Comparación del desempeño de LLMs en la generación de diagramas de clases UML mediante un sistema RAG

Performance comparison of LLMs in the Generation of UML class diagrams using a RAG system

Lorena Martínez Sixto
Universidad Tecnológica de la Mixteca, Oaxaca, México
martinezsixto.10@gs.utm.mx

Carlos Alberto Fernández y Fernández
Universidad Tecnológica de la Mixteca, Oaxaca, México
caff@gs.utm.mx
ORCID: 0000-0002-1586-8772

Christian Eduardo Millán Hernández
Universidad Tecnológica de la Mixteca, Oaxaca, México
cmillanh@gs.utm.mx
ORCID: 0000-0002-8683-7500

https://doi.org/10.36825/RITI.13.31.007

Recibido: Junio 22, 2025
Aceptado: Septiembre 30, 2025

Resumen: Este estudio presenta una evaluación comparativa del desempeño de seis modelos LLM (Gemini-2.5-Pro, Claude-Sonnet-4, GPT-4, DeepSeek-R1, Llama3.2 y Mistral) en un sistema RAG para la generación de diagramas de clases UML a nivel de análisis, a partir de historias de usuario redactadas en lenguaje natural. Para la generación y evaluación de los diagramas se utilizó código PlantUML, lo que permitió comparar los diagramas generados con los diagramas de referencia mediante la métrica ROUGE-L, que se centra en el *recall* promedio. Los resultados demostraron que los modelos Gemini-2.5-Pro, Claude-Sonnet-4 y GPT-4 obtuvieron un mejor desempeño, destacando Claude-Sonnet-4 por alcanzar los puntajes promedio más altos en la mayoría de las historias de usuario. En contraste, los modelos DeepSeek-R1, Llama3.2 y Mistral presentaron dificultades, incluyendo la generación de código inválido en PlantUML, lo cual limitó la evaluación automática en ciertos casos. La incorporación del sistema RAG brindó una base inicial para explorar mejoras en la calidad de las respuestas, lo que sugiere limitaciones en la calidad y pertinencia del contexto recuperado. Finalmente, se identificaron oportunidades de mejora, como el afinamiento del *prompt* y la mejora del contexto utilizado por el sistema RAG.

Palabras clave: *Ingeniería de Software (SE), Ingeniería de Requisitos (RE), Modelos de Lenguaje Largo (LLMs), Sistemas RAG, Diagramas UML.*

Abstract: This study presents a comparative evaluation of the performance of six LLMs (Gemini-2.5-Pro, Claude-Sonnet-4, GPT-4, DeepSeek-R1, Llama3.2, and Mistral) within a RAG system for generating UML class diagrams at the analysis level, based on user stories written in natural language. PlantUML code was used for the generation and evaluation of the diagrams, enabling a comparison between the generated diagrams and reference diagrams using the ROUGE-L metric, which focuses on average recall. The results showed that Gemini-2.5-Pro, Claude-Sonnet-4, and GPT-4 achieved better performance, with Claude-Sonnet-4 standing out by obtaining the highest average scores in most user stories. In contrast, DeepSeek-R1, Llama3.2, and Mistral presented difficulties, including the generation of invalid PlantUML code, which limited automated evaluation in some cases. The incorporation of the RAG system provided an initial foundation for exploring improvements in response quality, suggesting limitations in the quality and relevance of the retrieved context. Finally, opportunities for improvement were identified, such as prompt refinement and enhancement of the context used by the RAG system.

Keywords: *Software Engineering (SE), Requirements Engineering (RE), Large Language Models (LLMs), RAG Systems, UML Diagrams.*

1. Introducción

Para Kotti *et al.* [1], el objetivo de la Ingeniería de Software (SE, por sus siglas en inglés) es desarrollar productos de alta calidad y fáciles de mantener. Hoffman *et al.* [2] explican que este proceso abarca fases como el análisis de requisitos, diseño, desarrollo, pruebas y mantenimiento, durante las cuales los ingenieros de software enfrentan tareas complejas que dependen de su experiencia, lo que incrementa costos y dificulta el desarrollo. En este sentido, Ferrario y Winter [3] explican que las crecientes expectativas de los usuarios, junto con la aparición constante de nuevas aplicaciones, imponen desafíos adicionales como mejorar la calidad del software, promover una mayor colaboración y reducir los tiempos de entrega.

Para hacer frente a estos desafíos, las metodologías ágiles han ganado popularidad al fomentar ciclos de desarrollo iterativos, entrega continua y una comunicación constante entre los miembros del equipo y los usuarios finales. En este contexto, las historias de usuario son utilizadas como una herramienta clave para capturar los requisitos del sistema desde la perspectiva del usuario final [4], [5]. Dado su papel central en la comunicación entre usuarios y desarrolladores, han surgido diversos estudios que buscan automatizar procesos a partir de estas descripciones funcionales. Estudios como el de Jin et al. [6], analizan cómo mejorar la automatización en la SE con el objetivo de hacer más eficiente el desarrollo y mantenimiento del software. En la actualidad, se destinan esfuerzos al desarrollo de herramientas que apoyen a los ingenieros de software en tareas clave, como la generación automática de código [7], la creación automática de casos de prueba [8], la detección de vulnerabilidades [9] y la generación de modelado UML de requisitos [10].

Los diagramas UML son fundamentales en la Ingeniería de Requisitos (RE, por sus siglas en inglés) para mejorar la comunicación y guiar el diseño del sistema de software. Sin embargo, complementar los requisitos redactados en lenguaje natural con modelado UML requiere un esfuerzo manual considerable. En entornos que adoptan metodologías ágiles, es fundamental facilitar a los equipos de desarrollo la automatización en tareas de modelado UML y diseño del software. Esto permite una integración más fluida entre la documentación generada a partir de historias de usuario y los artefactos de diseño, como los diagramas UML, lo que facilita el proceso y contribuye a reducir el tiempo requerido y la probabilidad de errores, por lo que Conrardy y Cabot [11] destacan que el uso de herramientas basadas en NLP para la generación de diagramas UML puede facilitar el trabajo de los ingenieros de requisitos y mejorar los procesos en RE.

En esta misma línea, la SE ha encontrado en los Modelos de Lenguaje Largo (LLMs, por sus siglas en inglés) una herramienta valiosa para abordar desafíos asociados al análisis de datos, código y texto. Zheng *et al.* [12] describen a los LLMs como modelos basados en redes neuronales entrenadas con una gran cantidad de datos y el número de parámetros puede llegar a superar los cientos de miles de millones. Hou *et al.* [13] destacan sus avances en tareas como el resumen y la generación de código, mientras que Jin *et al.* [6] resaltan su capacidad para automatizar tareas repetitivas, como crear fragmentos de código a partir de descripciones en lenguaje natural. Kalyan [14] destaca que los LLMs representan un avance significativo en el Procesamiento de Lenguaje Natural (NLP, por sus siglas en inglés), siendo capaces de interpretar el lenguaje natural, generar texto coherente, adaptarse al contexto e incorporar conocimientos especializados. Estas capacidades los hacen útiles en tareas como la traducción, la redacción, la recuperación de información y los sistemas de preguntas y respuestas, como señalan

Yao *et al.* [15]. Actualmente, existen numerosos LLMs ampliamente utilizados en el procesamiento de lenguaje natural. Entre los más relevantes se encuentran GPT [16], Claude [17], [18], Gemini [19], [20], LLaMA [21], [22], PaLM [22], DeepSeek [17], [23], Mistral [24] y Qwen [25], cada uno con características particulares que los hacen adecuados para distintos contextos. A continuación, se describen los LLMs más conocidos al momento de llevarse a cabo el presente estudio comparativo:

- **GPT:** Desarrollado por OpenAI, utiliza la arquitectura *Transformer* y un alto número de parámetros para generar respuestas detalladas y coherentes [16].
- **Gemini:** Desarrollado por Google, reconocido por su capacidad para realizar tareas complejas de NLP y su procesamiento multimodal [19], [20].
- **LLaMA:** Desarrollado por Meta, es un modelo que ha ganado reconocimiento por su eficiencia y flexibilidad. Su capacidad de adaptación mediante ajuste de instrucciones lo hace útil en diversos contextos [21], [22].
- **Claude:** Desarrollado por Anthropic AI, es un modelo enfocado en generar respuestas claras y fundamentadas, con el fin de reducir riesgos y priorizar la ética de sus interacciones, incluso con grandes volúmenes de información [17], [18].
- **DeepSeek:** Desarrollado por DeepSeek AI, con el objetivo de superar las limitaciones de los modelos ya conocidos, está diseñado para ser eficiente, adaptable y especializado en distintos dominios [17], [23].
- **Mistral:** Desarrollado por Mistral AI, es un modelo de última generación que ha ganado relevancia por sus capacidades avanzadas en tareas de NLP. Su arquitectura le permite comprender conceptos complejos, interpretar el contexto y generar texto fluido y preciso [24].

A pesar de su potencial, los LLMs presentan ciertas limitaciones, entre las que se destacan la restricción en la longitud del contexto, lo que restringe su capacidad para generar respuestas precisas cuando se enfrentan a entradas extensas. Además, su efectividad puede verse limitada en entornos dinámicos, lo que dificulta su adaptación a requisitos cambiantes, como mencionan Zheng *et al.* [12]. Asimismo, estos modelos dependen en gran medida de datos estáticos, lo que los hace susceptibles a generar información desactualizada o poco específica, especialmente cuando deben procesar grandes volúmenes de información, como señalan Zhao *et al.* [17] y Brown [26]. Estas restricciones pueden generar errores, conocidos como alucinaciones, que consisten en la generación de contenido falso o impreciso que, aunque puede parecer coherente, no se basa en datos reales ni en el contexto proporcionado, afectando así la precisión de las respuestas a causa de la falta de comprensión del contexto, la desconexión del tema y los sesgos en los datos de entrenamiento, tal como describen Martino *et al.* [27] y Rocco *et al.* [28]. Ram *et al.* [8] destacan que las alucinaciones representan un desafío significativo, e indican que la Generación Aumentada por Recuperación (RAG, por sus siglas en inglés) se presenta como una solución efectiva para mitigar sus efectos.

Li *et al.* [29] señalan que los sistemas RAG mejoran el rendimiento de los LLMs al proporcionar acceso a información externa y actualizada, generando respuestas más precisas. Chen *et al.* [30] resaltan su utilidad en escenarios con datos en constante cambio. Meng *et al.* [31] explican que RAG reduce la dependencia del conocimiento previo del modelo, mejorando la precisión y minimizando errores. De Bari *et al.* [32], Ferrari *et al.* [33] y Conrardy y Cabot [11] explican que los sistemas RAG constan de dos componentes: el recuperador, que extrae información relevante, y el generador, que la integra en la respuesta final, como se muestra en la Figura 1.

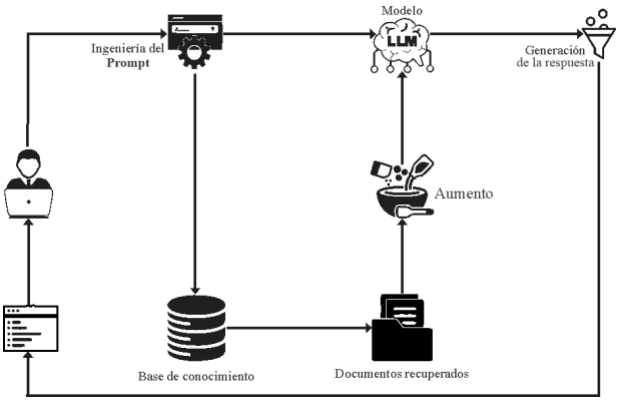


Figura 1. Diagrama de Generación Aumentada por Recuperación.

En este contexto, los sistemas RAG se presentan como una alternativa viable para automatizar y mejorar la generación de diagramas UML en etapas tempranas del desarrollo de software. Por lo tanto, el objetivo de este trabajo es evaluar cómo la incorporación de distintos LLMs dentro de un sistema RAG propuesto impacta en el desempeño de la generación automática de diagramas de clases a partir de historias de usuario.

No obstante, a pesar del potencial de los LLMs y de las ventajas de los sistemas RAG, existe poca evidencia sobre su efectividad concreta en tareas específicas de SE, como la generación automática de diagramas UML a partir de historias de usuario. Este trabajo parte del problema de que, en contextos ágiles, transformar requisitos expresados en lenguaje natural en modelos formales como diagramas de clases sigue siendo una tarea manual, costosa y propensa a errores. Por tanto, se busca evaluar en qué medida la incorporación de diferentes LLMs dentro de un sistema RAG puede mejorar dicha tarea, y cuáles son sus limitaciones prácticas al aplicarse a este dominio.

La estructura del presente trabajo es la siguiente: en la Sección 2 se presenta el estado del arte relacionado con el uso de LLMs para la generación de diagramas UML. La Sección 3 describe la metodología utilizada para la comparación de los LLMs, incluyendo el uso de un sistema RAG, la herramienta PlantUML [34] y la evaluación con de la respuesta de los LLMs con la métrica ROUGE. En la Sección 4 se presentan los resultados experimentales obtenidos. La Sección 5 ofrece una discusión de los resultados obtenidos, así como las conclusiones derivadas del estudio comparativo.

2. Estado del arte

Recientes investigaciones han explorado el uso de LLMs en la generación automática de diagramas UML, con el objetivo de evaluar su capacidad para complementar o mejorar las prácticas manuales tradicionales empleadas en el modelado de software. En este contexto, la atención se ha centrado en el análisis de la coherencia estructural y la utilidad práctica de los diagramas generados por estos modelos.

El estudio realizado por De Bari *et al.* [32] compara la calidad de los diagramas UML de clases generados por ChatGPT-4 con aquellos elaborados por expertos, en cuanto a sintaxis, semántica y pragmática. Utilizaron reglas específicas para identificar errores en cada aspecto, y también midieron la distancia semántica entre los diagramas generados y una solución de referencia. Además, consideraron factores como la dificultad, el tamaño y la legibilidad de los requisitos en lenguaje natural para analizar su impacto en la calidad del resultado. Los resultados mostraron que el LLM cometió más errores, especialmente en calidad semántica, y sus soluciones fueron menos precisas que las generadas por los expertos.

Por otra parte, la investigación realizada por Ferrari *et al.* [33], evaluó la capacidad de ChatGPT para generar diagramas de secuencia UML a partir de requisitos redactados en lenguaje natural, utilizando veintiocho documentos de requisitos de distintos dominios. A través de un análisis cualitativo se observó que, aunque los diagramas generados eran comprensibles y cumplían con los estándares de UML, presentaron deficiencias en exactitud y cobertura, como omisión de elementos esenciales e inconsistencias en su estructura. Estas deficiencias fueron más evidentes en requisitos ambiguos o de baja calidad. Los autores sugirieron complementar el uso de LLMs con estrategias especializadas, como la Ingeniería de *Prompts*, y un proceso iterativo de revisión y ajuste por parte de los analistas para mejorar la precisión de los diagramas generados.

Conrardy y Cabot [11] evaluaron el uso de GPT-4V, Gemini Pro, Gemini Ultra y CogVLM para generar diagramas de clases UML a partir de imágenes de diagramas dibujados a mano. Los resultados muestran que GPT-4V fue el modelo con mejor desempeño, sin errores de sintaxis en ninguna de sus salidas, mientras que CogVLM y los modelos Gemini presentaron más fallos, especialmente al manejar la sintaxis de PlantUML. Además, se evidenció una correlación entre la complejidad del diagrama y el número de errores generados. Los modelos Gemini, además, omitieron la generación de respuestas en algunos casos.

Estos trabajos de investigación destacan el potencial de los LLMs para transformar la generación de diagramas UML, mejorando la comprensión y comunicación entre los miembros del equipo de desarrollo y reduciendo la carga de trabajo en tareas de modelado UML. En la Tabla 1 se presenta una comparativa de los trabajos previamente mencionados, lo que permite visualizar las diferencias y similitudes entre los enfoques adoptados, facilitando una comprensión integral del estado actual de la investigación en esta área.

Tabla 1. Comparativa de los trabajos del estado del arte.

Criterio	De Bari *et al.* [20]	Ferrari *et al.* [21]	Conrardy y Cabot [22]
Diagrama	clases	secuencia	clases

Entrada	Requisitos en NL	Documentos de requisitos en NL	Imágenes de diagramas hechos a mano
LLM	ChatGPT-4	ChatGPT	GPT-4V, Gemini Pro, Ultra y CogVLM
Métricas	Calidad: sintáctica, semántica y pragmática	Completitud, exactitud, estándar, comprensibilidad y terminología	Número de errores por intento, modelo, solicitud y nivel

Fuente: Elaboración propia.

3. Comparación de LLMs para la generación de diagramas UML

En el enfoque de este estudio comparativo, se evalúa el desempeño de diferentes LLMs en la generación de diagramas UML a partir de historias de usuario. Para ello, se ha diseñado un proceso metodológico estructurado en tres etapas, con el objetivo de seleccionar los modelos más adecuados para esta tarea. Las etapas del proceso son:

1. Selección y preparación del *corpus* de historias de usuario
2. Implementación del sistema RAG para la generación de diagramas UML
3. Evaluación del desempeño de los LLMs

3.1. Selección y preparación del corpus de historias de usuario

El conjunto de historias de usuario empleado como entrada para este estudio fue generado a partir de diagramas de clases UML disponibles en el repositorio *Real World PlantUML*[35], los cuales fueron elaborados por expertos humanos, garantizando su diversidad de casos de uso, calidad de los diagramas y alineación con escenarios del mundo real. A partir de estos diagramas, se realizó la redacción de manera manual de las historias de usuario en inglés, siguiendo una estructura estándar del tipo: *As a [role], I want to [action], so that [goal]*. Si bien esta estrategia permitió mantener el control sobre la estructura y coherencia de los textos, también conlleva riesgos de ambigüedad, omisiones o interpretaciones subjetivas que podrían afectar los resultados. Para mitigar estos riesgos, se procuró asegurar la consistencia en los elementos clave de cada historia, lo que permitió una evaluación más uniforme entre los modelos.

El conjunto final incluye diez historias de usuario (ver Apéndice A) que abarcan diversos contextos funcionales, tales como gestión de usuarios, gestión de libros, operaciones bancarias, generación de documentos, entre otros.

3.2. Implementación del sistema RAG para la generación de diagramas UML

Se llevó a cabo la implementación de un sistema RAG con el objetivo de evaluar los LLMs seleccionados, aprovechando su capacidad para enriquecer la generación de texto mediante el acceso contextual a fuentes de información relevantes.

3.2.1. Revisión de LLMs actuales disponibles

Diversos estudios, como los de Kalyan [14], Yao *et al.* [15] y Zhao *et al.* [17], han llevado a cabo análisis comparativos del desempeño de diversos LLMs, permitiendo identificar aquellos con mayor reconocimiento y uso dentro del campo del NLP. A partir de esta revisión, se identificaron seis modelos relevantes por su impacto, características y disponibilidad: GPT de OpenAI, Gemini de Google, LLaMA de Meta, Claude de Anthropic, DeepSeek de DeepSeek AI y Mistral de Mistral AI.

Durante la revisión, se observaron varios criterios recurrentes que los estudios emplean para evaluar y comparar estos LLMs. Entre los más destacados se encuentran el tipo de acceso (abierto o propietario), el número de parámetros y la capacidad de adaptación a tareas específicas. Estos aspectos fueron señalados como determinantes tanto para el rendimiento de los modelos como para su aplicabilidad en distintos contextos.

En particular, se identificó que los modelos de código abierto como LLaMA, DeepSeek y Mistral, son valorados por su flexibilidad, ya que permiten ser adaptados y reutilizados en tareas personalizadas. Esta capacidad

ha sido destacada por estudios como los de Minaee *et al.* [18] y Liu *et al.* [28], quienes resaltan su utilidad en aplicaciones que requieren una configuración específica del modelo.

Por otro lado, aunque los modelos de código propietario, como GPT, Gemini y Claude, no permiten modificaciones, se destacan por su excelente desempeño en tareas de generación de contenido estructurado. Este rendimiento sobresaliente ha sido señalado en diversos estudios, como los de Kalyan [14] y Zhao *et al.* [17], quienes destacan la capacidad de estos modelos para abordar tareas complejas con alta precisión. Además, el incremento en el número de parámetros de cada modelo influye de manera significativa en su capacidad para generar contenido detallado y bien organizado, lo cual resulta esencial para la creación de diagramas UML. Este aspecto ha sido destacado por estudios como los de Meng *et al.* [32] y De Bari *et al.* [33], quienes subrayan la importancia de un mayor número de parámetros para mejorar la calidad y precisión en tareas complejas como la generación de diagramas UML.

2.2.2. Definición de los criterios de selección

A partir de la revisión de los LLMs disponibles en la literatura, se identificaron una serie de aspectos recurrentes utilizados para evaluar su desempeño y aplicabilidad en diversas tareas. Estos hallazgos permitieron establecer los criterios de selección empleados en este estudio, con el objetivo de identificar los modelos más adecuados para la tarea de generar diagramas UML a partir de historias de usuario. Los criterios fueron los siguientes:

- **Número de parámetros:** El tamaño de un modelo, medido por el número de parámetros, es un factor crucial en su desempeño. Los modelos más grandes, debido a su mayor capacidad de aprendizaje, tienden a ser más efectivos en tareas complejas, como la generalización de patrones. Sin embargo, un mayor número de parámetros también puede llevar a una mayor demanda computacional. Por lo tanto, se priorizaron modelos que ofrecieran un buen balance entre tamaño y desempeño, asegurando que su capacidad para resolver el problema fuera eficiente sin comprometer el uso de recursos.
- **Disponibilidad de código:** La distinción entre modelos de código abierto y propietarios fue clave. Los modelos de código abierto fueron preferidos por su accesibilidad y costo nulo, lo que permitió el estudio sin restricciones económicas. No obstante, también se incluyeron modelos propietarios debido a su popularidad y capacidades avanzadas en la industria, permitiendo así una comparación entre accesibilidad, costo, rendimiento y características distintivas de ambos tipos.
- **Capacidad de ajuste (*Fine-Tuning*):** La posibilidad de realizar un ajuste en el modelo fue crucial para poder adaptar su comportamiento a tareas particulares, como la generación de diagramas UML a partir de historias de usuario.
- **Capacidad computacional requerida:** La ejecución local de LLMs requiere una capacidad de procesamiento elevada, que incluye múltiples GPUs con amplia memoria y sistemas de almacenamiento rápido. Dado que el equipo disponible no contaba con estos recursos, se priorizó el uso de modelos que pudieran ser desplegados dentro de las limitaciones técnicas existentes, asegurando un equilibrio adecuado entre desempeño y viabilidad operativa.

Los LLMs seleccionados para la realización del presente estudio comparativo son los mostrados en la Tabla 2. Esta selección responde a los criterios previamente definidos, con el objetivo de representar una variedad de enfoques en cuanto a arquitectura y accesibilidad, asegurando así una evaluación equilibrada y representativa de sus capacidades en la tarea de generación de diagramas de clases UML. En el caso de los modelos de código propietario, se utilizaron las versiones más actuales disponibles al momento del estudio, mientras que, para los modelos de código abierto, se eligieron aquellas versiones que se adaptaran a los requerimientos computacionales del equipo utilizado para la implementación del sistema RAG y las pruebas realizadas.

Tabla 2. Características de los LLMs seleccionados.

Modelo	Número de parámetros	Disponibilidad de código	Capacidad de ajuste	Capacidad computacional requerida
GPT-4	1.7T	X	X	-
Claude-Sonnet-4	-	X	X	-
Gemini-2.5-Pro	-	X	X	-
Llama3.2	3B	✓	✓	2.0GB

DeepSeek-R1	8B	✓	✓	5.2GB
Mistral	7B	✓	✓	4.1GB

Fuente: Elaboración propia.

2.2.3. Implementación del sistema RAG

A continuación, se describen los componentes clave del sistema RAG implementado (ver Apéndice B), su flujo de ejecución, la construcción del *prompt* utilizado y las reglas a seguir para la construcción de los diagramas de clases, para obtener resultados en sintaxis PlantUML:

- **Arquitectura general:** La arquitectura del sistema sigue un enfoque modular en Python, compuesto por cinco etapas principales: carga de documentos, indexación semántica, recuperación contextual, generación del diagrama mediante un LLM y evaluación del diagrama generado. Se integraron tecnologías, como FAISS para la construcción del índice vectorial para la indexación semántica, en cuanto a los métodos de incrustación se utilizó el modelo all-MiniLM-L6-v2 de *HuggingFace* integrado a través de *LangChain*, PlantUML para la renderización de los diagramas generados y la métrica ROUGE para evaluar la similitud entre los diagramas generados y los diagramas de referencia (ver Apéndice C).
- **Preparación de documentos:** Para cada HU, se carga la fuente de información en formato PDF, que contiene el contexto del proyecto correspondiente a la HU. El documento se carga y divide en fragmentos semánticamente coherentes mediante un segmentador recursivo. Estos fragmentos se vectorizan con el modelo all-MiniLM-L6-v2 y se almacenan en un índice FAISS, lo que permite recuperar información relevante de manera eficiente durante la generación de diagramas.
- **Construcción del *prompt* y reglas de modelado:** Con el objetivo de estandarizar la generación y asegurar la validez sintáctica de los diagramas en PlantUML, se diseñó un *prompt* en inglés (ver Apéndice D). Este *prompt* contiene instrucciones para el LLM sobre el formato esperado, el lenguaje de salida y las reglas del modelado de diagramas de clases. Durante el diseño del *prompt* se realizaron cinco iteraciones para mejorar la generación de diagramas de clases. Inicialmente, el *prompt* en español produjo resultados inconsistentes, con omisiones y errores en la sintaxis. Al incorporar instrucciones más específicas, como el uso obligatorio de sintaxis PlantUML y limitar la salida a elementos clave, se mejoró la precisión y estructura de los diagramas generados.
- **Recuperación y generación con los LLMs:** El componente de generación del sistema se fundamenta en la clase *RetrievalQA* de la biblioteca *LangChain*, la cual facilita la integración entre la recuperación de información y la generación de texto. Esta clase permite consultar el *vectorstore* construido previamente para recuperar fragmentos textuales relevantes que contienen información contextual clave para cada historia de usuario. Dichos fragmentos actúan como contexto adicional para el modelo generativo. El proceso RAG integra el contexto recuperado, el *prompt* diseñado y la historia de usuario, enviando esta entrada compuesta al LLM seleccionado.
- **Renderización y almacenamiento:** Las respuestas generadas son almacenadas en archivos.puml y .txt, organizados según el modelo, el tipo de diagrama y el identificador de la historia de usuario. Por lo tanto, mediante una invocación a PlantUML utilizando Java, se realiza la renderización automática de estos archivos, generando imágenes en formato .png, lo que permitió una inspección visual de la calidad estructural y semántica de los diagramas de clases generados, facilitando tanto el análisis cuantitativo como cualitativo.

2.3. Evaluación del desempeño de los modelos

Para realizar la evaluación de la calidad de los diagramas de clases generados, en este estudio comparativo se optó por utilizar la métrica de ROUGE (*Recall-Oriented Understudy for Gisting Evaluation*), una métrica ampliamente reconocida en la evaluación de resúmenes y generación de texto, propuesta por Lin [36]. Esta métrica se ha consolidado como un método eficaz para comparar textos generados de manera automática contra textos de referencia elaborados manualmente. Aunque originalmente fue diseñada para la evaluación de resúmenes,

investigaciones recientes han demostrado su aplicabilidad en tareas de generación de texto estructurado, incluyendo modelado de diagramas en formato textual [28].

En el presente estudio comparativo, los diagramas de clases generados están orientados al análisis, lo cual implica que su objetivo principal es facilitar la comprensión y comunicación de conceptos clave entre los miembros del equipo de desarrollo, y no necesariamente representar un diagrama técnico detallado.

Dado que los diagramas de clases generados en este estudio se expresan en lenguaje textual usando sintaxis PlantUML, se empleó la métrica ROUGE para evaluar su similitud con los diagramas de referencia. En particular, se utilizó la variante ROUGE-L, ya que a diferencia de otras variantes como ROUGE-1 o ROUGE-2, que se centran únicamente en la coincidencia de unigramas o bigramas, ROUGE-L captura relaciones de orden y estructura dentro del texto, lo cual es fundamental para reflejar con mayor fidelidad la calidad de diagramas donde el orden y jerarquía de clases, atributos y relaciones es relevante. Por esta razón, ROUGE-L se consideró la variante más apropiada para la evaluación en el contexto de este estudio:

- **ROUGE-L:** Evalúa la longitud de la subsecuencia común más larga (*Longest Common Subsequence, LCS*) entre el texto generado y el texto de referencia. A diferencia de otras métricas, no requiere coincidencias consecutivas, sino que identifica aquellas palabras que mantienen el mismo orden relativo en la secuencia. Esta característica permite capturar la estructura global del contenido sin necesidad de definir previamente la longitud de los n-gramas [37].

La variante ROUGE-L se calcula a partir de tres medidas fundamentales:

- **Precisión:** Mide la proporción de coincidencias correctas respecto a los elementos generados [38].
- ***Recall*:** Mide la proporción de coincidencias correctas respecto a los elementos esperados en la referencia [39].
- **Medida F1:** Es la combinación de la precisión y el *recall* en un solo valor, representando un equilibrio entre ambos [40].

Para la evaluación del desempeño de los LLMs en la generación de diagramas de clases, se empleó un enfoque basado en un sistema RAG. La métrica ROUGE se aplicó al contexto específico de diagramas de clases, representados en formato PlantUML, facilitando una comparación entre los diagramas generados y los diagramas de referencia. La evaluación se centró específicamente en la medida de *recall*, dado que esta métrica refleja la capacidad del modelo para capturar la mayor cantidad posible de elementos relevantes presentes en los diagramas de referencia.

Aunque ROUGE-L también permite calcular precisión y medida F1, en este estudio se optó por centrarse únicamente en *recall* debido a la naturaleza de la tarea. El objetivo principal fue evaluar en qué medida los LLMs eran capaces de recuperar los elementos esperados en los diagramas de referencia, es decir, clases, atributos y relaciones relevantes. En este contexto, el uso de precisión y medida F1 podría penalizar innecesariamente a los modelos que generan elementos adicionales que, si bien no aparecen en el diagrama de referencia, podrían ser válidos o incluso útiles.

Para la evaluación se seleccionaron LLMs tanto de código abierto como propietarios, considerando su disponibilidad, arquitectura y facilidad de integración con el sistema RAG implementado. Algunos modelos fueron ejecutados localmente mediante Ollama, mientras que otros se utilizaron a través de una API Key. En la Tabla 3 se indica el tipo de acceso de los LLMs seleccionados.

Tabla 3. Tipo de acceso de los LLMs evaluados.

Modelo	Tipo de acceso
GPT-4	Remoto (API KEY pagada - OpenAI)
Claude-Sonnet-4	Remoto (API KEY pagada - Anthropic)
Gemini-2.5-Pro	Remoto (API KEY pagada - Google)
Llama3.2	Local (Ollama 0.7.0)
DeepSeek-R1	Local (Ollama 0.7.0)
Mistral	Local (Ollama 0.7.0)

Fuente: Elaboración propia.

La inclusión de los LLMs tanto de código abierto ejecutados de manera local como de soluciones comerciales ofrecidas por proveedores reconocidos permitió explorar distintos enfoques disponibles en la actualidad. Esta

diversidad ayudó a realizar una evaluación más equilibrada y representativa del desempeño de los LLMs en la generación de diagramas de clases orientados al análisis.

4. Resultados

En esta sección se presentan los resultados obtenidos tras la evaluación de los LLMs en la generación de diagramas de clases UML a nivel de análisis para un conjunto de diez historias de usuario. Para evaluar el desempeño de los LLMs en la tarea de la generación de diagramas de clases UML a partir de historias de usuario, se llevaron a cabo un conjunto de experimentos en dos modalidades: utilizando el sistema RAG implementado y sin el uso del sistema RAG, es decir realizando la consulta directa al LLM. En ambos escenarios, se aplicaron diez casos de prueba correspondientes a un conjunto de historias de usuario previamente definidas. Cada uno de los modelos seleccionados fue evaluado en estas dos condiciones, generando un total de diez salidas por modalidad, con el fin de observar de forma sistemática su comportamiento. La calidad de los diagramas generados fue medida utilizando la métrica ROUGE-L, haciendo énfasis en el valor promedio de la medida *recall*, dado su enfoque en la recuperación de elementos clave del diagrama de referencia.

4.1. Resultados sin el uso del sistema RAG

En esta primera fase del experimento, se evaluó el desempeño de los LLMs sin la intervención del sistema RAG, es decir, enviando directamente las historias de usuario como parte del *prompt*. Es importante señalar que, aunque no se utilizó la recuperación del contexto adicional, la historia de usuario fue incluida explícitamente en la consulta enviada al modelo, asegurando que cada LLM contará con la información mínima necesaria para generar el correspondiente diagrama de clases UML.

Para cada modelo LLM se realizaron diez casos de prueba, correspondientes a un conjunto de historias de usuario previamente definidas. La calidad de los diagramas generados fue evaluada utilizando la métrica ROUGE-L, centrándose particularmente en el valor de la medida *recall*, con el fin de medir la proporción de elementos relevantes del diagrama de referencia que fueron correctamente representados por el modelo.

Los resultados obtenidos, promediando los valores de ROUGE-L con la medida *recall* en las diez ejecuciones por modelo, se presentan en la Tabla 4.

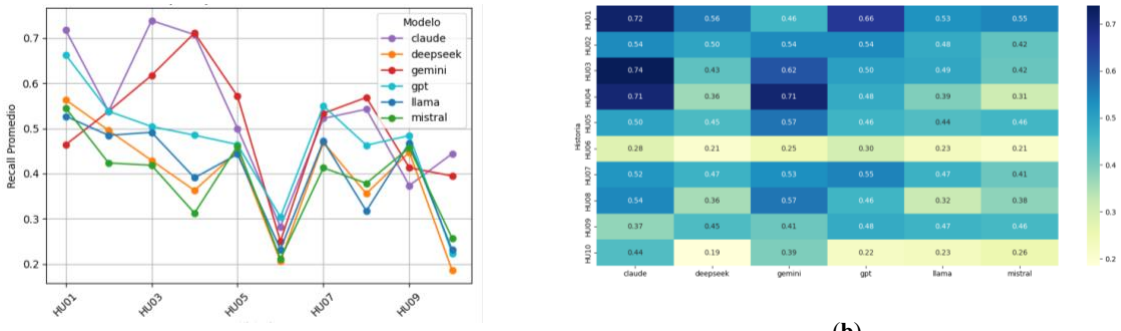
Tabla 4. Resultados promedio de ROUGE-L sin el uso del Sistema RAG.

Ranking	Modelo	Promedio ROUGE-L (*recall*)
1	Claude-Sonnet-4	0.537
2	Gemini-2.5-Pro	0.506
3	GPT-4	0.468
4	Llama3.2	0.406
5	DeepSeek-R1	0.396
6	Mistral	0.388

Fuente: Elaboración propia.

Los resultados muestran que Claude-Sonnet-4 alcanzó el mejor desempeño promedio, seguido por Gemini-2.5-Pro y GPT-4, evidenciando su capacidad para capturar con mayor fidelidad los elementos esperados del diagrama a partir de la historia de usuario proporcionada. En contraste, modelos como DeepSeek-R1 y Mistral presentaron un desempeño inferior, posiblemente debido a limitaciones en su habilidad para interpretar y estructurar adecuadamente la información textual sin un mecanismo de recuperación contextual adicional.

En la Figura 2 se presentan dos gráficos distintos sobre los resultados obtenidos sin el uso del Sistema RAG, en donde se observa que el modelo Claude-Sonnet-4 destaca con el mejor rendimiento promedio en la mayoría de las historias de usuario, alcanzando valores cercanos a 0.7, especialmente en HU03 y HU04, mientras que Llama3.2 y Mistral presentan resultados más bajos y menos consistentes.



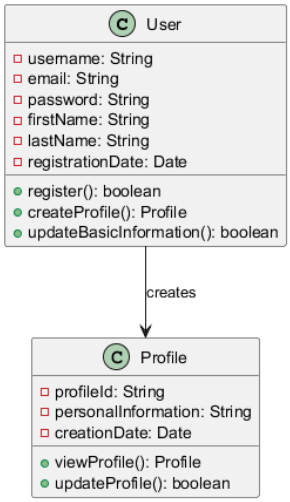
(**a**) (**b**)

Figura 2. Resultados obtenidos sin el uso del sistema RAG. (a) Gráfico de líneas; (b) Mapa de calor.

En la Tabla 5 se presenta el diagrama de clases correspondiente a la HU01, el cual fue generado por el modelo Claude-Sonnet-4. Este modelo logró representar de manera significativa los elementos clave descritos en la historia de usuario, alcanzando un valor de *recall* de 0.818. Este resultado indica que el modelo fue capaz de recuperar una proporción considerable de los conceptos y relaciones esperados en el diagrama de referencia, evidenciando un desempeño sólido en términos de cobertura de la información relevante para este caso en particular.

Tabla 5. Diagrama generado por Claude-Sonnet-4 para la HU01.

Código	Código en PlantUML	Diagrama generado

HU01/ Claude-Sonnet-4	@startuml
	class User {
	- username: String
	- email: String
	- password: String
	- firstName: String
	- lastName: String
	- registrationDate: Date
	+ register(): boolean
	+ createProfile(): Profile
	+ updateBasicInformation(): boolean
	}
	
	class Profile {
	- profileId: String
	- personalInformation: String
	- creationDate: Date
	+ viewProfile(): Profile
	+ updateProfile(): boolean
	}
	
	User --> Profile : creates
	@enduml
		

Fuente: Elaboración propia.

4.2. Resultados con el uso del sistema RAG

En esta segunda fase, se incorporó el sistema RAG como parte del proceso de generación. A diferencia de la evaluación anterior, en esta configuración los modelos recibieron una consulta enriquecida con contexto relevante, además de la historia de usuario, con el fin de mejorar la calidad de los diagramas de clases generados.

Al igual que en la primera fase, se utilizaron las mismas diez historias de usuario y se realizaron diez ejecuciones por modelo, evaluando los resultados mediante la variante ROUGE-L y la medida *recall*. Los resultados promedio obtenidos se presentan en la Tabla 6.

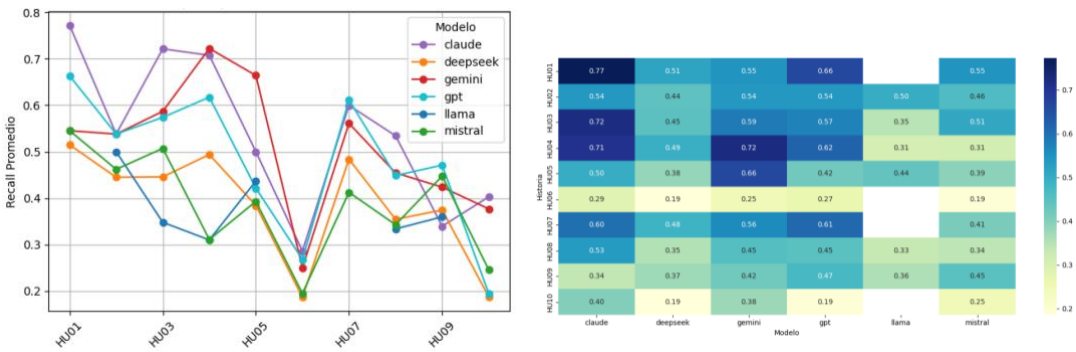
Tabla 6. Resultados promedio de ROUGE-L con el uso del Sistema RAG.

Ranking	Modelo	Promedio ROUGE-L (*recall*)
1	Claude-Sonnet-4	0.540
2	Gemini-2.5-Pro	0.512
3	GPT-4	0.481
4	DeepSeek-R1	0.387
5	Mistral	0.386
6	Llama3.2	0.381

Fuente: Elaboración propia.

Los resultados obtenidos al incorporar el sistema RAG muestran valores de ROUGE-L (*recall*) similares a los registrados en la evaluación sin RAG. En modelos como Claude-Sonnet-4, Gemini-2.5-Pro y GPT-4, se observaron variaciones mínimas en los promedios, sin cambios significativos en su comportamiento general. De manera similar, DeepSeek-R1, Mistral y Llama3.2 presentaron puntuaciones comparables entre ambos enfoques. Estos resultados sugieren que la inclusión de contexto adicional no tuvo un impacto considerable en la recuperación de información, al menos bajo las condiciones evaluadas.

En la Figura 3 se presentan dos gráficos distintos sobre los resultados obtenidos con el uso del Sistema RAG, en donde se observa que los modelos Claude-Sonnet-4 y Gemini-2.5-Pro consistentemente presentan los valores más altos, destacándose por su capacidad para generar diagramas de clases UML más precisos a partir de las historias. Modelos como GPT-4 también muestran buenos resultados, especialmente en historias específicas como HU01 y HU09. En contraste, modelos como DeepSeek-R1, Mistral y Llama3.2 muestran mayor variabilidad y valores promedio más bajos, indicando menor consistencia en la calidad de la generación con el sistema RAG.



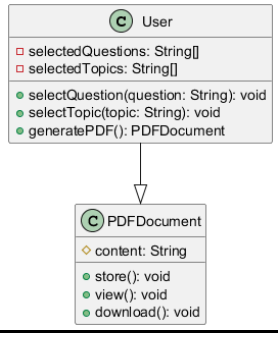
(**a**) (**b**)

Figura 3. Resultados obtenidos con el uso del sistema RAG. (a) Gráfico de líneas; (b) Mapa de calor.

En cuanto al contexto recuperado por el sistema RAG, se observó que, aunque la mayoría de los fragmentos resultaron relevantes, algunos no contenían la información necesaria para guiar de manera adecuada la generación del diagrama de clases. Esta inconsistencia en la calidad del contexto recuperado afectó directamente los resultados, lo que evidenció la necesidad de mejorar los mecanismos de recuperación de información. La Tabla 7 presenta el diagrama generado por el modelo GPT-4, como parte del sistema RAG, para la HU06, el cual obtuvo un valor de *recall* de 0.818.

Tabla 7. Diagrama generado por GPT-4 como parte del Sistema RAG para la HU06.

Código	Código en PlantUML	Diagrama generado

HU06/ GPT-4	@startuml
	class User {
	- selectedQuestions: String[]
	- selectedTopics: String[]
	+ selectQuestion(question: String): void
	+ selectTopic(topic: String): void
	+ generatePDF(): PDFDocument
	}
	
	class PDFDocument {
	# content: String
	+ store(): void
		

```
    + view(): void
    + download(): void
}

User --|> PDFDocument
@enduml
```

Fuente: Elaboración propia.

4.3. Resultados comparativos con y sin sistema RAG

Para facilitar la comparación del desempeño de los modelos con y sin el sistema RAG, se calculó el porcentaje de cambio relativo en el promedio de la métrica ROUGE-L (*recall*) para cada modelo. La Tabla 8 resume estos resultados, permitiendo identificar de forma más precisa el impacto que tuvo el uso del sistema RAG en cada caso. Como se puede observar, los valores de variación son en su mayoría bajos, lo que sugiere que el sistema RAG no produjo mejoras significativas en términos de recuperación léxica de acuerdo con la métrica empleada. En algunos modelos, como Claude-Sonnet-4 y Gemini-2.5-Pro, la diferencia fue inferior al 1%, mientras que, en otros, como Llama3.2, se observó una leve disminución en el desempeño.

Es importante destacar que los resultados obtenidos reflejan particularidades específicas del contexto y los datos empleados en este estudio comparativo, por lo que no pueden considerarse una evaluación definitiva sobre el desempeño del enfoque RAG. Si bien en este caso no se logró un impacto significativo, ello puede deberse a factores como la naturaleza de las historias de usuario, la calidad del contexto recuperado y las características particulares de los modelos evaluados.

Tabla 8. Porcentaje de variación en ROUGE-L con el uso del Sistema RAG.

Modelo	Promedio sin RAG	Promedio con RAG	% de cambio
Claude-Sonnet-4	0.537	0.540	+0.56%
Gemini-2.5-Pro	0.506	0.512	+1.18%
GPT-4	0.468	0.481	+2.78%
DeepSeek-R1	0.406	0.387	–2.27%
Mistral	0.396	0.386	–0.52%
Llama3.2	0.388	0.381	–6.16%

Fuente: Elaboración propia.

Estos resultados permiten concluir que, bajo las condiciones de este experimento, la inclusión del sistema RAG no implicó una mejora sustancial en la recuperación de elementos relevantes en los diagramas generados, según la variante ROUGE-L (*recall*). Esta observación sugiere que el impacto del sistema RAG podría estar condicionado no únicamente por el modelo utilizado, sino por la relevancia y utilidad del contexto proporcionado durante la recuperación. Por lo tanto, resulta fundamental seguir explorando estrategias que permitan mejorar la calidad del contenido recuperado, de modo que éste aporte información realmente significativa para la generación precisa de diagramas UML a partir de historias de usuario.

La Tabla 9 presenta un resumen de los costos estimados y los tiempos promedio de ejecución para cada modelo evaluado. Esta información complementa el análisis de desempeño, permitiendo valorar no solo la calidad de los resultados, sino también la eficiencia y la viabilidad económica de su uso en entornos prácticos.

Tabla 9. Tiempo y costo promedio por LLM.

Modelo	Tiempo Promedio	Costo Promedio
Claude-Sonnet-4	10.81 s	~$0.0063
Gemini-2.5-Pro	35.66 s	~$0.0091
GPT-4	7.82 s	~$0.017
DeepSeek-R1	47.85 s	-
Mistral	15.72 s	-
Llama3.2	6.46 s	-

Fuente: Elaboración propia.

Todos los diagramas generados, incluyendo los correspondientes a cada una de las ejecuciones, se encuentran disponibles en el repositorio indicado para su consulta completa ver el Apéndice E.
Cabe señalar que, durante el análisis de los diagramas generados, se identificaron errores sintácticos recurrentes que limitaron su correcta visualización y afectaron el proceso de evaluación automatizada. Entre los errores más comunes se encontraron: omisión de llaves en la definición de clases (como *class Player* sin {} en HU05 por Llama3.2), relaciones con conectores inválidos (--> en lugar de o--, como en HU07 con Mistral), uso incorrecto de tipos (*String!, Text*), visibilidades no reconocidas (*), e inclusión de atributos fuera del cuerpo de la clase.

5. Conclusiones y Discusiones

La presente investigación se centró en la comparación de distintos LLMs antes mencionados como parte de un sistema RAG orientado a la generación automática de diagramas de clases UML a partir de historias de usuario. El objetivo principal fue evaluar qué tan efectivamente cada modelo puede aprovechar información contextual recuperada para generar representaciones UML coherentes y relevantes.

Los resultados evidencian que los modelos propietarios presentaron un rendimiento superior, con valores de ROUGE-L más consistentes y salidas generalmente más estructuradas y alineadas con las historias de usuario. En cambio, los modelos de código abierto mostraron un desempeño más bajo, tanto en calidad de las respuestas como en la coherencia del código UML generado.

Al comparar el desempeño con y sin la implementación del sistema RAG, se observa que la inclusión de un contexto externo recuperado aporta un marco de información que, en teoría, debería mejorar la generación del diagrama UML. Sin embargo, en esta investigación, la mejora obtenida con el sistema RAG fue limitada posiblemente a la calidad y relevancia del contexto proporcionado. Esto podría indicar que el beneficio de incorporar RAG depende en gran medida de la eficacia del módulo de recuperación de información.

Por lo tanto, si bien el sistema RAG tiene el potencial de enriquecer la capacidad de los LLMs para tareas específicas como la generación de diagramas UML, su impacto real está condicionado por la precisión y adecuación del contexto recuperado. En ausencia de un contexto relevante y bien construido, los modelos no logran explotar completamente las ventajas de la arquitectura RAG, resultando en un rendimiento similar al de los modelos sin contexto externo.

La incorporación de la arquitectura RAG se propuso con la expectativa de mejorar la calidad de los diagramas generados, al suministrar a los modelos LLM información contextual adicional que facilitara una interpretación más precisa de las historias de usuario. Esta arquitectura buscaba mitigar las limitaciones de los LLMs, proporcionando un contexto enriquecido capaz de capturar detalles relevantes y relaciones implícitas en los requerimientos. Sin embargo, los resultados obtenidos muestran que es fundamental mejorar los procesos de recuperación y filtrado del contexto para asegurar que la información proporcionada al modelo LLM coincida adecuadamente con lo solicitado en las historias de usuario. Cuando el contexto incluye datos irrelevantes o no relacionados directamente con los requerimientos, se dificulta la generación de diagramas de clases UML adecuados y coherentes. Por ello, es necesario mejorar los métodos para recuperar y filtrar solo la información pertinente, para así aprovechar mejor la arquitectura RAG y obtener diagramas de clases más fieles a las necesidades requeridas.

Además, esta investigación se basó exclusivamente en métricas automáticas como ROUGE-L para evaluar la similitud entre los diagramas generados y sus referentes esperados. Si bien estas métricas ofrecen una medida cuantitativa útil, no capturan aspectos semánticos o de utilidad práctica de los diagramas generados en contextos reales de desarrollo. Por tanto, aunque este estudio se centró en evaluaciones cuantitativas, la incorporación de una evaluación cualitativa por parte de expertos humanos en investigaciones futuras enriquecería la valoración, permitiendo apreciar aspectos como la claridad del modelo, su adecuación al dominio, la utilidad para los desarrolladores y su completitud funcional. Este enfoque complementario contribuiría a obtener una visión más integral sobre la calidad de los resultados producidos por los modelos LLM bajo diferentes configuraciones.

Una limitación importante de este estudio es el número reducido de historias de usuario (díez), lo que puede restringir la aplicabilidad de los resultados a contextos más amplios. Por ello, futuros estudios deben considerar conjuntos de historias de usuario más extensos para fortalecer la validez y mejorar los resultados actualmente obtenidos.

6. Referencias

[1] Kotti, Z., Galanopoulou, R., Spinellis, D. (2023). Machine learning for software engineering: A tertiary study. *ACM Computing Surveys, 55* (12), 1-39. https://doi.org/10.1145/3572905

[2] Hoffmann, M., Mendez, D., Fagerholm, F., Luckhardt, A. (2022). The human side of software engineering teams: an investigation of contemporary challenges. *IEEE Transactions on software engineering, 49* (1), 211-225. https://doi.org/10.1109/TSE.2022.3148539

[3] Ferrario, M. A., Winter, E. (2022). Applying human values theory to software engineering practice: Lessons and implications. *IEEE Transactions on Software Engineering, 49* (3), 973-990. https://doi.org/10.1109/TSE.2022.3170087

[4] Altameem, E. A. (2015). Impact of agile methodology on software development. *Computer and Information Science, 8* (2), 9-14. https://doi.org/10.5539/cis.v8n2p9

[5] Gupta, A., Poels, G., Bera, P. (2022). Using conceptual models in agile software development: A possible solution to requirements engineering challenges in agile projects. *IEEE Access, 10,* 119745-119766. https://doi.org/10.1109/ACCESS.2022.3221428

[6] Jin, H., Huang, L., Cai, H., Yan, J., Li, B., Chen, H. (2024). From llms to llm-based agents for software engineering: A survey of current, challenges and future. *arXiv preprint*. https://doi.org/10.48550/arXiv.2408.02479

[7] Feng, S., Chen, C. (2024, February). *Prompting is all you need: Automated android bug replay with large language models*. 46th IEEE/ACM International Conference on Software Engineering. Lisbon, Portugal. https://doi.org/10.1145/3597503.3608137

[8] Wang, C., Pastore, F., Goknil, A., Briand, L. C. (2020). Automatic generation of acceptance test cases from use case specifications: an nlp-based approach. *IEEE Transactions on Software Engineering, 48* (2), 585-616. https://doi.ieeecomputersociety.org/10.1109/TSE.2020.2998503

[9] Liu, Z., Qian, P., Wang, X., Zhuang, Y., Qiu, L., Wang, X. (2021). Combining graph neural networks with expert knowledge for smart contract vulnerability detection. *IEEE Transactions on Knowledge and Data Engineering, 35* (2), 1296-1310. https://doi.org/10.1109/TKDE.2021.3095196

[10] van Remmen, J. S., Horber, D., Lungu, A., Chang, F., van Putten, S., Goetz, S., Wartzack, S. (2023). *Natural language processing in requirements engineering and its challenges for requirements modelling in the engineering design domain*. International Conference on Engineering Design (ICED). Bordeaux, France. https://doi.org/10.1017/pds.2023.277

[11] Conrardy, A., Cabot, J. (2024). From image to uml: First results of image based uml diagram generation using llms. *arXiv preprint*. https://doi.org/10.48550/arXiv.2404.11376

[12] Zheng, Z., Ning, K., Zhong, Q., Chen, J., Chen, W., Guo, L., Wang, W., Wang, Y. (2025). Towards an understanding of large language models in software engineering tasks. *Empirical Software Engineering, 30* (2). https://doi.org/10.1007/s10664-024-10602-0

[13] Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., Wang, H. (2024). Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology, 33* (8), 1-79. https://doi.org/10.1145/3695988

[14] Kalyan, K. S. (2024). A survey of GPT-3 family large language models including ChatGPT and GPT-4. *Natural Language Processing Journal, 6*, 1-48. https://doi.org/10.1016/j.nlp.2023.100048

[15] Yao, Y., Duan, J., Xu, K., Cai, Y., Sun, Z., Zhang, Y. (2024). A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing, 4* (2), 1-21. https://doi.org/10.1016/j.hcc.2024.100211

[16] Sonoda, Y., Kurokawa, R., Nakamura, Y., Kanzawa, J., Kurokawa, M., Ohizumi, Y., Gonoi, W., Abe, O. (2024). Diagnostic Performances of GPT-4o, Claude 3 Opus, and Gemini 1.5 Pro in "Diagnosis Please" Cases. *Japanese Journal of Radiology, 42*, 1231-1235. https://doi.org/10.1007/s11604-024-01619-y

[17] Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., … Wen, J.-R. (2023). A Survey of Large Language Models. *arXiv preprint*. http://arxiv.org/abs/2303.18223

[18] Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., Gao, J. (2024). *Large Language Models: A Survey*. *arXiv preprint*. http://arxiv.org/abs/2402.06196

[19]Islam, R., Ahmed, I. (2024). *Gemini-the most powerful LLM: Myth or Truth*. IEEE 5th Information Communication Technologies Conference (ICTC). Nanjing, China. https://doi.org/10.1109/ICTC61510.2024.10602253

[20]Lewandowska-Tomaszczyk, B., Liebeskind, C. (2024). Opinion events and stance types: advances in LLM performance with ChatGPT and Gemini. *Lodz Papers in Pragmatics, 20* (2), 413-432. https://doi.org/10.1515/lpp-2024-0039

[21]Kim, S., Huang, D., Xian, Y., Hilliges, O., Van Gool, L., Wang, X. (2024). *Palm: Predicting actions through language models*. European Conference on Computer Vision. Milan, Italy. https://doi.org/10.1007/978-3-031-73007-8_9

[22]Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., ... Piao, Y. (2024). Deepseek-v3 technical report. *arXiv preprint.* https://doi.org/10.48550/arXiv.2412.19437

[23]Tsai, H. C., Huang, Y. F., Kuo, C. W. (2024). Comparative analysis of automatic literature review using mistral large language model and human reviewers. *Research Square*. https://doi.org/10.21203/rs.3.rs-4022248/v1

[24]Yang, A., Yu, B., Li, C., Liu, D., Huang, F., Huang, H., ... Zhang, Z. (2025). *Qwen2. 5-1m technical report*. *arXiv preprint.* https://doi.org/10.48550/arXiv.2501.15383

[25]Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., … Amodei, D. (2020). *Language Models are Few-Shot Learners*. 34th International Conference on Neural Information Processing Systems. Vancouver BC, Canada https://dl.acm.org/doi/abs/10.5555/3495724.3495883

[26]Martino, A., Iannelli, M., Truong, C. (2023). *Knowledge injection to counter large language model (LLM) hallucination*. European Semantic Web Conference. Hersonissos, Greece. https://doi.org/10.1007/978-3-031-43458-7_34

[27]Di Rocco, J., Di Ruscio, D., Di Sipio, C., Nguyen, P. T., Rubei, R. (2025). *On the use of large language models in model-driven engineering*: *Software and Systems Modeling, 24*, 923-948. https://doi.org/10.1007/s10270-025-01263-8

[28]Li, C., Liu, Z., Xiao, S., Shao, Y. (2023). Making Large Language Models A Better Foundation For Dense Retrieval. *arXiv preprint*. http://arxiv.org/abs/2312.15503

[29]Chen, J., Xiao, S., Zhang, P., Luo, K., Lian, D., Liu, Z. (2024). BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. *arXiv preprint*. http://arxiv.org/abs/2402.03216

[30]Meng, R., Liu, Y., Joty, S. R., Xiong, C., Zhou, Y., Yavuz, S. (2024). *SFR-Embedding-Mistral: Enhance Text Retrieval with Transfer Learning*. https://www.salesforce.com/blog/sfr-embedding/

[31]De Bari, D., Garaccione, G., Coppola, R., Torchiano, M., Ardito, L. (2024). *Evaluating Large Language Models in Exercises of UML Class Diagram Modeling*. 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. Barcelona, España. https://doi.org/10.1145/3674805.3690741

[32]Ferrari, A., Abualhaijal, S., Arora, C. (2024). *Model generation with LLMs: From requirements to UML sequence diagrams*. IEEE 32nd International Requirements Engineering Conference Workshops (REW). Reykjavik, Iceland. https://doi.org/10.1109/REW61692.2024.00044

[33]PlantUML. (2025). *PlantUML: Class diagrams*. https://real-world-plantuml.com/?type=class

[34]Lin, C. Y. (2004). *Rouge: A package for automatic evaluation of* summaries https://aclanthology.org/W04-1013/

[35]Lin, C. Y., Och, F. J. (2004). *Looking for a few good metrics: ROUGE and its evaluation* https://research.nii.ac.jp/ntcir/ntcir-ws4/NTCIR4-WN/OPEN/OPENSUB_Chin-Yew_Lin.pdf

[36]Lin, S., Wang, B., Huang, Z., Li, C. (2024). A Study on a Precision Geriatric Medical Knowledge Q&A Model Based on Retrieval-Augmented Generation. *Preprints*. https://www.preprints.org/manuscript/202412.2424/v1

[37]Liu, Y., Huang, L., Li, S., Chen, S., Zhou, H., Meng, F., ... Sun, X. (2023). Recall: A benchmark for llms robustness against external counterfactual knowledge. *arXiv preprint.* https://doi.org/10.48550/arXiv.2311.08147

[38]Rodríguez Limón, C. (2025). *Aumento de Datos Basado en Recursos Lingüísticos para RAG sobre Textos Legales en Español* [Trabajo de Fin de Grado]. Universidad Politécnica de Madrid, España. https://oa.upm.es/87925/1/TFG_CARLOS_RODRIGUEZ_LIMON.pdf

APÉNDICE A: Conjunto de historias de usuario
https://github.com/LoreMartinez/Resultados_estudio_comparativo/blob/main/Conjunto_de_Historias_de_Usuario.pdf

APÉNDICE B: Código de implementación del Sistema RAG
https://github.com/LoreMartinez/Resultados_estudio_comparativo/blob/main/Sistema_RAG.py

APÉNDICE C: Conjunto de diagramas de referencia:
https://github.com/LoreMartinez/Resultados_estudio_comparativo/blob/main/Diagramas_de_Referencia.pdf

APÉNDICE D: Diseño del prompt utilizado en el Sistema RAG
https://github.com/LoreMartinez/Resultados_estudio_comparativo/blob/main/Disenio_del_prompt.pdf

APÉNDICE E: Diagramas generados en las pruebas realizadas
https://github.com/LoreMartinez/Resultados_estudio_comparativo/tree/main/Resultados_con_sistema_RAG

https://github.com/LoreMartinez/Resultados_estudio_comparativo/tree/main/Resultados_sin_sistema_RAG