groq comenzó con la capa de software y no directamente con el chip, adaptando todas las necesidades de manera directa al hardware, maximizando así el rendimiento final.

Arquitectura de linea de ensamblaje programable: Una LPU utiliza una linea de ensamblaje programable que se centra en mover instrucciones y datos entre unidades funcionales SIMD, esta característica es la definitoria de una LPU, ya que se centra en aumentar el rendimiento de cada uno de sus componentes mediante la capa de software debido a que su objetivo es aislar la capa de hardware y contrasta sustancialmente con el funcionamiento de una GPU.[4]

Computación y redes deterministas: Para que la linea de montaje cumpla su objetivo es necesario asegurar que la computación sea determinista para así solucionar problemas de competencia por recursos. El flujo de datos se programa estéticamente durante la compilación y se ejecuta de la misma manera cada vez.[4]

Memoria On-Chip: Las LPUs integran tanto la memoria como el procesador en el mismo chip, esto mejora notoriamente la velocidad de almacenamiento y la recuperación de datos.La SRAM del chip de Groq presenta características del nivel de 80 terabytes/segundo de ancho de banda.[4]

2.5. Persistencia de datos para una LLM App Evaluation & Performance

La consideración de almacenar datos para su constante uso en una aplicación o plataforma de evaluación de LLMs, son esenciales debido al escenario en el que se desenvuelven, e incluso al objetivo planteado para dicha plataforma: ahorrar costos y recursos computacionales son factores clave que serán cubiertos en gran medida mediante una persistencia de datos, ya sea con el uso o implementación de una base de datos, como con un sistema de archivos básico que proporcione la capacidad de acceder a registros anteriores relacionando los prompts con los modelos elegidos para su implementación.

La evaluación de exactitud de un LLM es un tanto subjetiva y circunstancial,

no existe una forma estandarizada de medirla más que mediante prueba y error. Sin embargo, estos registros pueden ser de utilidad si se usan de la manera adecuada, incorporando las siguientes métricas que definen de que manera esto puede ser implementado correctamente.

3. Desarrollo

3.1. Identificación de requerimientos

La necesidad de una plataforma que aumente la eficiencia en la tarea de evaluación de modelos de lenguaje grandes es necesaria para observar de cerca y con métricas precisas el rendimiento y costo de uso de un LLM, esto debido a la gran variedad existente y la incertidumbre que representa su uso. El hecho de que los LLMs sean entrenados con datasets de distintos ámbitos para economizar gastos y no recurrir a un LLM lo suficientemente poderoso para adaptarse a cualquier rubro, representa una técnica fundamental en el uso y aplicación de LLMs para una tarea en especifico buscando maximizar su rendimiento de la manera más inteligente y basada en datos posible.

1. Elección de distintos LLM para su comparación
2. Despliegue de respuestas de cada LLM asociadas con su respectivo modelo
3. Información de tamaño y nombre de cada modelo a utilizar
4. Facilidad para ingreso de prompts
5. Información de tiempo de respuesta, tokens y costo de cada modelo
6. Generación de métricas para la evaluación de costos
7. Persistencia de datos para la evaluación de exactitud

3.2. Arquitectura de la LLM app

La aplicación de evaluación se tiene que diseñar en base a los requerimientos identificados previamente, aunque en el transcurso de la recaudación de información fueron modificándose o descartándose muchos de ellos por cuestiones de alcance. A pesar de las distintas vertientes y direcciones que puede tomar el proyecto es importante plantear una base solida que permita una escalabilidad robusta sin dejar de lado la facilidad para resolver distintas problemáticas sin afectar a componentes que no intervengan en su funcionamiento.

La arquitectura enfrente distintas problemáticas que se interponen al objetivo principal, por lo cual es necesario acudir a herramientas o bibliotecas externas que permitan desacoplar responsabilidades dentro de la misma y que de esta manera se convierta en una implementación limpia, fácil de depurar y altamente escalable. A

continuación se enunciarán las distintas problemáticas y la decisión de diseño tomada para enfrentarlas:

- **Incorporación de distintos modelos LLM**: Manejar distintos modelos aumenta la complejidad de manejo y administración de los mismos, pues requiere de un análisis detallado de modelos a utilizar con sus respectivos costos y/o especificaciones. Este requerimiento se ve cubierto por la capa de modelos mediante Ollama, Groq y Langchain.
- **Backend aislado de la interfaz para programar métricas de evaluación**: Es necesario aislar las métricas de programación para modificar y realizar pruebas de métricas de evaluación mediante funciones que sean faciles de adaptar a cualquier ámbito.
- **Frontend aislado para la incorporación de una interfaz limpia y funcional**: Se requiere de una interfaz limpia y funcional por lo que debe estar completamente aislada del manejo de LLMs y de la lógica de evaluación de métricas.
- **Acceso a distintas opciones de infraestructura**: La evaluación de distintos modelos requiere evaluarse en distintas infraestructuras para un mejor panorama de rendimiento y costos.

3.3. Desarrollo backend y aplicación de métricas de rendimiento

Evaluación de costo mediante modelos SaaS

$$Costo = \left(\frac{Total\_Tokens}{1,000,000}\right) \times Precio_{Modelo}$$

Con esta formula podemos calcular de manera básica el costo de nuestras consultas a un LLM en especifico, sin embargo, su uso está limitado a obtener un valor de token para poder cuantificar dicho parámetro. Especialmente los LLMs con los que accedemos día a día mediante APIs o interfaces web como ChatGPT o Claude son escenarios en los que fácilmente podemos obtener esta información mediante librerías de monitoreo, o de las mismas APIs de cada modelo en especifico. En el caso de nuestra plataforma, al hacer uso de Groq, nos agiliza y proporciona esta información de manera rápida y precisa, cualidad que nos conviene enormemente.

Evaluación de costo de modelos locales mediante Ollama sobre AWS

$$Costo = Tiempo_{(s)} \times Costo_{Infra/s}$$

Nuestra siguiente formula refleja una simulación de lo que sería en realidad una estimación de costo a partir de tiempo de uso, y no a numero de tokens. Esto debido a

las tarifas impuestas sobre servicios de maquinas virtuales donde se ejecutaría nuestro modelo. Evidentemente con esta expresión podemos relacionar la tarifa impuesta por un proveedor de servicios en la nube, como en este caso el costo de la tarifa de AWS que es de $0.0052, costo obtenido mediante la calculadora proporcionada por AWS EC2 2025, al mismo tiempo que ejecutamos un modelo sobre esta infraestructura. Esto nos es útil debido a que podríamos gestionar modelos LLM de manera optimizada considerando ausencia de Hardware como infraestructura propia, rentando así alguna máquina virtual que nos permita ejecutar modelos de una forma más eficiente, pero probablemente a un mayor costo.

Evaluación de costo mediante estimación de uso por infraestructura local

$$C_{Amortizacin} = \frac{Costo\_Total\_Hardware}{Vida\_til\_en\_Segundos}$$

$$C_{Energa} = \left(\frac{Watts}{1000}\right) \times \left(\frac{Tiempo\_sec}{3600}\right) \times Precio\_kWh$$

Se proponen dos expresiones que nos ayuden a calcular nuestros gastos por ejecutar un modelo en una computadora local. Para esto es necesario conocer el costo de de nuestro equipo y su vida útil estimada en años, que posteriormente la plataforma convertirá en segundos para la implementación de nuestra expresión.

Adicionalmente se propone una expresión que nos ayude a calcular nuestro consumo en kWh para obtener un aproximado de uso de energía eléctrica. Al final obtendremos un costo estimado que contempla estos dos valores, sin embargo es importante aclarar que esta estimación de costo es una estimación propuesta que no cubre el alcance del proyecto actual, por lo que se maneja como una opción a considerar para una futura actualización.

3.4. Diseño de interfaz de usuario

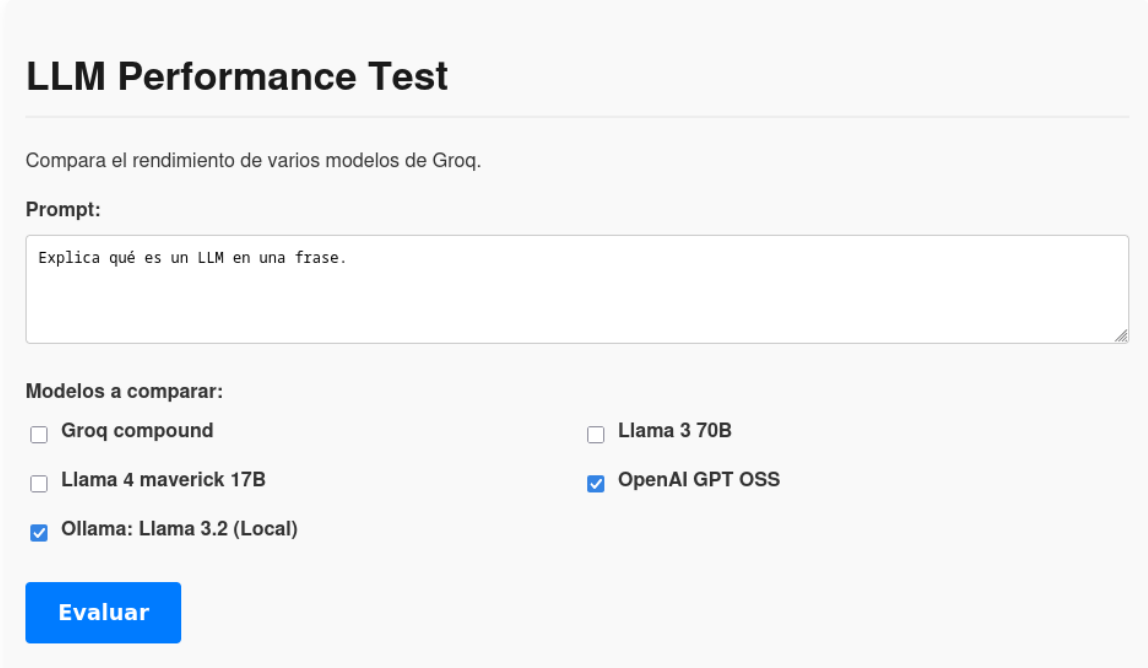

Figura 1: Interfaz principal

Para la interfaz principal podemos observar directamente un prompt el cual nos permita ingresar nuestra entrada en formato de texto, podemos comenzar a realizar evaluaciones que abarquen cualquier tema que nos sea de interés.

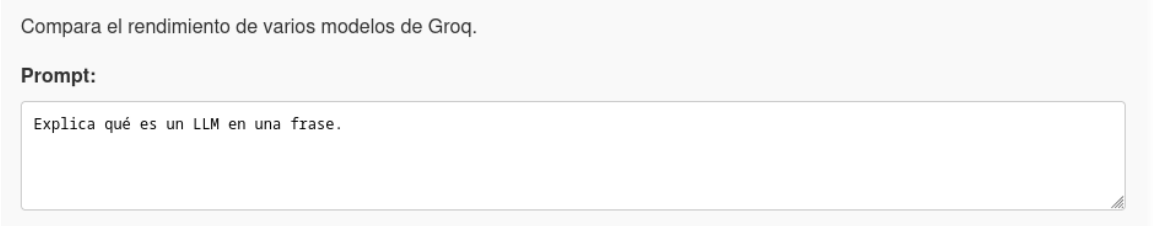

Figura 2: Sección de entradas

Por defecto se presenta una explicación de prueba que nos permita ejecutar la comparación de distintos modelos. Ëxplica que es un LLM en una sola fraseïos arroja salidas lo suficientemente simples e ilustrativas para comenzar a explorar la funcionalidad del proyecto.

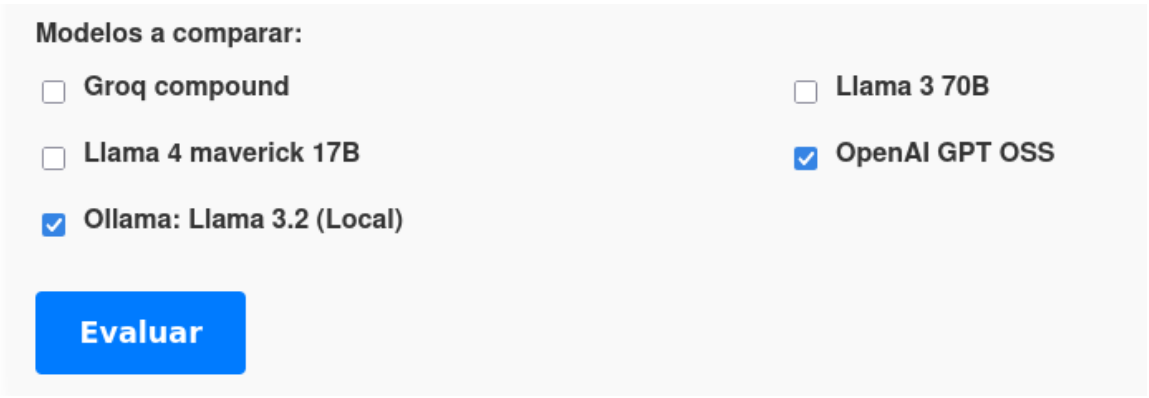

Figura 3: Selección de modelos

En esta sección podemos seleccionar el modelo de nuestro interés, se compone principalmente de modelos proporcionados por Groq para utilizar su inferencia, adicionalmente tenemos la opción de probar un modelo local pequeño mediante Ollama.

Figura 4: Resultados e información

Nuestros resultados se verán reflejados mediante una tabla que contiene información de interés para nuestras evaluaciones, el alcance actual de nuestro proyecto nos permite observar una simulación de inferencia de implementación de modelos en nubes publicas (AWS en este caso) cuyo nombre del modelo se encuentra representado en color morado, además de la implementación de inferencia mediante Groq que se representa en color azul.

3.5. Diseño e implementación de experimentos

Para evaluar el funcionamiento de nuestra plataforma comenzaremos a diseñar experimentos que abarquen distintas áreas y enfoques de pensamiento para simular

una evaluación de LLMs en un entorno real de investigación, asistencia profesional, conversaciones con atención a cliente, resúmenes de temas de interés y programación de software.

3.5.1. Experimento 1: Cruce de rentabilidad

El objetivo de este experimento es comprobar si existe algún punto en el que el costo de los modelos (local vs Groq) es afectado en función del numero de tokens, por ejemplo, si en algún punto resulta más caro el uso de un LLM de Groq en comparación a una ejecución local o viceversa.

Metodología: Para este experimento es importante el numero de tokens, que previamente desconocemos el valor aproximado de cada uno de ellos. Para esto comenzaremos a hacer uso de nuestra aplicación siguiendo el siguiente flujo de trabajo:

1. Planteamiento de los prompts: plantearemos cuatro prompts de distinta longitud y complejidad de tarea
2. Registro de tokens: para las cuatro tareas registraremos el numero de prompts promedio que registra nuestra salida
3. Iteración y registro de costo: realizaremos tres pruebas con los mismos prompts para obtener datos más robustos respecto a los datos de costos
4. Graficación de resultados: por ultimo realzaremos una gráfica con python para la obtención de resultados visuales y de esta manera poder concluir.

Modelo a evaluar con Groq: meta-llama/llama4-maverick-17b-128e-instruct
Modelo a evaluar con Ollama: llama3.2

Tabla 18: Prompts de evaluación

ID prompt	Prompt
PS01	¿Cuál es la capital de Francia? Responde con una sola palabra y sin signos de puntuación
PM02	Explica con máximo 50 palabras para que es utilizado comunmente el lenguaje de programación python.
PH03	Escribe un ensayo comparando las redes de datos usadas antes del 2010 contra las redes de datos actuales, necesito que redactes una introducción, características principales de cada una y por último una conclusión respecto a los cambios más destacados
PXH04	Requiero un código de programación en C# para consola que atienda a los siguientes requerimientos considerando usar patrones de diseño avanzados para el máximo rendimiento y buenas prácticas de programación: 1- Se trata de una tienda de pasteles, 2- El programa debe ser capaz de registrar ventas de los artículos registrados previamente en una lista, 3- Debe de calcular los el costo de los ingredientes también previamente registrados en base a recetas que tendrás que proponer (3 recetas), 4- Deberá generar tickets para cada orden solicitada

Tabla 19: Cantidad de tokens obtenidos por cada prompt para Groq

ID prompt	N. Tokens E.	N. Tokens S.
PS01	41	3
PM02	41	49
PH03	68	850
PXH04	134	2232

Tabla 20: Cantidad de tokens obtenidos por cada prompt para Ollama

ID prompt	N. Tokens E.	N. Tokens S.
PS01	56	3
PM02	54	75
PH03	85	732
PXH04	166	1280

Iteración 1

Tabla 21: Costo para casa prompt para la iteración 1

ID prompt	Costo Groq	Costo Ollama
PS01	$0.000004	$0.000812
PM02	$0.000009	$0.001687
PH03	$0.000092	$0.017152
PXH04	$0.000237	$0.033429

Iteración 2

Tabla 22: Costo para casa prompt para la iteración 2

ID prompt	Costo Groq	Costo Ollama
PS01	$0.000004	$0.000531
PM02	$0.000009	$0.002559
PH03	$0.000074	$0.019492
PXH04	$0.000209	$0.031899

Iteración 3

Tabla 23: Costo para casa prompt para la iteración 3

ID prompt	Costo Groq	Costo Ollama
PS01	$0.000004	$0.000193
PM02	$0.000009	$0.002516
PH03	$0.000091	$0.018964
PXH04	$0.000218	$0.031565

Gráfica obtenida

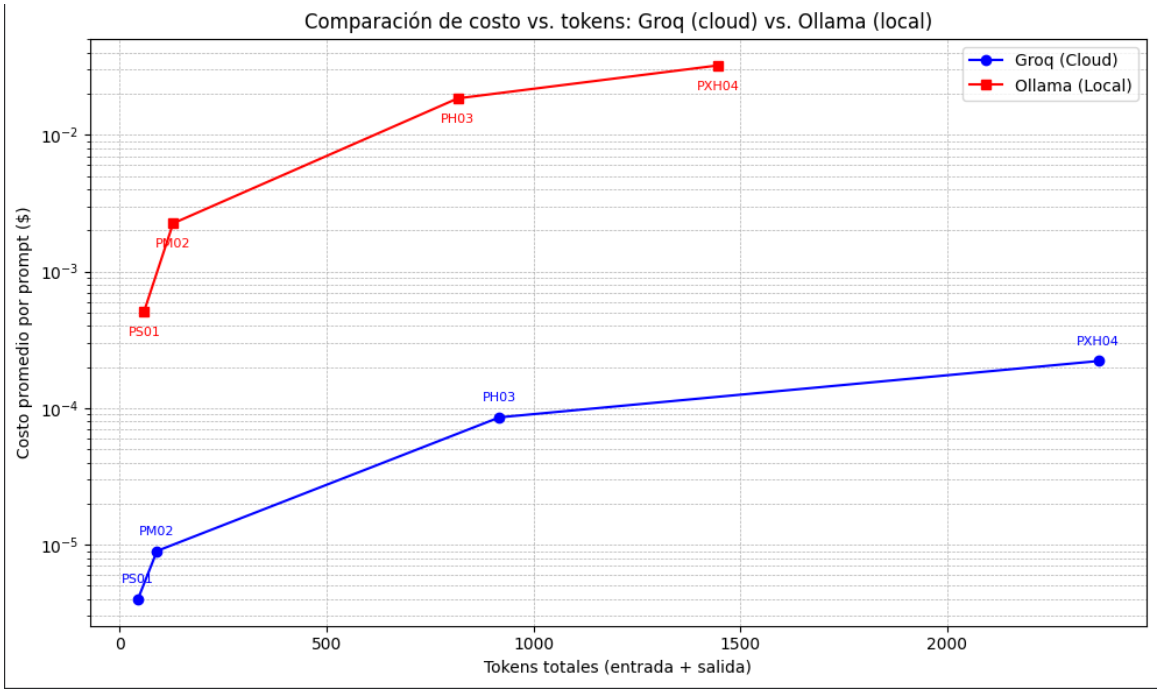

Figura 5: Gráfica obtenida a partir de los experimentos

3.5.2. Experimento 2: Exactitud y seguimiento de instrucciones

Para nuestro segundo experimento evaluaremos dos métricas sumamente importantes a la hora de trabajar con LLMs, ¿Qué tan bien responden y qué tan bien nos hacen caso?

Para esto formularemos cinco prompts distintos que nos permitan obtener una salida en concreto y de esta manera evaluar si nos está respondiendo correctamente y si realizó exactamente la tarea que le solicitamos o decidió alucinar en su respuesta

Modelo a evaluar con Groq: groq/compound

Modelo a evaluar con Ollama: llama3.2

Prompt 1: Pregunta especifica con restricción de formato

¿Cual es el país más grande del mundo? Responde con una sola palabra y sin signos de puntuación

Respuesta esperada: Rusia

Prompt 2: Lógica matemática exacta

¿Cuál de las siguientes oraciones es verdadera y cuál es falsa? Responde unicamente con los incisos de las oraciones verdaderas, separadas por comas y sin paréntesis u otro símbolo.

a) Los únicos enteros positivos que dividen a 7 son 1 y el mismo 7.

b) Alfred Hitchcock ganó un premio de la Academia en 1940 por la dirección de "Rebeca".

c) Para todo entero positivo n, existe un número primo mayor que n.

d) La Tierra es el único planeta en el universo que tiene vida

e) Compra dos boletos para el concierto de rock “Unhinged Universe” del viernes. [5]

Respuesta esperada: a, c

Prompt 3: Conversión de formato

Convierte la siguiente oración de tal forma que quede sin espacios y con todas las letras en minúsculas: Hola mi nombre es Aarón. Devuelve unicamente la oración en el formato solicitado

Respuesta esperada: holaminombreesaarón

Prompt 4: Conocimiento de un área de interés en especifico y decisión booleana

De acuerdo con la siguiente oración: Html es un lenguaje de programación estándar en la industria de los videojuegos. Responde si es verdadera o falsa con una sola palabra, sin signos de puntuación ni otra palabra.

Prompt 5: Secuencia matemática

Devuelve únicamente el número que se encuentra en la quinta posición de la sucesión de Fibonacci, asumiendo que el primer número de la secuencia es 0.

Respuesta esperada: 3

Tabla 24: Tabla de evaluación

Prompt	**Groq (compound)**	**Ollama (llama 3.2)**
1	Incorrecto: el modelo alucinó dando datos de más	Correcto
2	Incorrecto: el modelo arrojó una respuesta extensa y se equivocó en los incisos	Correcto
3	Correcto	Correcto
4	Incorrecto: el modelo dio explicaciones, las cuales no fueron solicitadas	Correcto
5	Incorrecto: el modelo dio explicaciones no solicitadas	Incorrecto: el modelo dio explicaciones no solicitadas

3.6. Despliegue y seguridad

Se exploraron diversas opciones de despliegue para realizar experimentos de una forma agilizada, especialmente usando a la nube como plataforma base, sin embargo, en el camino de implementación nos encontramos con distintos obstáculos, de los cuales valen la pena destacar por lo menos dos puntos a considerar:

1. Costos de despliegue

Para los costos de despliegue se consideran las tarifas impuestas por AWS, plataforma en la que se realizaron las respectivas pruebas de alojamiento (Bucket) y servidor frontend (E3)

2. Seguridad

   En el apartado de seguridad recae un riesgo muy importante relacionado a los costos de despliegue, debido a que es requerido una configuración exhaustiva dentro de las configuraciones de nuestros servidores cloud, debido a esto, la capa de implementación mediante esta vía resulta fuera del alcance de la versión del proyecto actual.

3.7. Despliegue mediante máquina virtual

El desarrollo de la aplicación se realizo en un ambiente totalmente aislado dentro de una máquina virtual VMWare Workstation en base a un sistema operativo Debian 13.0, se realizaron las configuraciones pertinentes para una máquina virtual con el objetivo de optimizar de la manera más eficiente la ejecución de nuestra aplicación, considerando tanto el despliegue local del servidor frontend, así como la ejecución de un LLM pequeño. Todo esto sin riesgo de compatibilidad, logrando así una portabilidad conveniente para esta fase de la implementación.

4. Resultados

4.1. Análisis de resultados: Experimento 1

A partir de los resultados obtenidos mediante el registro de valores de tokens-costo para cada infraestructura, podemos observar una clara diferencia entre cada uno de los prompts propuestos. En el caso de la gráfica obtenida, podemos afirmar que la disparidad es tan evidente como significativa, pues resulta varias ordenes de magnitud superior la curva representada por el costo de LLMs ejecutados en unas máquina virtual de AWS (simulación), dando como significado un precio superior para el caso de la inferencia de forma local dentro de una nube. Por otro lado, los costos de uso de la API de Groq rultan sumamente bajos en comparación a los modelos locales.

El experimentó demostró que mantener una instancia de CPU encendida para un volumen de consultas bajo resulta sumamente más costoso que el hacer uso de servicios en la nube con arquitectura serverless.

4.2. Análisis de resultados: Experimento 2

Los resultados del experimento 2, con objetivo de obtener resultados con respecto a la precisión y seguimiento de instrucciones a modelos pequeños, obtenemos una clara evidencia respecto al tamaño del modelo, en estos dos registros obtenidos podemos observar como la ventana de contexto del modelo ejecutado en Ollama (llama 3.2) presenta un mejor resultado respecto a las instrucciones diseñadas para un análisis de

problemas de matemáticas discretas, con el fin de observar comportamientos de respuestas a problemas lógicos, obteniendo no solo resultados correctos a los problemas, sino que también un mejor seguimiento de formato para la salida.

Los resultados para el experimento 2 resultaron interesantes, sin embargo, esto no depende totalmente de la infraestructura por la que fueron ejecutados ambos modelos de comparación, sino que en si mismos, los modelos representaron un claro factor para los resultados, esto se puede comprobar ejecutando distintos modelos a compound mediante la API de Groq, tales como maverick o openai oss, obteniendo un mejor resultado para estos últimos.

Para obtener un panorama amplio, aprovechando todas las funciones de nuestra aplicación, es posible incoroporar métricas adicionales, tales como la latencia y el costo, así como evaluar otros modelos tanto de forma local mediante Ollama como distintos modelos proporcionados por la API de Groq, mismas que podríamos implementar en un futuro tomando como base los mismos prompts diseñados que, como hemos analizado, funcionan para obtener estos resultados tan significativos y dispares en comportamiento de exactitud.

5. Conclusiones

A lo largo del desarrollo de la aplicación LLM App Evaluation & Performance, se encontraron distintas características que hacen a los LLMs una entidad compleja, tanto de diseñar, como de implementar y evaluar. Sin embargo, esta plataforma sirve perfectamente como base para implementar diversos experimentos que puedan evaluar un grupo de LLMs enfocados en cierta tarea, área, o sistema. De igual manera, fue posible abarcar algunas de las principales métricas de evaluación que añaden parámetros de evaluación aún más completos de manera estandarizada, considerando los distintos modelos de pago por uso de LLMs que varían dependiendo el proveedor, o de su implementación en distintos tipos de infraestructuras.

El alcance temporal del proyecto limita en gran medida las pruebas que se pueden llegar a ejecutar con la plataforma, así cómo la implementación de métricas aún más especificas, diversos tipos de servicios cloud, tipos de modelos, y tipos de escenarios o prompts enfocados a un resultado en especifico. De esta manera, podemos concluir que obtenemos una base sólida de evaluación, con resultados que demuestran el comportamiento de LLMs a distintos factores que interfieren en sus resultados. Con la posibilidad de expandirse con distintas vertientes, haciendo uso de un proyecto con suficiente robustez para la implementación de herramientas, conceptos y bibliotecas de uso moderno, tales como LangSmith, Prompt Engineering, y Agentes AI.

Referencias

[1] Anthropic. *Plans & Pricing for Claude*. `https://claude.com/pricing`. Accedido: 2026-02-17. Incluye información de precios para los planes Free, Pro, Max, Team y Enterprise, y tarifas de API para los modelos Opus, Sonnet y Haiku. Feb. de 2025.

[2] Google. *Gemma Terms of Use*. Términos de uso para el modelo Gemma de Google. 2024. URL: https://ai.google.dev/gemma/terms (visitado 17-10-2024).

[3] Google. *Google AI Pro & Ultra: get access to Gemini 3 Pro & more*. https://gemini.google/subscriptions/. Accedido: 2026-02-17. Detalla los planes de suscripción Google AI Free, Plus, Pro y Ultra, incluyendo características, almacenamiento y créditos de IA. Feb. de 2025.

[4] Groq, Inc. *Groq Thoughts: What is an LPU?* White paper. Descripción de los principios de diseño del Procesador de Unidad Lineal (LPU) de Groq para inferencia de IA. Groq, Inc., 2025. URL: https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/ (visitado 17-02-2026).

[5] Richard Johnsonbaugh. *Matemáticas discretas*. Trad. por Marcia Aída González Osuna. 6.ª ed. Pearson Educación, 2005. ISBN: 970-26-0637-3.

[6] Ari Kobren. *Introduction to Large Language Models*. Clase de curso en línea. Oracle University. 2025. URL: https://mylearn.oracle.com/ou/course/oracle-cloud-infrastructure-generative-ai-professional/147932/213425.

[7] Percy Liang et al. "Holistic Evaluation of Language Models". En: *arXiv preprint* arXiv:2211.09110 (nov. de 2022). Presentado en el Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS 2023) Track on Datasets and Benchmarks. URL: https://arxiv.org/abs/2211.09110 (visitado 16-02-2026).

[8] Mistral AI. *API Pricing*. Estructura de precios para la API de Mistral AI. 2024. URL: https://mistral.ai/pricing#api-pricing (visitado 17-10-2024).

[9] OpenAI. *What is ChatGPT Plus?* Descripción del servicio ChatGPT Plus. 2024. URL: https://help.openai.com/en/articles/6950777-what-is-chatgpt-plus (visitado 17-10-2024).

[10] Technology Innovation Institute. *Terms and Conditions*. Términos y condiciones para el uso de Falcon LLM. 2023. URL: https://falconllm.tii.ae/terms-and-conditions.html (visitado 17-10-2024).

[11] xAI. *Models and Pricing*. https://docs.x.ai/developers/models. Accedido: 2026-02-17. Documentación oficial de xAI que detalla los modelos Grok (incluyendo Grok 4.1 Fast), sus capacidades, contexto, y una tabla completa de precios por millón de tokens, herramientas y API por lotes. Feb. de 2025.