UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Revista Vínculos

https://revistas.udistrital.edu.co/index.php/vinculos/

Revista VÍNCULOS
Ciencia, Tecnología y Sociedad
Un Enlace hacia el Futuro

Recibido: 4 de febrero 2026 / Aceptado: 18 de mayo 2026

EVALUACIÓN COMPARATIVA DE LLMS EJECUTADOS EN ENTORNO LOCAL (OLLAMA) PARA RAG ACADÉMICO: PRECISIÓN, LATENCIA Y TASA DE GENERACIÓN

COMPARATIVE EVALUATION OF LOCAL LLMS (OLLAMA) FOR ACADEMIC RAG: ACCURACY, LATENCY AND GENERATION RATE

Noel Merino Peralta[1], Dr. Rene Edmundo Cuevas Valencia[2], Dr. Angelino Feliciano Morales[3]

Resumen:

Este artículo compara el rendimiento de varios modelos de lenguaje ejecutados localmente con Ollama, dentro de un sistema RAG para asistencia académica. Se evaluaron seis modelos abiertos entre ellos LLaMA, Qwen y un modelo propio DeepSeek-R1 en tareas que iban desde responder preguntas y resolver cálculos, hasta interpretar código y generar texto académico, las métricas clave fueron precisión, latencia (tiempo al primer token) y velocidad de generación. Los hallazgos muestran un juego de equilibrios: los modelos pequeños responden más rápido, pero se equivocan más; los grandes aciertan más, aunque tardan. Qwen destacó con 87.5% de aciertos, sobre todo en consultas de conocimiento, mientras que DeepSeek-R1:8B logró un balance interesante entre rapidez y razonamiento. En conjunto, la investigación demuestra que combinar modelos locales, asignando cada uno a la etapa en la que brilla, puede ofrecer asistentes académicos eficientes, independientes de la nube y accesibles para más personas.

Palabras claves: modelos de lenguaje grandes, recuperación de información, generación de lenguaje, precisión, latencia, rendimiento

Abstract

This article compares the performance of several language models run locally with Ollama, within a RAG system for academic assistance. Six open models were evaluated, including Ollama, Qwen, and a proprietary DeepSeek-R1 model, on tasks ranging from answering questions and solving calculations to interpreting code and generating academic text. Key metrics were accuracy, latency (time to first token), and generation speed. The findings show a trade-off: smaller models respond faster but make more mistakes; larger ones are more accurate, although they take longer. Qwen excelled with 87.5% accuracy, especially on knowledge queries, while DeepSeek-R1:8B achieved an interesting balance between speed and reasoning. Overall, the research demonstrates that combining local models, assigning each one to the stage at which it excels, can deliver efficient, cloud-independent, and accessible academic assistants.

Keywords: large language models, information retrieval, language generation, accuracy, latency, performance

1 Licenciatura En Ingeniería En Computación, Universidad Autónoma De Guerrero, México, Chilpancingo Guerrero. Afiliación institucional: Universidad/Organización/Empresa, País. Correo electrónico personal e institucional e-mail: noelmerino7@gmail.com, 19306462@uagro.mx ORCID https://orcid.org/0009-0009-48471964

2 Doctorado en enseñanza superior, Afiliación institucional: Universidad Autónoma De Guerrero. Correo electrónico personal e institucional e-mail: ORCID: https://orcid.org/0000-0001-9528-7603

3 Doctorado en tecnología educativa, México. Afiliación institucional: Universidad Autónoma De Guerrero. Correo electrónico personal e institucional e-mail: ORCID: https://orcid.org/0000-0002-7707-7319

1. Introducción

Redactar documentos académicos o profesionales exige tiempo, esfuerzo y un conocimiento profundo del tema, no basta con escribir bien: hay que ordenar ideas con precisión, respaldarlas con evidencias y presentarlas con claridad, aunque los procesadores de texto y correctores ortográficos facilitan el trabajo, no cubren todas las necesidades, ninguno integra de forma completa la búsqueda rápida de fuentes fiables, la adaptación del estilo y la generación automática de contenido especializado. [2] Por eso, investigadores, docentes y estudiantes siguen invirtiendo largas horas revisando información dispersa, extrayendo datos y armando narrativas coherentes. [1] El resultado: retrasos en la producción de conocimiento y, muchas veces, un desgaste innecesario.

En este panorama, las técnicas de Generación Aumentada por Recuperación o RAG, por sus siglas en inglés se perfilan como una solución prometedora. En esencia, RAG combina dos mundos: el de los sistemas de recuperación de información y el de los modelos de lenguaje (LLM). El primero busca en repositorios externos documentos relevantes; el segundo, sintetiza respuestas apoyándose en esas fuentes, produciendo textos más contextualizados, actualizados y respaldados por evidencia. [3] Frente a los LLM tradicionales entrenados únicamente con datos estáticos que se vuelven obsoletos y pueden "alucinar" información, RAG introduce una dinámica que minimiza errores factuales y mejora la precisión. [1]

Un punto crítico para materializar esta visión es seleccionar los modelos de lenguaje adecuados. [10] Dado que la implementación será local por motivos de privacidad, costo y accesibilidad, se evaluarán modelos open-source de distintas arquitecturas y tamaños. [3] El reto es encontrar el equilibrio entre precisión y velocidad. Un modelo muy preciso, pero lento entorpece la interacción; uno veloz pero inexacta pierde la confianza del usuario.

Para resolver esta ecuación, el trabajo incluye una evaluación comparativa de varios LLM de código abierto ejecutados con Ollama, una plataforma que facilita el despliegue local. [7] [4] Las métricas clave son tres: precisión en tareas académicas, latencia medida como tiempo hasta el primer token (TTFT) y velocidad de generación en tokens por segundo. Estas métricas permitirán identificar qué modelos son más aptos para las distintas fases del sistema RAG: búsqueda, redacción y entrega final. [1] [2]

Los resultados de esta evaluación no solo guiarán el diseño técnico del asistente, sino que también ofrecerán pautas prácticas para su uso en entornos educativos y profesionales, con el objetivo de transformar la manera en que producimos y compartimos conocimiento.

2. Metodología

Se plantea un diseño metodológico de carácter experimental comparativo, basado en un esquema de benchmarking controlado, la finalidad central es examinar y contrastar el rendimiento de distintos modelos de lenguaje ejecutados en entorno local e integrados en un pipeline de Recuperación Aumentada por Generación (RAG). Para garantizar objetividad, el análisis se apoya en tareas previamente definidas y en métricas cuantificables. [17] El protocolo establece ocho actividades representativas de la asistencia académica, todas formuladas en español y aplicadas de manera idéntica a cada modelo. Dichas tareas incluyen: consultas factuales breves, definición de conceptos, ejercicios de razonamiento lógico y aritmético, interpretación de fragmentos de código, generación breve de texto y otras pruebas de utilidad práctica. De este modo, se busca asegurar comparabilidad entre resultados y cubrir un espectro variado de escenarios comunes en contextos de apoyo académico. La evaluación contempla tres métricas principales: precisión, entendida como el porcentaje de aciertos por tarea; latencia, medida como el tiempo hasta la emisión del primer token (Time To First Token, TTFT); y tasa de generación, expresada en tokens producidos por segundo. [9] Estos indicadores permiten orientar la selección del modelo más adecuado en cada fase del sistema ya sea búsqueda, redacción o entrega de resultados. Todas las pruebas se llevaron a cabo de manera local mediante Ollama, con configuraciones homogéneas para cada modelo, por ello este procedimiento asegura control experimental, reproducibilidad de los resultados y condiciones justas para la comparación.

2.1. conjunto de tareas de evaluación

Para analizar el desempeño de los modelos en diversos aspectos relevantes a la asistencia académica, se diseñó un conjunto de ocho tareas de evaluación que abarcan las siguientes categorías:

Consulta factual breve: preguntas puntuales de conocimiento general o específico, similares a lo que un usuario académico podría buscar. Ejemplos: "¿Cuál es la capital de México?" (geografía básica) o "¿En qué año fue la Independencia de México?" (historia). Estas tareas evalúan la capacidad del modelo para proporcionar datos precisos almacenados en sus parámetros (o que requerirían recuperación).

Definición de concepto: una pregunta abierta que solicita la explicación de un término técnico.
Por ejemplo: "¿Qué es la generación aumentada por recuperación (RAG)?". Esto pone a prueba el conocimiento de dominio del modelo (comprensión de un concepto de PLN) y su habilidad para expresarlo con claridad.

Resolución lógica y matemática sencilla: problemas breves que requieren

ISSN 1794-211 X • e-lSSN 2322-939X • Vol 20, N° 2 (julio-diciembre 2023). pp. 91-98. Universidad Distrital Francisco José de Caldas-Facultad Tecnológica.

razonamiento o cálculo. Se incluyó una tarea de lógica ("razonamiento" y "lógica", con preguntas de deducción simples) y una de aritmética básica ("suma_simple", sumas de números pequeños). Un ejemplo concreto es calcular 2+2 u otras sumas simples para verificar si el modelo puede realizar operaciones aritméticas elementales sin herramientas adicionales.

Interpretación de código: una tarea tipo "python_out" donde se pide el resultado de ejecutar un fragmento corto de código (en este caso, Python). [5] Esto simula situaciones en las que el asistente debe comprender seudocódigo, fórmulas o salidas de algoritmos presentes en trabajos académicos.

Generación de texto breve: una tarea creativa denominada "escritura", que consiste en pedir al modelo que redacte un pequeño fragmento de texto académico (por ejemplo, un resumen de un tema). Aquí se evalúa la coherencia, el estilo y la pertinencia de la respuesta generada, aunque la corrección factual no esté directamente en juego.

Cada modelo procesó exactamente las mismas ocho entradas (prompts) correspondientes a las tareas mencionadas. Las preguntas se formularon en español, dado que el sistema final se orienta a usuarios hispanohablantes; no obstante, varios modelos evaluados son multilingües u originalmente entrenados en inglés, lo cual agrega un desafío adicional de adaptación al español en las respuestas.

2.2 Modelos evaluados en entorno Ollama

Se seleccionaron seis modelos de lenguaje de distintos orígenes y tamaños, procurando cubrir tanto LLMs de propósito general populares como modelos especializados desarrollados en este proyecto. [4] Todos los modelos se ejecutaron localmente utilizando Ollama, un entorno que facilita la carga de LLMs en hardware de escritorio. A continuación, se describen brevemente los modelos comparados:

Qwen (versión reciente "latest"): Modelo abierto de última generación proporcionado por Alibaba, con aproximadamente 7 mil millones de parámetros (7B). [6] Es conocido por su alto rendimiento general en múltiples tareas y un entrenamiento bilingüe (inglés-chino) que le confiere amplio conocimiento factual. Se espera que este modelo represente un "tope" de calidad dentro de los modelos de tamaño medio evaluados.

Qwen-1.7B: Una variante reducida de Qwen, con ~1.7 mil millones de parámetros. Este modelo más pequeño sacrifica capacidad de conocimiento y comprensión a cambio de mayor velocidad. [6] Resulta útil para evaluar qué tanto cae el desempeño al reducir drásticamente el tamaño en una misma familia de modelos.

LLaMA (formato instruccional, "instruct"): Un modelo derivado de LLaMA 2 de Meta AI, ajustado para seguimientos de instrucciones. [7] El tamaño de este modelo es del orden de 7-13B parámetros (la versión exacta empleada corresponde a "Llama-3.1-8B-Instruct" según la nomenclatura interna). Los modelos LLaMA han demostrado alta calidad en generación de texto, por lo que su versión fine-tuned para instrucciones debería manejar bien respuestas enfocadas y educadas.

LLaMA (versión base "latest"): La variante base sin afinamiento conversacional del modelo LLaMA (similar rango de parámetros que el anterior). Este modelo sirve para observar la diferencia que aporta el fine-tuning de instrucciones en la precisión y estilo de las respuestas. [7] En general, los modelos base pueden ser más propensos a respuestas genéricas o faltas de formato, pero a veces retienen mejor ciertos conocimientos técnicos.

DeepSeek-R1:8B: Modelo desarrollado en el marco de este proyecto, con ~8 mil millones de parámetros. Se trata de una versión destilada de un modelo de razonamiento propietario (DeepSeek-R1 original), construida sobre la arquitectura LLaMA 3.1 (8B) instructiva. Su entrenamiento incorpora técnicas avanzadas de destilación y aprendizaje por refuerzo con el objetivo de ofrecer alta capacidad de razonamiento lógico, manejo de código y matemáticas en un formato compacto. Este modelo fue liberado bajo licencia MIT en enero de 2025 y está optimizado para entornos locales, soportando ventanas de contexto extremadamente amplias (hasta 32k–128k tokens). [8] [15] Se espera que DeepSeek-8B alcance un equilibrio entre la precisión de un modelo mediano y la eficiencia computacional, priorizando su uso como motor generativo central en el backend del sistema.

DeepSeek-R1:1.5B: Versión ultracompacta del modelo anterior, con ~1.5 mil millones de parámetros. Representa un experimento de miniaturización extrema: sacrificar profundidad en la comprensión a cambio de tiempos de respuesta casi instantáneos. [8] Puede resultar útil en etapas muy tempranas del pipeline (por ejemplo, para procesamiento superficial de consultas o filtrado rápido de resultados), siempre y cuando conserve un rendimiento aceptable en tareas sencillas. Es importante destacar que todos los modelos son open-source o de acceso público, en línea con el objetivo del proyecto de utilizar fuentes económicas y abiertas para favorecer la equidad tecnológica. [3] La elección de Ollama como entorno de ejecución responde a la necesidad de implementar los modelos localmente de forma eficiente; esta herramienta permite cargar modelos optimizados (por ejemplo, en formato cuantizado de 4-bit) aprovechando la aceleración por GPU o incluso CPU de equipos personales. [4] Las evaluaciones se realizaron en un equipo con hardware de gama media, simulando el escenario de uso accesible para un investigador individual, en la figura 1 se pueden ver los componentes del hardware de prueba, cada modelo se corrió secuencialmente en condiciones controladas, reiniciando la instancia de Ollama entre pruebas para evitar interferencias o cache de respuestas previas.

ISSN 1794-211 X • e-lSSN 2322-939X • Vol 20, N° 2 (julio-diciembre 2023). pp. 91-98. Universidad Distrital Francisco José de Caldas-Facultad Tecnológica.

Sistema operativo:	Windows 11 Home 64 bits (10.0, compilación 26100)
Idioma:	español (configuración regional: español)
Fabricante del sistema:	LENOVO
Modelo del sistema:	81X2
BIOS:	EECN41WW
Procesador:	AMD Ryzen 5 4500U with Radeon Graphics (6 CPUs), ~2.4GHz
Memoria:	16384MB RAM
Archivo de paginación:	11493MB usados, 8216MB disponibles
Versión de DirectX:	DirectX 12

Figura 1. Hardware del equipo

Fuente: elaboración propia.

2.3 Métricas de desempeño

Para cada combinación de modelo y tarea se registraron las siguientes métricas:

Precisión (Score): se definió como una variable binaria por tarea, asignando 1 si la respuesta del modelo fue correcta (o cumplió adecuadamente lo solicitado) y 0 en caso contrario. [16] Posteriormente, se calculó la precisión promedio por modelo sobre el total de 8 tareas, expresada en porcentaje de aciertos (%). [10] Este indicador refleja la exactitud global del modelo en las distintas pruebas. La corrección se determinó comparando las respuestas con un ground truth conocido (en tareas factuales, se verificó con fuentes confiables la respuesta correcta; en lógica o código, se comprobó el resultado esperado; en redacción, se evaluó si el texto era coherente y pertinente al tema solicitado). Dado el carácter conciso de las tareas, esta métrica es equivalente a una medida de exactitud o accuracy clásica.

Tiempo hasta el primer token (TTFT): medida de latencia en segundos, contabilizada desde que se envía la pregunta al modelo hasta que éste produce el primer carácter de la respuesta. Es un indicador crítico para la responsividad percibida por el usuario, especialmente en consultas donde se espera una reacción rápida (por ejemplo, al realizar búsquedas iterativas). [9] Un TTFT bajo significa que el modelo comienza a responder casi instantáneamente, lo cual es deseable para mantener una interacción fluida. Factores que influyen en el TTFT incluyen la carga inicial del modelo, la complejidad del prompt y las optimizaciones internas de decodificación.

Tasa de generación (tokens/segundo): velocidad de generación de texto tras el primer token, calculada como el promedio de tokens producidos por segundo durante la respuesta. Esta métrica complementa al TTFT para entender el rendimiento: dos modelos pueden tener arranques lentos similares, pero uno de ellos quizá genere texto mucho más rápido una vez iniciado. La tasa se obtuvo dividiendo el número total de tokens generados en la respuesta por el tiempo de generación (excluyendo el TTFT). [9] Un valor más alto indica que el modelo puede elaborar respuestas largas en menos tiempo, lo cual es útil cuando se requieren explicaciones extensas o desarrollo de ideas. No obstante, tasas muy altas podrían corresponder a modelos que "producen" texto rápidamente a costa de la calidad, por lo que siempre se interpretan junto con la precisión lograda. Todas las métricas se calcularon de forma automatizada mediante scripts conectados a la API de Ollama, y se validaron manualmente en casos atípicos para asegurar su congruencia. Para facilitar la comparación, en la siguiente sección se presentan los resultados agregados de cada modelo en las tres métricas mencionadas. Asimismo, se analizan ejemplos cualitativos de las respuestas para contextualizar las diferencias de desempeño numérico.

3. Resultados y discusión

Como se observa en la Figura 2, qwen3: latest alcanza la puntuación promedio más alta (0.875), seguido por llama3: instruct y llama3: latest (ambos alrededores de 0.750). En un segundo bloque de desempeño aparecen qwen3:1.7b y deepseek-r1:8b (≈ 0.625). Finalmente, deepseek-r1:1.5b se ubica claramente rezagado con un valor cercano a 0.25. Comparativa de precisión promedio (porcentaje de tareas respondidas correctamente) y velocidad de generación (tokens generados por segundo) para los modelos evaluados.

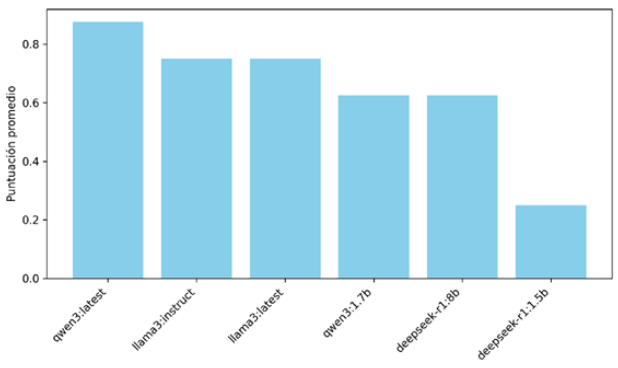

Figura 2. Comparación de calidad promedio por modelo

Fuente: elaboración propia.

Se aprecia la compensación rendimiento-precisión: los modelos más pequeños obtienen menores porcentajes de acierto (barra azul) pero generan texto con mayor rapidez (barra verde).

ISSN 1794-211 X • e-lSSN 2322-939X • Vol 20, N° 2 (julio-diciembre 2023). pp. 91-98. Universidad Distrital Francisco José de Caldas-Facultad Tecnológica.

3.1 Precisión de los modelos en tareas académicas

En términos de precisión, los modelos evaluados exhibieron diferencias sustanciales. El mejor desempeño correspondió al modelo Qwen "latest", con un 87.5% de precisión, es decir, respondió correctamente 7 de las 8 tareas. Este modelo falló únicamente en la tarea de aritmética básica (suma_simple), donde sorprendentemente no entregó la respuesta correcta a una suma sencilla. En todas las demás pruebas (consultas factuales, definición, lógica, código y redacción), Qwen demostró un alto nivel de acierto, lo que refleja su amplio conocimiento estático y buena capacidad de razonamiento. Por ejemplo, identificó correctamente "Ciudad de México" como la capital del país, proporcionó el año exacto de la independencia (1810), definió RAG de forma coherente y hasta dio una respuesta válida a la salida del código Python propuesto. Este rendimiento destaca considerando que trabajó sin acceso a fuentes externas; su base de entrenamiento parece suficientemente rica en información general y técnica.

Los modelos LLaMA evaluados (tanto la variante instruct afinada, como la versión base) alcanzaron un 75% de precisión cada uno (6 de 8 respuestas correctas). Ambos coincidieron en los aciertos y errores, lo cual sugiere que la afinación instruccional mejoró principalmente el estilo o formato de respuesta más que la cantidad de conocimiento aplicado. En concreto, los LLaMA respondieron correctamente las preguntas factuales simples (capital y año histórico), resolvieron bien la pregunta lógica y la suma aritmética, y cumplieron satisfactoriamente la tarea de redacción. Sin embargo, fallaron en la definición de RAG (sus explicaciones fueron incompletas o imprecisas, posiblemente por lagunas en ese término específico) y también en la tarea de interpretar el código Python (dieron una salida incorrecta del fragmento de código). [5] Estos errores sugieren que, a pesar de su sólido desempeño general, los modelos basados en LLaMA tuvieron dificultades con cierto conocimiento especializado (RAG es un concepto relativamente nuevo en PLN) y con tareas que implican ejecución simulada de código. Es posible que su entrenamiento no reforzara tanto esas habilidades, o que adoptaran un estilo de respuesta más genérico en lugar de concretar la solución correcta en esos casos.

El modelo DeepSeek-R1:8B logró una precisión del 62.5% (5 de 8 tareas correctas). Sus aciertos y fallos evidencian un perfil distinto al de los modelos generales: DeepSeek-8B respondió correctamente a la definición de RAG (donde las LLaMA flaquearon), lo que era esperable dado que este modelo fue diseñado con énfasis en razonamiento y conocimientos técnicos. También acertó en la pregunta lógica, la consulta de capital geográfica, y cumplió con la tarea de redacción. No obstante, falló en la pregunta factual del año de independencia (su respuesta no coincidió con la fecha histórica exacta) y en la suma aritmética simple, así como en la interpretación del código Python. Esto indica que, si bien DeepSeek-8B maneja conceptos complejos y mantiene coherencia en la generación, puede carecer de ciertos datos triviales o sufrir lapsos en cálculos muy básicos. [12] Dado que su entrenamiento estuvo orientado a cadena de pensamiento (CoT) y resolución de problemas lógicos, podría ser que no "memoriza" tanto datos concretos como un modelo general, esperando recibir esa información vía contexto externo. [14] En un entorno RAG real, esta debilidad no sería grave puesto que la información factual vendría de la base de datos; de hecho, es consistente con la filosofía RAG: el modelo se especializa en integrar y razonar con datos, más que en almacenarlos. En síntesis, DeepSeek-8B mostró fortalezas en razonamiento y conocimientos especializados (acertando donde otros erraron, como la definición técnica), a costa de un rendimiento apenas regular en preguntas de cultura general o cálculo sencillo.

El modelo Qwen-1.7B obtuvo igualmente 62.5% de precisión (5/8). Interesantemente, sus aciertos complementaron de cierta forma a los de DeepSeek-8B: Qwen-1.7B respondió bien el año de la independencia (poseía ese dato) y las tareas de lógica y definición RAG, pero falló en la capital de México y en la suma simple, además del código Python. Esto sugiere que el submodelo Qwen reducido retuvo algo de conocimiento histórico técnico (acertó donde su contraparte grande también lo hizo, salvo la capital), pero su entendimiento del español pudo haber influido en confusiones para "México capital". El bajo resultado en la suma confirma que el manejo de números de este modelo miniaturizado es limitado. En general, con 1.7B de parámetros, es notable que superara en precisión a un modelo más grande en ciertas preguntas, lo que apunta a posibles diferencias de entrenamiento o énfasis: tal vez Qwen-1.7B fue afinado en español o en benchmarking de QA históricas, dándole ventaja en la pregunta del año, mientras que la capital (que uno pensaría igual de simple) pudo haberse formulado de modo que el modelo se confundiera.

Finalmente, el modelo más pequeño, DeepSeek-R1:1.5B, alcanzó apenas 25% de precisión (2 de 8 tareas). Solo logró responder correctamente la pregunta lógica y la tarea de redacción creativa. En todas las demás, su desempeño fue deficiente: erró ambas preguntas factuales (pareció no conocer los datos solicitados), no supo definir RAG (posiblemente nunca incorporó ese concepto escasamente difundido), falló en la suma simple (pudo haber dado un resultado equivocado o incoherente) y

ISSN 1794-211 X • e-lSSN 2322-939X • Vol 20, N° 2 (julio-diciembre 2023). pp. 91-98. Universidad Distrital Francisco José de Caldas-Facultad Tecnológica.

tampoco logró interpretar el código Python. Estos resultados confirman las limitaciones esperadas de un modelo tan compacto: su base de conocimiento es muy reducida, por lo que responde al azar o con generalidades en preguntas de hecho, y su capacidad de cálculo interno también es prácticamente nula. Sin embargo, es interesante notar que acertó la pregunta de lógica, lo cual sugiere que quizás fue capaz de seguir correctamente un silogismo sencillo, y cumplió la tarea de escribir un texto. Esto último indica que incluso un modelo pequeño puede producir texto gramaticalmente aceptable, pero la confiabilidad de su contenido será baja.

La Figura 3 muestra claramente tres ligas de velocidad. En la punta del podio están deepseekr1:1.5b y qwen3:1.7b, sacando ventaja con ritmos de generación que rondan los 21.9 y 18.1 tokens por segundo, son rápidos, casi como si estuvieran con prisa; un poco más atrás, en terreno intermedio, aparece llama3:instruct, que marca unos 7.4 tokens/s. No es un rayo, pero tampoco se queda rezagado. Y luego están qwen3:latest y llama3:latest, que se mueven a un paso estable, entre 5.3 y 5.5 tokens/s. El más pausado del grupo es deepseek-r1:8b, con 4.9 tokens/s, que parece tomarse su tiempo para elegir cada palabra. En realidad, no es casualidad, porque los modelos más pequeños (1.5B–1.7B) tienden a escribir más rápido, algo muy útil para pruebas rápidas o interfaces donde cada segunda cuenta. En cambio, los modelos medianos (7B–8B) sacrifican algo de velocidad, pero ganan en detalle y consistencia. Es como elegir entre un coche deportivo y un sedán cómodo: uno te lleva más rápido, el otro te da un viaje más suave, al final lo práctico es combinarlos: usar los veloces para explorar ideas y los más tranquilos para pulir el texto final cuando la precisión es lo que manda.

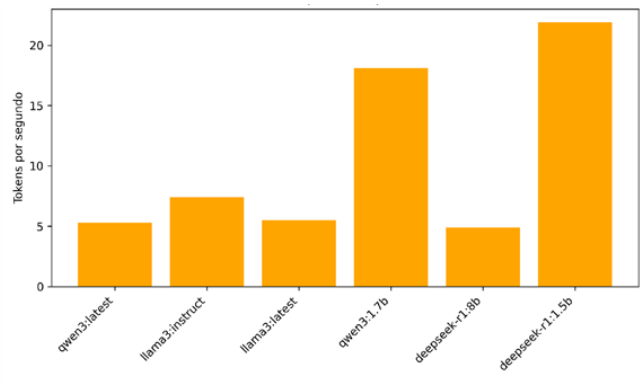

Figura 3. Velocidad por modelo

Fuente: elaboración propia.

En resumen, la jerarquía de precisión observada fue: Qwen-large (87.5%) > LLaMA (75%) > DeepSeek-8B ≈ Qwen-small (62.5%) > DeepSeek-1.5B (25%). Esta gradación sugiere que los modelos más grandes y recientes (Qwen, LLaMA) aún llevan la delantera en conocimiento general y versatilidad, mientras que los modelos especializados o distilados pueden rivalizar en áreas puntuales (DeepSeek-8B en definiciones técnicas) pero no cubren todo el espectro. Los modelos diminutos, si bien muy rápidos, no son confiables para brindar información precisa de forma autónoma. En un sistema RAG, esto refuerza la idea de que modelos pequeños solo deberían usarse con apoyo fuerte de un módulo recuperador (por ejemplo, alimentándoles directamente el dato factual para que lo inserten en la respuesta), o bien limitarlos a funciones donde un error no sea crítico.

model	avg_score	avg_tok_s	avg_ttft_s
qwen3:latest	0.875	5.3	3.818
llama3:instruct	0.75	7.4	3.73
llama3:latest	0.75	5.5	4.039
qwen3:1.7b	0.625	18.1	1.379
deepseek-r1:8b	0.625	4.9	3.507
deepseek-r1:1.5b	0.25	21.9	0.966

Tabla 1. Resultados por modelos

Fuente: elaboración propia.

3.2 Latencia y velocidad de generación

En aplicaciones interactivas, la velocidad es crucial. Los resultados de latencia (TTFT) y rendimiento muestran diferencias notables entre los modelos, alineadas en gran medida con el tamaño de cada uno. Los modelos más pequeños iniciaron y completaron sus respuestas mucho más rápido que los grandes.

El modelo DeepSeek-R1:1.5B fue el más ágil con diferencia: tuvo un TTFT promedio de solo ~0.97 segundos, prácticamente inmediato, y una tasa de generación de ~21.9 tokens/segundo, la más alta del conjunto. En la práctica, esto significa que este modelo comienza a responder casi al instante de recibir la pregunta y puede producir alrededor de 22 palabras por segundo (suponiendo ~1 token ≈ palabra corta). Una respuesta de longitud moderada (por ejemplo, 100 tokens ≈ 2-3 párrafos cortos) podría estar lista en unos 5 segundos, lo cual es excelente desde la perspectiva de experiencia de usuario. Esta celeridad confirma la expectativa de que un modelo de 1.5B, al tener muchos menos cálculos internos, ofrece respuestas casi en tiempo real. Como referencia, su desempeño es comparable al de un asistente tradicional basado en reglas, pero con la diferencia de que sí genera lenguaje natural (aunque

ISSN 1794-211 X • e-ISSN 2322-939X • Vol 20, N° 2 (julio-diciembre 2023). pp. 91-98. Universidad Distrital Francisco José de Caldas-Facultad Tecnológica.

su precisión, como vimos, es baja). De forma similar, el modelo Qwen-1.7B mostró latencia reducida (TTFT ~1.38 s) y una notable rapidez de generación (~18.1 tokens/s), siendo el segundo más veloz. Ambos modelos pequeños serían ideales para escenarios donde se requieren múltiples iteraciones rápidas, como refinar una búsqueda con varias consultas cortas, ya que introducen muy poca demora en cada ciclo.

Por otro lado, los modelos de tamaño mediano presentaron latencias intermedias. DeepSeek-R1:8B tuvo un TTFT en torno a 3.5 s y generó a ~4.9 tokens/s, mientras que los modelos LLaMA (7-13B) registraron TTFT ~3.7–4.0 s y velocidades de ~5.3–7.4 tokens/s. En particular, la variante LLaMA-Instruct fue ligeramente más rápida (7.4 tok/s) que la LLaMA base (5.5 tok/s), lo cual podría atribuirse a optimizaciones en su implementación o a un estilo de respuesta más directo. La latencia de ~3-4 segundos, aunque perceptible, sigue siendo aceptable en un contexto interactivo; corresponde a una pequeña "pausa" antes de que el usuario vea la respuesta comenzar a aparecer. Para un sistema que asiste en redacción, este retraso inicial es tolerable, en especial si lo compensan con mayor calidad en la salida. La tasa de generación de ~5-7 tokens/s implica que un párrafo de 100 palabras tomaría del orden de 15-20 segundos en generarse completamente.

El modelo Qwen "latest" pese a su alta precisión presentó una velocidad en la misma línea que LLaMA: TTFT ~3.8 s y ~5.3 tokens/s. Es decir, sacrifica rapidez a cambio de mayor profundidad. De hecho, comparando Qwen-7B con Qwen-1.7B, vemos que el grande es casi tres veces más lento en tokens/segundo. Esto evidencia la clásica disyuntiva en LLMs: modelos más complejos requieren más tiempo de inferencia, proporcional al número de parámetros y la carga computacional de sus atenciones. Sin embargo, vale resaltar que en todos los casos evaluados la latencia inicial se mantuvo debajo de 5 segundos, lo cual es fruto de optimizaciones en el entorno Ollama y posiblemente el uso de cuantización que reduce el tamaño efectivo del modelo. Cinco segundos es un umbral psicológico importante; por encima de eso, la interacción podría volverse tediosa.

El mensaje de la figura 4 es directo: a menor TTFT, mayor sensación de fluidez. [16] Así, los modelos compactos (1.5B–1.7B) "encienden" antes, aunque como se mostró en la figura de precisión siempre sostienen la misma calidad en la respuesta final. [9] En cambio, los modelos medianos (7B–8B) tardan más en arrancar, pero suelen compensar con mayor exactitud y estabilidad en tareas de redacción.

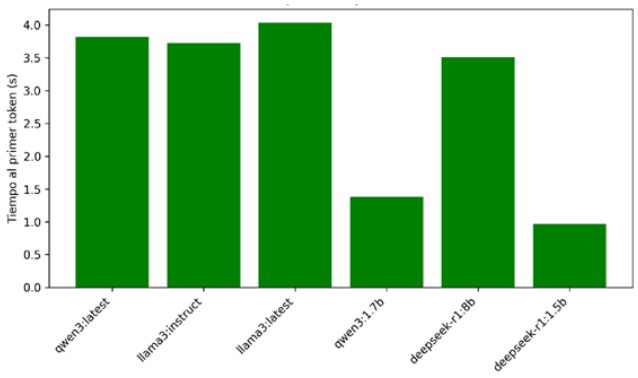

Figura 4. TTFT promedio por modelo

Fuente: elaboración propia.

En síntesis, desde la perspectiva de latencia: DeepSeek-1.5B > Qwen-1.7B (ambos prácticamente instantáneos) > DeepSeek-8B ≈ LLaMA-Instruct > Qwen-7B ≈ LLaMA-base (los más lentos, pero aún bajo 4 s TTFT). Y en tasa de generación: DeepSeek-1.5B > Qwen-1.7B >> LLaMAInstr > LLaMA-base ≈ Qwen-7B > DeepSeek-8B. Cabe preguntarse por qué DeepSeek-8B fue ligeramente más lento en tokens/s que modelos de tamaño similar; posiblemente su arquitectura con ventana extendida y mecanismos de razonamiento añadidos conlleva un costo computacional extra por token. Este punto merece atención, ya que en un entorno RAG DeepSeek-8B podría manejar documentos largos gracias a sus 32k tokens de contexto, pero procesarlos será más lento, aun así, sus tiempos siguen siendo prácticos.

Desde luego, la velocidad no lo es todo: estos resultados deben equilibrarse con la precisión discutida antes. Por ejemplo, DeepSeek-1.5B es rapidísimo, pero respondía mal en la mayoría de las preguntas; no conviene entregarle tareas críticas solo por velocidad. Un hallazgo interesante es que Qwen-1.7B combinó buena velocidad con precisión moderada (5/6), lo que lo posiciona como un candidato atractivo para ciertas funciones: es casi tan rápido como el más pequeño, pero sustancialmente más preciso. Esta observación sugiere que, si se tuviera que escoger un único modelo de compromiso para todo el sistema, un modelo en el rango ~2B parámetros podría ofrecer el mejor equilibrio entre rapidez y aceptabilidad de respuestas.

ISSN 1794-211 X • e-lSSN 2322-939X • Vol 20, N° 2 (julio-diciembre 2023). pp. 91-98. Universidad Distrital Francisco José de Caldas-Facultad Tecnológica.

4. Conclusiones

Este estudio presenta una evaluación exhaustiva de diversos modelos de lenguaje ejecutados en entornos locales, con el objetivo de explorar su aplicabilidad dentro de un sistema de Recuperación Aumentada para Generación (RAG) orientado al ámbito académico, los hallazgos obtenidos evidencian que no existe un modelo único capaz de dominar en todas las dimensiones de desempeño; por el contrario, cada uno manifiesta fortalezas y limitaciones que pueden aprovecharse de manera diferenciada en distintas fases del proceso; en este contexto, el modelo Qwen-7B destacó por su precisión y versatilidad, consolidándose como la opción más adecuada para la producción de contenidos complejos y coherentes, los modelos LLaMA exhibieron también una calidad notable, aunque con un alcance ligeramente inferior en el conocimiento específico, resultando, sin embargo, altamente útiles para tareas de redacción general. Por su parte, el modelo especializado DeepSeek-R1:8B demostró el valor de la personalización: pese a contar con un tamaño moderado, alcanzó un rendimiento competitivo en razonamiento y manejo de conceptos técnicos, lo que sugiere su potencial en etapas donde se privilegia el pensamiento estructurado sobre la simple recuperación factual.

Asimismo, los modelos de menor escala, como Qwen-1.7B y DeepSeek-1.5B, resaltaron por su eficiencia computacional. Su baja latencia y elevada velocidad de generación los posicionan como herramientas estratégicas en escenarios donde la inmediatez es prioritaria, aunque sacrificando profundidad semántica. Dentro de un ecosistema RAG, lejos de ser descartados, estos modelos ligeros pueden cumplir un rol de apoyo complementario, actuando como filtros iniciales o asistentes que optimizan la interacción con modelos de mayor envergadura.

A partir de estas observaciones, se plantea la conveniencia de adoptar una arquitectura híbrida, asignando a cada etapa del sistema el modelo con mejor relación costo-beneficio según la naturaleza de la tarea, de este modo, los modelos livianos pueden potenciar las búsquedas preliminares, los de escala intermedia o grande encargarse de la generación textual fundamentada, y los pequeños o deterministas perfeccionar los ajustes finales, esta orquestación no solo maximiza el rendimiento global del sistema, sino que también garantiza estándares elevados de precisión, coherencia y adecuación estilística en los documentos académicos producidos, los resultados obtenidos confirman la hipótesis central de este trabajo: la integración de un enfoque RAG inteligente permite optimizar la producción de conocimiento académico mediante la sinergia entre recuperación y generación. Incluso los modelos locales de menor tamaño se benefician de la inyección de información actualizada, minimizando errores significativos y potenciando su utilidad práctica. Con ello, se vislumbra un escenario en el que la precisión alcanzada por los mejores modelos (cercana al 90% en ausencia de RAG) podría aproximarse a niveles casi perfectos al habilitar esta arquitectura, reduciendo sustancialmente el fenómeno de las "alucinaciones". Finalmente, como línea futura, se proyecta la implementación integral del sistema RAG propuesto y la evaluación en contextos reales con usuarios finales. Ello incluirá no solo pruebas de desempeño técnico, sino también análisis de usabilidad y calidad percibida de los documentos generados. De manera paralela, se contempla la incorporación de modelos emergentes de código abierto, cuya acelerada evolución sugiere la pronta aparición de alternativas intermedias con un balance aún más favorable entre precisión y eficiencia. Esta flexibilidad modular asegura la posibilidad de integrar mejoras de manera continua, sentando las bases para sistemas de redacción académica de nueva generación, sustentados en inteligencia artificial local y adaptados a entornos dinámicos y exigentes.

5. Referencias

[1] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, et al., "Improvinglanguage models by retrieving from trillions of tokens," arXiv preprint, arXiv:2112.04426, 2022. [En línea]. Disponible en: https://arxiv.org/abs/2112.04426

[2] P. Lewis, E. Pérez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, et al., "Retrievalaugmented generation for knowledge-intensive NLP tasks," Advances in Neural Information Processing Systems, vol. 33, pp. 9459–9474, 2020. [En línea]. Disponible en: https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5Abstract.html

[3] K. Guu, K. Lee, Z. Tung, P. Pasupat, y M. Chang, "Retrieval augmented language model pretraining," en Proc. of ICML, PMLR, vol. 119, 2020, pp. 3929–3938. [En línea]. Disponible en: https://arxiv.org/abs/2002.08909

[4] F. Liu, Z. Kang, y X. Han, "Optimizing RAG techniques for automotive industry PDFchatbots: A case study with locally deployed Ollama models," en Proc. of the 2024 3rd International Conference on Artificial

Intelligence and Intelligent Information Processing (AIIIP '24), Nueva York, NY, EE.UU.: ACM, 2025, pp. 152–159. [En línea]. Disponible en: https://doi.org/10.1145/3707292.3707358

[5] E. Cassingena Navone, "Python ejemplos de código – tutorial de programación en Pythondesde cero para principiantes," 24 febrero 2022. [En línea]. Disponible en: https://www.freecodecamp.org/espanol/news/python-ejemplos-de-codigo-tutorial-deprogramacion-en-python-desde-cero-para-principiantes/

[6] H. Shah, "Evaluating Sub-3B Parameter Language Models on Math Word Problems," TechRxiv, 21 mayo 2025. [En línea]. Disponible en: https://www.techrxiv.org/users/922842/articles/1294782-evaluating-sub-3b-parameterlanguage-models-on-math-word-problems

[7] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre,T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. C. Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve, "Code Llama: Open Foundation Models for Code," arXiv preprint, arXiv:2308.12950, submitted 24 August 2023; revised 31 January 2024. [En línea]. Disponible en: https://arxiv.org/abs/2308.12950

[8] E. Evstafev, "Token-Hungry, Yet Precise: DeepSeek R1 Highlights the Need for Multi-StepReasoning Over Speed in MATH," arXiv preprint, arXiv:2501.18576, submitted 30 enero 2025. [En línea]. Disponible en: https://arxiv.org/abs/2501.18576

[9] J. Liu, B. Chen y C. Zhang, "Speculative Prefill: Turbocharging TTFT with Lightweight andTraining-Free Token Importance Estimation", mayo 2025. [En línea]. Disponible en: https://arxiv.org/abs/2502.02789

[10] C. Goutte y E. Gaussier, "A Probabilistic Interpretation of Precision, Recall and F-Score,with Implication for Evaluation", marzo 2005. [En línea]. Disponible en: https://link.springer.com/chapter/10.1007/978-3-540-31865-1_25

[11] J. Johnson, M. Douze, y H. Jégou, "Billion-scale similarity search with GPUs," IEEE Trans.Big Data, vol. 7, no. 3, pp. 535–547, 2019. DOI: 10.1109/TBDATA.2019.2921572

[12] F. Petroni, P. Lewis, A. Piktus, T. Rocktäschel, Y. Wu, A. H. M. Khattab, et al., "KILT: Abenchmark for knowledge intensive language tasks," en Proc. of NAACL, 2021, pp. 2523–2544. arXiv:2009.02252.

[13] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, et al., "Dense passage retrievalfor open-domain question answering," en Proc. of EMNLP, 2020, pp. 6769–6781. DOI: 10.18653/v1/2020.emnlp-main.550.

[14] M. Izacard y E. Grave, "Leveraging passage retrieval with generative models for opendomain question answering," en Proc. of EACL, 2021, pp. 874–880. [En línea]. Disponible en: https://aclanthology.org/2021.eacl-main.74

[15] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, et al., "Exploring the limitsof transfer learning with a unified text-to-text transformer," J. Mach. Learn. Res., vol. 21, no. 140, pp. 1–67, 2020. [En línea]. Disponible en: https://jmlr.org/papers/v21/20074.html

[16] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, et al., "Language modelsare few-shot learners," arXiv preprint, arXiv:2005.14165, 2020. [En línea]. Disponible en: https://arxiv.org/abs/2005.14165

[17] "Benchmarking de Modelos de Lenguaje Discriminativos ...", Trabajo Final de Máster,Máster Universitario en Ciencia de Datos (Data Science), Universitat Oberta de Catalunya (UOC), 2025. [En línea]. Disponible en: https://openaccess.uoc.edu/server/api/core/bitstreams/e85d728f-75d9-4fe7-a67c5173d8d59610/content