



UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY



FACULTAD DE
INGENIERÍA

Chatbot para la educación: un asistente conversacional sobre inteligencia artificial

Informe de Proyecto de Grado presentado por

Federico Gil, Agustina Moraes, William Tift

en cumplimiento parcial de los requerimientos para la graduación de la carrera
de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de
la República

Supervisores

Regina Motz
Libertad Tansini

Montevideo, 2 de julio de 2025



Chatbot para la educación: un asistente conversacional sobre inteligencia artificial por Federico Gil, Agustina Moraes, William Tift tiene licencia [CC Atribución 4.0](#).

Agradecimientos

Queremos comenzar agradeciendo a la Facultad de Ingeniería y a la Universidad de la República por brindarnos una formación de calidad, que nos preparó para ejercer como ingenieras e ingenieros con responsabilidad y solidez técnica. También agradecemos los recursos académicos y humanos que nos permitieron llevar adelante este proyecto.

Agradecemos especialmente a nuestras tutoras, Regina Motz y Libertad Tansini, por acompañarnos, guiarnos y brindarnos confianza durante todo este proceso. Su orientación fue fundamental para concretar este proyecto.

Queremos agradecer también a Soledad Caballero y Carolina Rodríguez, quienes asumieron el rol de usuarias expertas y aportaron su tiempo y conocimiento tanto en el refinamiento de los requerimientos como en la validación de la solución.

Asimismo, agradecemos a Antel por proveernos gratuitamente una máquina virtual en la cual desplegar nuestra solución, y a Delfos por facilitarnos sin costo el acceso a los modelos de OpenAI utilizados durante el desarrollo del proyecto.

Expresamos nuestro más profundo agradecimiento a nuestras familias, por haber sido nuestro pilar durante todos estos años de carrera. Gracias por priorizar siempre nuestra formación, por brindarnos el tiempo, el espacio y el apoyo necesarios para estudiar, y por acompañarnos en cada etapa con interés genuino. Su aliento constante fue fundamental para llegar hasta esta instancia.

Finalmente, agradecemos al tribunal por tomarse el tiempo de leer este informe y evaluar nuestro trabajo.

Resumen

La inteligencia artificial (IA), en particular la IA generativa, está transformando diversos sectores, incluyendo la educación, ofreciendo nuevas herramientas para mejorar los procesos de enseñanza y aprendizaje. Sin embargo, muchos docentes enfrentan desafíos para comprender e integrar estas tecnologías en su día a día.

Este proyecto de grado presenta el diseño e implementación de un chatbot generativo de código abierto, diseñado para actuar como asistente para docentes interesados en entender la IA generativa y explorar su aplicación y utilidad en su trabajo. A diferencia de otros asistentes conversacionales, este chatbot se enmarca específicamente en un contexto educativo, por lo que se puso especial énfasis en que tenga un comportamiento pedagógico, utilice fuentes curadas de acceso abierto al generar sus respuestas y provea referencias validas de las fuentes de información.

El chatbot se basa en un sistema de grandes modelos de lenguaje (LLMs) y fue integrado con un sistema de generación aumentada por recuperación (RAG) para que las respuestas estén basadas en un repositorio de documentos curados construido en este proyecto. Además, se restringe el dominio de *expertise* del chatbot a Inteligencia Artificial, por lo que es instruido a mantenerse en el tema y el repositorio fue poblado únicamente con recursos sobre inteligencia artificial en educación.

Los resultados incluyen la prototipación y validación de un chatbot para inteligencia artificial en educación (Chatbot Educativo) como una herramienta efectiva para apoyar a los docentes en la comprensión de la IA generativa, facilitando el acceso a información curada y aplicando estrategias pedagógicas a través de interacciones adaptativas y dialogadas. Se provee también como resultado de este trabajo una guía para la modificación del corpus o personalización del dominio específico sobre el cual actúa el chatbot.

Palabras clave: Chatbot, LLM, RAG

Índice general

1. Introducción	1
1.1. Objetivos del proyecto	2
1.2. Resultados obtenidos	2
1.3. Organización del documento	4
2. Conceptos Preliminares	5
2.1. Grandes Modelos de Lenguaje (LLMs)	5
2.2. Personalización de LLMs	6
2.2.1. Métodos de Personalización del Comportamiento	7
2.2.2. Métodos de Personalización del Conocimiento	8
2.2.3. Análisis Técnico de los Métodos de Personalización	12
2.3. Integración de modelos en un sistema informático	14
2.3.1. Entorno de ejecución	14
2.3.2. Formato de respuesta del modelo	15
2.3.3. Frameworks	16
2.3.4. Arquitecturas	18
2.3.5. Evaluación	20
3. Chatbot para la Educación	25
3.1. Requerimientos del Proyecto	25
3.1.1. Requerimientos Funcionales	25
3.1.2. Requerimientos No Funcionales	27
3.1.3. Análisis Técnico de los Requerimientos	28
3.2. Revisión de Antecedentes	31
3.2.1. Análisis de soluciones existentes	31
3.2.2. Artículos Revisados	31
3.2.3. Comentarios de la Revisión	36
3.3. Solución Planteada	37
3.3.1. Diseño y Arquitectura	37
3.3.2. Implementación	46
4. Evaluación y Resultados	55
4.1. Evaluación <i>Offline</i>	55
4.1.1. Objetivos de la Evaluación <i>Offline</i>	55

4.1.2.	Mecánica de la Evaluación <i>Offline</i>	56
4.1.3.	Aspectos no implementados y trabajos futuros	60
4.1.4.	Resultados	61
4.2.	Evaluación <i>Online</i>	65
4.2.1.	Objetivos de la Evaluación <i>Online</i>	66
4.2.2.	Mecánica de la Evaluación <i>Online</i>	66
4.2.3.	Resultados	68
4.3.	Evaluación con Usuarios	71
4.3.1.	Objetivos de la Evaluación con Usuarios	71
4.3.2.	Enfoque Propuesto	72
4.3.3.	Rúbrica de Evaluación	72
4.3.4.	Resultados	74
5.	Experimentación	79
5.1.	Experimentación sobre la arquitectura	79
5.1.1.	Primera arquitectura planteada	79
5.1.2.	Segunda arquitectura planteada	82
5.2.	Experimentación sobre la Componente <i>RAG</i>	87
5.2.1.	Técnicas de <i>Chunking</i>	87
5.2.2.	Modelos de <i>Embeddings</i> Evaluados para la Recuperación	88
5.2.3.	Comparativa de Desempeño de Modelos de Lenguaje	88
6.	Conclusiones y Trabajos Futuros	91
6.1.	Conclusiones	91
6.2.	Trabajos futuros	95
6.2.1.	Culminación de los requerimientos pendientes	95
6.2.2.	Ampliación de la evaluación <i>offline</i>	97
6.2.3.	Refuerzo de la accesibilidad y la usabilidad	97
6.2.4.	Optimización del subsistema <i>RAG</i>	97
6.2.5.	Mejoras en la extracción y almacenamiento de metadatos para el citado	100
6.2.6.	Ampliación del corpus técnico	100
6.2.7.	Investigaciones y proyecciones tecnológicas	101
	Referencias	103
A.	Investigación sobre Grandes Modelos de Lenguaje	111
A.0.1.	Categorías de LLMs	112
A.0.2.	Arquitectura de los LLMs	113
A.0.3.	Modelos de Embeddings	114
A.0.4.	Ventana de contexto	117
A.0.5.	Proceso de entrenamiento	118
A.0.6.	Inferencia en Grandes Modelos de Lenguaje	118
A.0.7.	Evaluación de Benchmarks y Métricas en LLMs	120
B.	Prompts	123

C. Respuestas anonimizadas de la evaluación con usuarios	131
D. Reseña histórica de métodos de evaluación	133
E. Técnicas de Query Analysis	135
F. Técnicas de Chunking para Retrieval-Augmented Generation	137

Capítulo 1

Introducción

En los últimos años la **inteligencia artificial (IA)** —y, muy especialmente, la *IA generativa* basada en Grandes Modelos de Lenguaje— ha logrado niveles de desempeño que la sitúan como una herramienta transformadora en prácticamente todos los sectores productivos y del conocimiento. En el ámbito educativo, estas tecnologías ofrecen un potencial inédito para personalizar la enseñanza, facilitar el acceso a recursos de calidad, analizar grandes volúmenes de datos sobre el aprendizaje de los estudiantes, aportar ideas y materiales innovadores para su uso en el aula, así como automatizar tareas administrativas de forma que docentes y estudiantes puedan concentrarse en actividades de mayor valor pedagógico.

No obstante, muchos docentes se enfrentan a dudas sobre cómo utilizar la IA, qué herramientas elegir y cuáles son sus implicaciones éticas, entre otras inquietudes. Además, los enfoques existentes para aprender sobre estas tecnologías pueden dificultar que los cuerpos docentes comprendan con rigor su funcionamiento, evalúen sus limitaciones y terminen por adoptarlas para su uso en el aula.

La problemática central radica en la existencia de una importante brecha de conocimiento entre el profesorado —que muchas veces no cuenta con la formación necesaria para comprender la mecánica de los Grandes Modelos de Lenguaje ni los riesgos asociados, como las alucinaciones, los sesgos o las implicaciones en materia de privacidad— y la ausencia de soluciones pedagógicas adecuadas. Los materiales disponibles, como videos o lecturas extensas, pueden resultar tediosos y no siempre responden a las dudas concretas de cada docente individual. Por otro lado, las plataformas comerciales basadas en chatbots suelen centrarse en brindar información sin proporcionar sus fuentes, lo que puede generar desconfianza, y rara vez integran estrategias didácticas que promuevan un aprendizaje activo y reflexivo.

1.1. Objetivos del proyecto

Considerando el escenario previamente descrito, este proyecto se plantea el siguiente objetivo general, del cual se desprenden los posteriores objetivos específicos.

Objetivo general

Diseñar e implementar un chatbot generativo de código abierto, basado en un corpus curado, que asista a docentes de nivel secundario y terciario en la comprensión de un nuevo dominio específico, en particular se trabaja con el dominio de la aplicación de la inteligencia artificial generativa en la educación.

Objetivos específicos

1. Realizar un relevamiento bibliográfico y un análisis de herramientas existentes relacionadas con chatbots educativos e inteligencia artificial generativa.
2. Diseñar una arquitectura adecuada que satisfaga los requerimientos funcionales y no funcionales establecidos, contemplando criterios como la mantenibilidad, escalabilidad y facilidad de integración.
3. Implementar un prototipo funcional utilizando, en la medida de lo posible, tecnologías de código abierto, siempre que esto no comprometa la calidad de los datos y sea técnicamente viable, considerando los requisitos de hardware, económicos y otras posibles restricciones técnicas.
4. Evaluar el prototipo con usuarios reales involucrados en el ámbito educativo, con el fin de analizar su efectividad, usabilidad y capacidad de adaptación a diferentes perfiles docentes.

1.2. Resultados obtenidos

El principal resultado de este proyecto es el diseño e implementación de un chatbot generativo de código abierto con las siguientes características:

- **Enfoque temático:** El chatbot se centra en responder preguntas dentro de un dominio específico. En particular, el prototipo desarrollado se enfoca en la inteligencia artificial aplicada a la educación. Los temas seleccionados para este prototipo incluyen: aspectos técnicos (como el funcionamiento de los modelos de lenguaje), cuestiones éticas, aplicaciones en distintos sectores y, en particular, su integración en contextos educativos. El objetivo principal es acompañar la formación del cuerpo docente en estos temas, facilitando el acceso a conocimiento relevante, confiable y actualizado. El dominio de la conversación se encuentra explícitamente acotado a esta temática, lo que garantiza un enfoque claro y reduce el riesgo de respuestas incorrectas fuera de este ámbito.

- **Respuestas fundamentadas en información curada y con citas a las fuentes utilizadas** El chatbot genera sus respuestas a partir de documentos previamente seleccionados por su calidad, relevancia y acceso abierto. Se asegura así que el contenido ofrecido sea riguroso y confiable. Además, cada respuesta incluye referencias a las fuentes utilizadas, respetando en que todas tienen acceso abierto, permitiendo así al usuario verificar la información y profundizar en los contenidos. Esta estrategia contribuye a evitar alucinaciones y fortalecer la confianza en el sistema.
- **Comportamiento pedagógico** Más allá de entregar información, el chatbot adopta técnicas de enseñanza activa: formula repreguntas para fomentar la reflexión, ajusta la complejidad de cada explicación acorde al perfil del usuario y presenta la información de forma incremental, evitando la sobrecarga cognitiva. De este modo, funciona como un tutor interactivo que promueve un aprendizaje más auto-guiado y que supera la rigidez de los diálogos tipo “pregunta-respuesta” convencionales. Para lograr esta característica se trabajó con dos asesoras, las profesoras Carolina Rodríguez, Dra. en Ciencias de la Educación (PhD), y Soledad Caballero, coordinadora del grupo de investigación “Inteligencia Artificial para la Investigación Social” de la Facultad de Información y Comunicación. A lo largo del informe, ambas serán referidas como *usuarias expertas* por su rol en el proceso de diseño y validación del sistema.
- **Licencia y extensibilidad** Al liberarse bajo la licencia abierta MIT, el proyecto invita a la comunidad educativa y técnica a adaptar el chatbot a nuevos idiomas, dominios o repositorios específicos, así como también a ampliar su funcionalidad, introducir mejoras y desarrollar nuevas estrategias didácticas o características según las necesidades de cada contexto. Esta apertura no solo multiplica sus posibilidades de uso, sino que también fomenta un ecosistema colaborativo. El código fuente completo se encuentra disponible en el siguiente repositorio público: <https://github.com/Fedee120/ProyGrad2024>.

Estas características hacen que el chatbot atienda directamente las necesidades identificadas: reduce la brecha de conocimiento ofreciendo explicaciones fundamentadas y verificables, y mejora el acompañamiento pedagógico mediante interacciones que promueven la reflexión activa y se apoyan en principios didácticos orientados a favorecer un aprendizaje efectivo y significativo.

El prototipo del chatbot desarrollado puede ser accedido con previa solicitud de credenciales en la dirección: <http://179.27.96.162:8081>. Este requisito tiene como objetivo controlar el acceso y asegurar un uso responsable del sistema, evitando que usuarios no autorizados abusen de los servicios que implican un costo, así como proteger su estabilidad, seguridad y recursos frente a posibles ataques o sobrecargas.

Otro resultado importante que se desprende de este proyecto es la guía para la modificación del corpus o personalización del dominio específico sobre el cual

actúa el chatbot. Esta guía permite adaptar el chatbot a cualquier otro repositorio curado que el cuerpo docente quiera utilizar. Puede encontrarse junto al código fuente, en el archivo `README.md` del repositorio público especificado anteriormente.

1.3. Organización del documento

El resto del documento se organiza de la siguiente manera:

Capítulo 2: Conceptos Preliminares — Presenta los principales conceptos necesarios para comprender la solución desarrollada y las decisiones de diseño tomadas. Se abordan temas como los Grandes Modelos de Lenguaje, métodos de personalización, y el desarrollo de sistemas informáticos basados en este tipo de modelos.

Capítulo 3: Chatbot para la Educación — Detalla los requerimientos, la revisión de antecedentes y la solución propuesta, abarcando tanto el diseño y arquitectura como la implementación.

Capítulo 4: Evaluación y Resultados — Describe las metodologías de prueba (*offline*, *online* y con usuarios) y presenta los resultados obtenidos.

Capítulo 5: Experimentación — Documenta las distintas iteraciones y aprendizajes del proceso de diseño e implementación.

Capítulo 6: Conclusiones y Trabajo Futuro — Recoge las conclusiones principales, las limitaciones identificadas y las líneas abiertas para desarrollos posteriores.

Capítulo 2

Conceptos Preliminares

Este capítulo presenta los conceptos que sustentan este proyecto. En primer lugar, se abordan algunos aspectos fundamentales en relación a los Grandes Modelos de Lenguaje (LLMs, por sus siglas en inglés: *Large Language Models*); posteriormente se examinan estrategias de personalización de modelos; y, por último, se presentan consideraciones sobre su integración en sistemas informáticos. La comprensión de estos conceptos permite fundamentar las decisiones de diseño y del desarrollo de la solución propuesta, y son esenciales para una mejor comprensión de este informe.

2.1. Grandes Modelos de Lenguaje (LLMs)

Esta sección resume los aspectos esenciales de la investigación desarrollada sobre LLMs, cuyo detalle completo se encuentra en el Anexo A. Se incluyen únicamente los puntos imprescindibles para comprender y contextualizar la solución propuesta, manteniendo el enfoque en los objetivos del proyecto.

En esencia, un **LLM** (*Large Language Model*) es una red neuronal diseñada para predecir la siguiente palabra¹ de un texto. Esta capacidad, adquirida tras entrenar el modelo con enormes cantidades de texto, le permite captar patrones gramaticales, semánticos e incluso ciertos aspectos del razonamiento. El punto de inflexión llegó en 2017 con la publicación del artículo “*Attention Is All You Need*” (Vaswani y cols., 2017), que introdujo la arquitectura *Transformer*, base de la mayoría de los LLMs actuales.

El proceso de desarrollo de un LLM consta de dos etapas principales. Primero, un proceso de *pre-entrenamiento* masivo recorre enormes volúmenes de texto utilizando infraestructuras de GPUs/TPUs, dando lugar a un modelo generalista. Posteriormente, una etapa de *fine-tuning* más acotada ajusta los parámetros del modelo para adaptarlo a tareas específicas, como actuar como asistente conversacional, seguir instrucciones y adoptar un tono personalizado.

¹Se predicen *tokens* que son partes de palabras, palabras enteras o conjunto de símbolos, pero se eligió explicar el concepto como palabra para simplificar la lectura.

Una vez entrenado, durante la fase de *inferencia*, el modelo genera texto a partir de una entrada dada. Durante esta etapa, opera dentro de una **ventana de contexto**, que determina cuánta información puede procesar para producir su respuesta. Esta ventana, medida en *tokens*, puede variar entre versiones (por ejemplo, 8k, 32k o 128k) y constituye un límite clave: todo lo que quede fuera de este rango no es visible para el modelo. Por lo tanto, su capacidad de leer o razonar sobre documentos extensos depende directamente de este límite.

Desde el punto de vista de su escala o tamaño, los modelos se agrupan en tres rangos según la cantidad de parámetros que poseen: pequeños, medianos y grandes. Esto determina tanto sus requisitos de cómputo (para entrenamiento e inferencia) como su capacidad de razonamiento y cobertura lingüística. Para evaluar y comparar las capacidades de los modelos, se utilizan benchmarks estandarizados que permiten identificar mejoras reales entre distintas versiones. En el caso de este proyecto de grado, y considerando el estado de los modelos disponibles al momento de la investigación, se concluyó que es necesario utilizar una instancia de gran tamaño para satisfacer los requerimientos definidos, las cuales únicamente están disponibles a través de proveedores en la nube.

Por último, un concepto fundamental en el uso de los LLMs es el de los ***embeddings***: representaciones vectoriales densas que los modelos generan para codificar palabras, frases o documentos. Estas codificaciones preservan relaciones semánticas y se utilizan ampliamente para comparar o recuperar información textual de forma eficiente, cumpliendo un rol central en arquitecturas aumentadas con recuperación, como se describirá más adelante en la sección dedicada a *RAGs*.

2.2. Personalización de LLMs

Los modelos de lenguaje preentrenados, como GPT, LLama, Claude o Gemini, vienen con cierto conocimiento por la gran cantidad de texto empleado en su pre-entrenamiento, y exhiben un comportamiento determinado por la etapa de post-entrenamiento al cual son sometidos, que los alinea con el rol de asistentes capaces de seguir instrucciones y generar respuestas detalladas. Sin embargo, este conocimiento y comportamiento muchas veces no se ajustan a los requerimientos específicos de ciertos casos de uso, como el desarrollo de un chatbot que debe responder únicamente en base a información curada y mantener un estilo conversacional particular.

Por esto, se vuelve necesario utilizar técnicas de personalización que permitan adaptar tanto el conocimiento como el comportamiento del modelo a las necesidades particulares del sistema en desarrollo. La idea central es aprovechar el conocimiento y las capacidades ya adquiridas por el modelo como punto de partida, en lugar de entrenar uno desde cero. Esta estrategia no solo es más eficiente en términos de tiempo y recursos, sino también más viable en contextos donde el entrenamiento completo de un modelo de gran escala, con millones o incluso miles de millones de parámetros, resulta prohibitivo en términos de cómputo y almacenamiento. (IBM, 2024a)

A continuación, se describen distintos métodos para personalizar tanto el **conocimiento** como el **comportamiento** de los LLMs.

2.2.1. Métodos de Personalización del Comportamiento

Existen diversas técnicas para personalizar el comportamiento de un LLM, y la elección de una u otra dependerá de los objetivos y del contexto de uso. En esta sección se presentan dos de las estrategias más utilizadas en el desarrollo de sistemas de inteligencia artificial que requieren adaptarse a necesidades específicas.

Instrucciones en el prompt (*Prompt Engineering*)

Un *prompt* es el texto de entrada que se envía a un LLM con el objetivo de obtener una respuesta generada por el modelo. Generalmente incluye cuatro componentes: instrucciones, que indican explícitamente lo que se espera que el modelo haga; un contexto, que proporciona información adicional para enriquecer la respuesta; la pregunta que el modelo debe procesar; y el formato o estilo deseado para la salida. (Databricks Partner Academy, 2024)

La ingeniería de instrucciones (*prompt engineering*) es la práctica de diseñar y refinar *prompts* -en particular las instrucciones- con el fin de optimizar las respuestas generadas por el modelo y adaptarlas a los objetivos del sistema. A través de estos textos de entrada es posible definir el comportamiento esperado del modelo, incluyendo el tono de la respuesta, el formato de salida, el rol asumido, entre otros aspectos. Por ejemplo, se le podría indicar: “Actúa como un docente de secundaria y respondé de forma breve y clara”.

Entre las estrategias más comunes de *prompt engineering* se encuentran el **Few-shot Prompting** (Brown y cols., 2020) y el **Chain of Thought (CoT)** (Wei y cols., 2022). En el caso de *Few-shot Prompting*, el prompt incluye algunos ejemplos de entradas y salidas esperadas, con el fin de guiar al modelo en la generación de respuestas similares. Por ejemplo, se pueden mostrar varios pares de pregunta y respuesta para ilustrar el comportamiento deseado ante diferentes situaciones. En contraste, la técnica de **Zero-shot Prompting** no proporciona ejemplos y se basa únicamente en instrucciones explícitas para guiar el comportamiento.

Por otro lado, la técnica de *chain of thought* consiste en instruir al modelo para que descomponga su razonamiento en una secuencia de pasos intermedios. En su artículo, Wei y colaboradores ((Wei y cols., 2022)) demuestran que esto mejora sus capacidades para resolver tareas complejas, especialmente aquellas que requieren lógica, aritmética o múltiples inferencias. Cada paso expone de forma clara un resultado parcial que sirve de base para los pasos siguientes, hasta llegar a la respuesta final. Por ejemplo, se podría presentar al modelo la siguiente pregunta, junto con la indicación de razonar paso a paso:

La panadería tenía 23 bananas. Si usaron 20 para hacer una torta y compraron 6 más, ¿cuántas bananas tienen ahora? Piensa paso a paso para resolverlo.

La respuesta del modelo en este caso podría ser:

La panadería tenía originalmente 23 bananas. Usaron 20 para hacer la torta. Entonces les quedaron $23 - 20 = 3$. Luego compraron 6 bananas más, así que ahora tienen $3 + 6 = 9$. La respuesta es 9. (Databricks Partner Academy, 2024)

Elección de Estrategias de Decodificación y Muestreo y Ajuste de Parámetros

Las estrategias de decodificación y muestreo son métodos utilizados por los LLMs para seleccionar el siguiente *token* durante la generación de texto, a partir de una distribución de probabilidad condicionada por los *tokens* anteriores. Estas estrategias determinan cómo se exploran las posibles continuaciones y, por lo tanto, influyen directamente en el comportamiento del modelo: definen cuán diversa, precisa o creativa será la respuesta generada.

Ajustando estas estrategias y sus parámetros asociados es posible personalizar el estilo de generación según los objetivos del sistema. Las estrategias más comunes son *Greedy Search*, *Beam Search* y *Sampling* (Gong, 2025). En el anexo A.0.6 se detallan sus características y parámetros configurables.

En particular, el método de *Sampling* es ampliamente utilizado por su eficiencia y por ser una generalización de otras técnicas como *Greedy Search*. Es común ajustar su parámetro *temperatura* para favorecer respuestas más creativas y variadas (con valores altos) o textos más deterministas y precisos (con valores bajos). Por ejemplo, se podría generar una historia ficticia usando una temperatura más alta, mientras que para generar un documento legal se recomienda usar un valor mucho menor (Bullwinkle, Pavan, Farley, y Urban, 2025).

En resumen, la elección de la estrategia de decodificación y el ajuste de sus parámetros permiten adaptar el estilo de generación del modelo a distintos objetivos. Estos elementos pueden ajustarse y refinarse de manera iterativa hasta alcanzar el comportamiento deseado.

2.2.2. Métodos de Personalización del Conocimiento

Al igual que existen diversas técnicas para personalizar el comportamiento de un LLM, también es posible adaptar su conocimiento mediante distintos enfoques. En esta sección se presentan dos de los métodos más utilizados, comenzando por el que implica mayor complejidad y costo computacional, y luego el que ofrece una alternativa más simple y liviana (Databricks Partner Academy, 2024).

Fine-Tuning sobre modelos pre-entrenados

Este método implica reentrenar parcialmente un modelo ya pre-entrenado con un nuevo conjunto de datos más reducido y específico, con el fin de ajustar sus parámetros para un dominio, estilo o tarea particular. (Srivatsa, 2024)

Una de sus aplicaciones más relevantes es la incorporación de conocimiento específico de un dominio. A pesar de que los LLMs se entrenan inicialmente con un corpus masivo de datos, no poseen conocimiento absoluto ni exhaustivo de todos los dominios. Podría ser necesario, entonces, complementar el conocimiento del modelo base, especialmente en entornos como el legal, financiero o médico, donde se emplea un vocabulario técnico muy preciso. También resulta útil para la incorporación de datos propietarios, lo que permite al modelo acceder a información exclusiva y adaptarse a las necesidades particulares de un sistema u organización.

Si bien el enfoque se presenta aquí en la sección de métodos de personalización del conocimiento, es importante destacar que también se utiliza ampliamente para adaptar el comportamiento del modelo. Entre sus aplicaciones se encuentra la personalización de estilo, lo que permite para ajustar el tono de las respuestas del modelo, haciéndolo más formal, coloquial o ajustado a un rol determinado. De forma análoga, en un modelo de generación de imágenes, el fine-tuning puede utilizarse para adaptar el estilo de las ilustraciones generadas. Asimismo, el fine-tuning permite lograr una mayor especialización en tareas concretas, por ejemplo, dentro de los modelos Llama 2 desarrollados por Meta se lanzó una variante ajustada para chatbots –Llama-2-Chat (Meta AI, 2023b)– y una variante especializada en generar código de programación –Code Llama (Meta AI, 2023a)–.

Cabe señalar que, aunque esta técnica es más eficiente que entrenar desde cero, aún requiere miles de ejemplos etiquetados para obtener resultados efectivos. Además, el conocimiento adquirido sigue alojándose en la “memoria” opaca del modelo, sin ofrecer mecanismos claros para rastrear el origen de la información que genera.

Inclusión de información relevante en el contexto

Anteriormente se habló de la inclusión de instrucciones y ejemplos en el prompt de entrada de un modelo de lenguaje como una estrategia para personalizar su comportamiento. De forma análoga, la inclusión de información relevante en el contexto del prompt se ha consolidado como un mecanismo clave para personalizar el conocimiento de un LLM (Databricks Partner Academy, 2024). Este enfoque permite combinar el conocimiento del modelo base con datos externos que refuerzan la exactitud fáctica de las respuestas, reduciendo alucinaciones y facilitando la posibilidad de citar fuentes, sin la necesidad de continuar el entrenamiento del modelo como en el enfoque de *fine-tuning*.

Sin embargo, la capacidad de la ventana de contexto, la cual incluye al prompt, es limitada. Además de los costos crecientes a medida que el número de tokens se incrementa, su tamaño es finito, por lo que no siempre se puede incluir toda la información deseada. Recientemente, se han desarrollado modelos con ventanas de contexto muy amplias, por ejemplo GPT-4o con 128k tokens (Open AI, 2023) o GPT-4.1 con 1M tokens (Open AI, 2024), y se continúa avanzando hacia contextos aún mayores. No obstante, estas soluciones pueden

resultar costosas en términos de cómputo, tiempo de inferencia y tarifas de uso (cuando se emplean APIs), y no siempre garantizan que el modelo atienda adecuadamente toda la información.

En este escenario, la técnica de **Retrieval Augmented Generation (RAG)** se presenta como una forma eficiente de superar las limitaciones de la ventana de contexto. La idea central es combinar un LLM con un sistema externo de recuperación de información, de manera que, ante una consulta, se recupere información relevante y se la incluya en el contexto del prompt, en lugar de incluir toda la base de conocimientos disponible. Para esto, primero se fragmentan los documentos (proceso conocido como *chunking*), luego cada fragmento se convierte en un vector (*embedding*) y tanto el vector como el fragmento original se almacenan en una base de datos vectorial. Luego, ante una consulta, esta también se vectoriza y se comparan los vectores para recuperar los fragmentos más relevantes, utilizando técnicas de búsqueda por similitud. Estos fragmentos se inyectan en el prompt en tiempo de inferencia, lo que permite al LLM responder con información específica y, si se quiere, citar las fuentes correspondientes. De este modo, se “supera” la finitud de la ventana de contexto al incluir únicamente un conjunto específico de datos relevantes, y se reducen los costos asociados al disminuir la cantidad de tokens a incluir en el contexto.

Existen varios desafíos asociados a los modelos de lenguaje que respaldan el uso de RAGs. Algunos de los principales son:

- Presentar información falsa cuando no tienen la respuesta.
- Presentar información desactualizada o genérica cuando el usuario espera una respuesta específica y actual.
- Brindar información proveniente de fuentes no autorizadas o curadas.
- Crear respuestas inexactas debido a una confusión terminológica, en la que diferentes fuentes de entrenamiento utilizan la misma terminología para hablar de cosas diferentes.

Desafortunadamente, la naturaleza de los LLMs, basados en la generación de texto como una secuencia de predicciones, introduce imprevisibilidad en las respuestas. Además, los modelos no incorporan automáticamente información nueva tras su entrenamiento, lo que impone una fecha límite en los conocimientos que tienen. El RAG es un enfoque para resolver algunos de estos desafíos, ya que aumenta significativamente la probabilidad de controlar la salida de texto generada por el modelo al presentarle, durante la inferencia, información relevante proveniente de fuentes autorizadas y previamente seleccionadas. De este modo, se evita depender únicamente de *lo que alguna vez vió* durante su entrenamiento. ([Amazon Web Services, 2023a](#))

A continuación, se describe con mayor detalle el funcionamiento del enfoque RAG, representado en la Figura 2.1.

- En primer lugar, el usuario realiza una consulta. Sin la arquitectura RAG, la respuesta sería generada por el LLM únicamente a partir de lo aprendido durante su entrenamiento y del *prompt* de sistema. En cambio, con el RAG, se introduce una componente de recuperación de información que toma la entrada del usuario y busca información relevante en una fuente externa al modelo, incorporándola como contexto en el *prompt*.
- Para recuperar la información, la consulta del usuario se convierte en una representación vectorial y se compara con los vectores almacenados en la base de datos. De esta manera, se devuelven los fragmentos específicos que son relevantes para la consulta. Es fundamental resaltar que el modelo de *embeddings* utilizado para vectorizar los documentos almacenados en la base de datos tiene que ser el mismo que el empleado para vectorizar las consultas, ya que en caso contrario, los espacios vectoriales no coincidirían y la recuperación sería ineficaz.
- Una vez que se tiene esta información, se la incorpora al *prompt*, aumentando la entrada del modelo de lenguaje con información relevante. De esta manera, el modelo combina el conocimiento recuperado con lo aprendido durante su entrenamiento para producir respuestas más precisas y fundamentadas. (Amazon Web Services, 2023a)

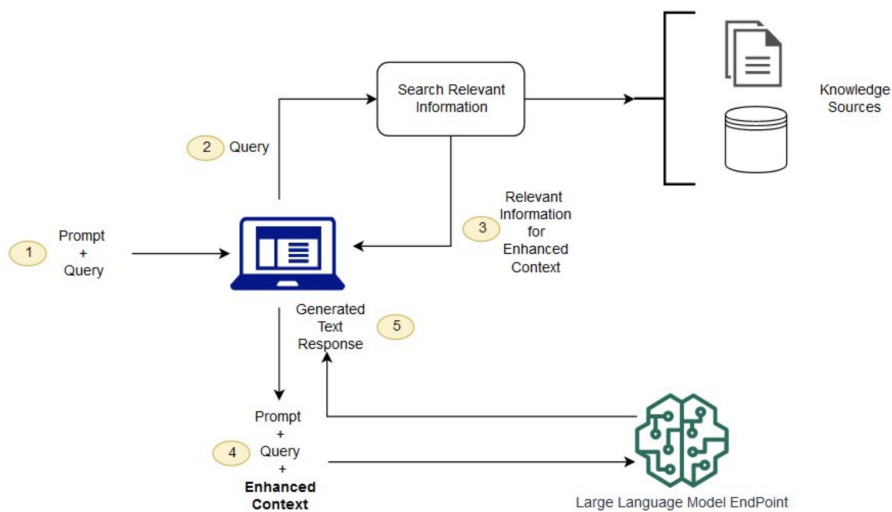


Figura 2.1: Funcionamiento de un RAG. Imagen recuperada del sitio de AWS (Amazon Web Services, 2023a)

2.2.3. Análisis Técnico de los Métodos de Personalización

Luego de haber presentado distintas técnicas para personalizar el comportamiento y conocimiento por defecto de un LLM, en esta sección se realiza un análisis técnico comparativo con el objetivo de identificar sus principales implicancias, ventajas y limitaciones. Este análisis permitirá, más adelante, fundamentar la elección de los métodos más adecuados según los requerimientos del sistema en desarrollo.

Comparativa de Métodos para la Personalización del Comportamiento

Requerimientos computacionales Tanto el *prompt engineering* como la elección de estrategias de decodificación y el ajuste de parámetros como la temperatura son técnicas livianas en términos de requerimientos computacionales, ya que en el primer caso se trata únicamente de construir el texto de entrada, y en el segundo, de configurar valores o seleccionar algoritmos directamente en el código.

Costos asociados La estrategia de decodificación elegida no implica ningún costo adicional. Por su parte, el *prompt engineering* puede llegar a tener un impacto económico si se utilizan *prompts* extensos, ya que los proveedores de modelos suelen cobrar por la cantidad de *tokens* procesados. Por lo tanto, cuanto más largo sea el *prompt* (por ejemplo, al incluir muchas instrucciones o ejemplos), mayor será el costo por consulta. Esta consideración resulta especialmente relevante en sistemas que atienden un gran volumen de consultas y realizan inferencias a gran escala.

Facilidad de implementación Desde el punto de vista de la implementación, las diferencias entre estas técnicas tampoco son significativas. El *prompt engineering* puede aplicarse directamente en la entrada del modelo y la elección de un algoritmo de decodificación y el ajuste de sus parámetros suelen ser fácilmente configurables al instanciar los modelos.

Nivel de personalización que permite Finalmente, estas técnicas presentan diferencias en el grado de personalización que pueden alcanzar. El *prompt engineering* permite establecer roles, tonos, estilos de respuesta y otras indicaciones de comportamiento. No obstante, puede volverse difícil de escalar cuando la cantidad de tareas o indicaciones aumentan en gran medida, ya que cada ajuste en el *prompt* requiere validaciones cuidadosas y puede afectar el rendimiento en otras tareas. Una posible solución a esta limitación es adoptar una arquitectura con múltiples LLMs especializados, ésta visión se abordará en la sección [2.3.4](#). Por su parte, el ajuste de parámetros de decodificación permite modificar aspectos generales del estilo de generación, por ejemplo haciéndolo más creativo o más preciso, pero no influye en otros aspectos del comportamiento del modelo.

Comparativa de Métodos para la Personalización del Conocimiento

Requerimientos computacionales El *fine-tuning* implica reentrenar parcial o completamente un modelo pre-entrenado, lo que puede conllevar costos computacionales y financieros considerables, especialmente si el modelo es de gran tamaño. En cambio, el enfoque basado en RAG no requiere reentrenar el modelo, lo que reduce significativamente los costos iniciales. No obstante, puede demandar mayor capacidad de cómputo durante la operación diaria, particularmente en la fase de recuperación, si se exploran grandes bases de datos.

Velocidad y latencia Un modelo ajustado mediante *fine-tuning* suele tener latencias más bajas en la etapa de inferencia, ya que toda la información se encuentra embebida en sus pesos. En cambio, el RAG introduce un proceso adicional de recuperación de datos antes de la generación, lo que puede incrementar la latencia. Sin embargo, el grado de esta penalización depende de varios factores, como la arquitectura del sistema, el tamaño del corpus de documentos y, en particular, la configuración del motor de recuperación. La estrategia de indexación utilizada, el algoritmo de búsqueda elegido y la parametrización del mismo (por ejemplo, el número de vecinos más cercanos a recuperar o el tipo de métrica de similitud) pueden tener un impacto significativo en los tiempos de respuesta, y ofrecen oportunidades de optimización para mantener la latencia dentro de márgenes aceptables.

Alucinación y confianza de los usuarios En el contexto de la inteligencia artificial, una alucinación es una respuesta generada por la IA que contiene información falsa o engañosa presentada como un hecho. En ocasiones, el modelo puede inventar datos o generar narrativas que, aunque suenen convincentes, carecen de base en la realidad. Si bien cualquier LLM puede “alucinar”, el RAG reduce significativamente esta probabilidad al proporcionarle al modelo información de fuentes externas. Además, este enfoque permite que el modelo cite y referencie documentos específicos, incrementando la confianza del usuario al ofrecer la posibilidad de verificar la información. En cambio, un modelo ajustado mediante *fine-tuning* sigue siendo propenso a alucinaciones como cualquier modelo de lenguaje por su naturaleza predictiva, sin una forma directa de atribuir las fuentes de la información. (Srivatsa, 2024).

Dinamismo y actualización de los datos El RAG sobresale en entornos donde la información evoluciona con frecuencia, ya que permite actualizar la base de datos sin necesidad de reentrenar el modelo. Esta actualización podría realizarse incluso de forma automatizada, o mediante la conexión directa con fuentes de información dinámicas como redes sociales, sitios de noticias, etc. Esto facilita la integración de nuevos documentos o la sustitución de fuentes obsoletas, pudiendo así proporcionar la información más reciente a los usuarios (Amazon Web Services, 2023a). Por su parte, el *fine-tuning* es más apropiado cuando el dominio o conjunto de datos es estable y no precisa de actualizaciones periódicas, debido a los costos y tiempos que conlleva.

La elección de una estrategia de personalización depende de múltiples factores, como el estilo de respuesta deseado, las restricciones sobre la información que puede utilizarse, la necesidad de incluir fuentes, los recursos disponibles, los requisitos técnicos de cada enfoque, la viabilidad de implementación y los objetivos generales del sistema. En definitiva, no hay una única opción correcta, sino que la estrategia más adecuada dependerá del contexto específico del sistema a desarrollar.

2.3. Integración de modelos en un sistema informático

Además de definir cómo personalizar los modelos para ajustarlos a los objetivos del sistema, integrar LLMs en aplicaciones reales requiere tomar otras decisiones técnicas clave. Estas incluyen desde la elección de la arquitectura del sistema y la infraestructura necesaria, hasta la forma en que se interactuará con el modelo.

En esta sección se examinan aspectos centrales a considerar al integrar LLMs en aplicaciones de software: los entornos de ejecución posibles (local o en la nube), los formatos de respuesta que puede adoptar el modelo, los *frameworks* disponibles, algunas configuraciones arquitectónicas comunes y los nuevos enfoques de evaluación que surgen a partir del uso de este tipo de sistemas.

2.3.1. Entorno de ejecución

Existen dos aproximaciones principales para la ejecución de LLMs en entornos productivos. La primera consiste en la ejecución local, que requiere de una infraestructura de cómputo altamente especializada, por lo general con unidades de procesamiento gráfico (GPUs) de alto desempeño. Esta opción brinda un control completo sobre la infraestructura y, en particular, sobre los datos que se envían al modelo: se puede definir con precisión qué información accede el LLM, cómo se procesa y si se almacena o no. Esta característica resulta clave en contextos que manejan datos sensibles o requieren políticas estrictas de privacidad. Sin embargo, se trata de un enfoque con un costo inicial elevado debido al hardware requerido, además de la necesidad de conocimientos técnicos para mantener y optimizar el modelo en servidores propios.

La segunda aproximación consiste en el uso de *APIs* que se ofrecen desde la nube, las cuales permiten acceder a modelos alojados por terceros sin necesidad de infraestructura propia. Este enfoque facilita la escalabilidad (por no depender de la capacidad de un hardware local) y reduce la inversión inicial, aunque puede generar costos por consulta. Además, plantea desafíos en términos de gobernanza y privacidad de datos, ya que la información enviada al modelo es procesada por servidores externos, lo cual puede implicar riesgos si no se cuenta con garantías claras sobre el almacenamiento y uso de esos datos por parte del proveedor. Por este motivo, es relevante evaluar cuidadosamente los términos

de servicio y las políticas de seguridad del proveedor antes de optar por este enfoque (Dave Club, 2025).

2.3.2. Formato de respuesta del modelo

Los LLMs están diseñados principalmente para procesar y generar texto. Sin embargo, en aplicaciones complejas, es común requerir que la salida del modelo adopte un formato estructurado o incluya acciones específicas que puedan ser interpretadas automáticamente por otros componentes del sistema. Para lograrlo, se han desarrollado mecanismos que permiten controlar el formato de salida del modelo. Esto puede lograrse mediante *prompting* o *fine-tuning*, aunque en la práctica también se utilizan técnicas más avanzadas como enforzado mediante expresiones regulares o gramáticas libres de contexto (NVIDIA, 2024). Entre los enfoques más destacados se encuentran *tool calling* y *structured output*, que se describen a continuación.

Tool Calling En este mecanismo, el modelo no solo produce texto libre, sino que también puede indicar explícitamente la invocación de ciertas funciones, incluyendo su nombre y los parámetros correspondientes. Las funciones disponibles, también denominadas herramientas (*tools*), se definen previamente, junto con una descripción que guía al LLM para decidir cuándo invocarlas. De esta forma, el modelo puede devolver una salida estructurada –por ejemplo un bloque JSON con el nombre de la función y sus parámetros– que la aplicación interpreta para ejecutar la acción correspondiente (LangChain, Inc., 2024b).

Por ejemplo, un asistente virtual diseñado para ayudar con diversas tareas, además de responder con una frase como “Claro, crearé un evento para mañana a las 10 a.m.”, puede devolver un bloque estructurado como el siguiente si está configurado para funcionar con *tool calling* y tiene definida una función para agendar reuniones:

```
{
  "tool_name": "crear_evento_calendario",
  "tool_args": {
    "fecha": "2025-03-15",
    "hora_inicio": "10:00",
    "hora_fin": "11:00",
    "descripcion": "Reunión semanal de seguimiento"
  }
}
```

Luego, el sistema puede reconocer fácilmente este tipo de estructuras para invocar la función correspondiente y automatizar la creación del evento, lo cual resulta mucho más sencillo que interpretar texto en lenguaje natural.

Como se puede observar, este enfoque es útil cuando se requiere que el modelo no solo interactúe con el usuario, sino que también interactúe con un sistema, ejecutando funciones en base al contenido de la consulta.

Structured Output Este enfoque, conocido como *Structured Output* o *Salida Estructurada*, también es útil cuando se requiere que la salida del modelo tenga un formato que pueda procesarse de forma determinista. A diferencia del enfoque anterior, en este caso no se busca ejecutar funciones condicionalmente, sino garantizar que la salida del modelo mantenga siempre una estructura predefinida y consistente. Para ello, el LLM es “instruido” para generar respuestas en bloques JSON que sigan un esquema específico, por ejemplo, con claves fijas o valores tipados (LangChain, Inc., 2024a).

Por ejemplo, si se requiere que el modelo analice opiniones de usuarios acerca de un chatbot y registre únicamente su calificación y un comentario breve, podría generar:

```
{
  "calificacion": 5,
  "comentario": "El chatbot resolvió mis dudas de manera rápida."
}
```

De esta manera, la aplicación puede *parsear* automáticamente la información para procesarla o almacenarla en una base de datos.

En ambos casos, el principal beneficio radica en que la respuesta del modelo deja de ser texto libre y pasa a tener una estructura definida que la aplicación puede interpretar de forma automática. De esta manera, se simplifica la tarea de convertir la salida del modelo en acciones o datos utilizables por otros componentes del sistema.

2.3.3. Frameworks

Un *framework* o marco de trabajo es una colección de componentes de software reutilizables que hacen que sea más eficiente desarrollar nuevas aplicaciones (Amazon Web Services, s.f.). Proporcionan un punto de partida para el desarrollo, permitiendo ahorrar tiempo y esfuerzo al ofrecer funcionalidades ya implementadas y herramientas listas para usar.

En el contexto de la IA generativa, han surgido diversas opciones que permiten integrar modelos, gestionar flujos de datos y conectar con herramientas externas, entre otras tareas. En general estos *frameworks* proveen:

- Mecanismos de integración de modelos que permiten utilizar distintos proveedores, alternar entre ejecución local y en la nube, y ajustar parámetros de configuración, todo sin modificar sustancialmente la lógica de la aplicación.
- Estructuras predefinidas para la interacción con modelos, que incluyen abstracciones comunes como *prompts* y mensajes (de usuario, asistente, sistema, etc.), y también otras como *tools*, *retrievers*, *agents*, entre otros, que permiten estructurar flujos más complejos.

- Herramientas para orquestar flujos de trabajo, facilitando la composición de múltiples pasos de procesamiento.
- Capacidad de integración con herramientas externas como bases de datos, APIs u otros servicios.
- Soporte para funciones avanzadas como *tool calling* y salidas estructuradas, así como técnicas complementarias como *metadata filtering*, *re-ranking*, *chain of thought*, entre otras.
- Herramientas para evaluar el desempeño de los sistemas basados en LLMs, ya sea a través de métricas tradicionales o entornos especializados que permiten monitoreo y trazabilidad.

Si bien existe una gran variedad de frameworks disponibles como LlamaIndex, Haystack o Rasa, se analizará uno de los más populares en el ámbito de la IA generativa: **LangChain**, debido a su amplia adopción, la calidad de su documentación y la solidez de su comunidad de desarrollo. En particular, se considerarán criterios clave como la disponibilidad de una variedad de LLMs, modelos de embeddings, bases de datos vectoriales, así como mecanismos de evaluación y trazabilidad, y soporte para integrar técnicas avanzadas en aplicaciones basadas en LLMs.

LLMs compatibles Puesto que los modelos de lenguaje constituyen la base de los sistemas de IA generativa, es fundamental que el *framework* permita integrar modelos cuyas capacidades se ajusten a los requerimientos de la aplicación. Esto puede incluir un buen manejo del idioma, habilidades sólidas de razonamiento y la capacidad de responder a preguntas complejas de manera precisa, entre otros aspectos. LangChain es compatible con una gran cantidad de APIs como OpenAI, Anthropic, Hugging Face o Groq, lo que permite acceder a modelos tanto de código abierto como propietarios. Hugging Face, por ejemplo, aloja más de 300.000 modelos accesibles a través de LangChain. Además, el *framework* admite el uso de modelos que se ejecutan localmente, como los de Ollama (LangChain, Inc, 2024b).

Modelos de embeddings compatibles En caso de utilizar el enfoque RAG, los modelos de *embeddings* juegan un rol central en la recuperación de información, por lo que también es importante considerar cuáles pueden integrarse a través del *framework*. Al igual que con los LLMs, LangChain ofrece compatibilidad con una amplia variedad de proveedores de modelos de *embeddings*, lo que permite seleccionar el más adecuado para las necesidades específicas del proyecto (LangChain, Inc, 2024a).

Bases de datos vectoriales compatibles Las bases de datos vectoriales son clave en aplicaciones como RAG y otros sistemas que requieren recuperación eficiente de información mediante búsquedas por similitud. Por eso, es fundamental que el *framework* soporte diversas bases vectoriales, para elegir la que

mejor se adapte al sistema en desarrollo. Langchain proporciona acceso a más de 40 bases de datos vectoriales populares, como Chroma, Pinecone, Milvus, Weaviate, Faiss, entre otras. Estas tienen distintas características: algunas son gratuitas, otras de pago; algunas de código abierto, otras propietarias. Además, pueden ejecutarse localmente (*self-hosting*), en servidores remotos (en la nube) o en contenedores Docker, ofreciendo flexibilidad para ajustarse a distintos proyectos ([LangChain, Inc, 2025e](#)).

Trazabilidad y monitoreo de ejecuciones A medida que las aplicaciones basadas en LLMs aumentan en complejidad, se vuelve esencial contar con mecanismos que permitan registrar y analizar su flujo de ejecución. Esto brinda trazabilidad para identificar fallas, cuellos de botella y oportunidades de mejora, así como comparar distintas configuraciones en términos de precisión, latencia o costo. LangChain ofrece integración con la plataforma **LangSmith**, enfocada en la evaluación, el monitoreo y la trazabilidad (*tracing*) de aplicaciones de IA. Esta plataforma permite almacenar y visualizar las ejecuciones, incluyendo cada paso involucrado (llamadas al modelo, ejecución de funciones, comunicación con herramientas externas, etc.), lo que facilita la verificación del comportamiento esperado y el análisis del rendimiento del sistema. Además, posibilita el análisis comparativo entre ejecuciones, facilitando la iteración sobre diferentes *prompts*, modelos o parámetros, y la evaluación de métricas como el tiempo de respuesta o el costo de cómputo ([LangChain, Inc, 2025a](#)).

Soporte para técnicas avanzadas LangChain ofrece soporte para integrar una gran variedad de técnicas avanzadas que permiten mejorar la capacidad de razonamiento de los modelos y la relevancia de las respuestas, así como también adaptarse a las necesidades específicas de cada proyecto. De esta manera, permite construir flujos de trabajo complejos y adaptativos, personalizados según el caso de uso.

En resumen, LangChain se presenta como un *framework* completo que constituye una opción sólida y confiable para el desarrollo y mantenimiento de aplicaciones basadas en IA generativa. Para profundizar en sus características y comprender mejor las razones que lo hacen destacar, puede consultarse su documentación oficial ([LangChain, Inc, 2025f](#)).

2.3.4. Arquitecturas

El desarrollo de aplicaciones basadas en IA generativa puede abordarse mediante diferentes arquitecturas, que equilibran la complejidad y la modularidad de formas diversas. Al diseñar una solución, es fundamental evaluar si conviene emplear un único LLM para todas las tareas o distribuir las responsabilidades entre múltiples modelos. Esta decisión va a depender de la cantidad y complejidad de las funcionalidades a implementar, la rapidez de implementación, los requisitos de escalabilidad y la flexibilidad que se quiera tener, entre otros

factores. A continuación, se presentan dos enfoques representativos que ilustran distintas formas de organizar y distribuir las funcionalidades dentro del sistema.

LLM único

En este primer enfoque, un único modelo asume todas las funcionalidades y responsabilidades a través de un *prompt* extenso. De esta forma, el LLM puede, en una sola invocación, combinar tareas como la clasificación de texto y la generación de resúmenes. Este enfoque también es conocido como *Standard Prompting*, ya que se basa en un único *prompt* para guiar la respuesta del modelo, sin estructuras adicionales ni procesos de interacción complejos. (Khot y cols., 2023)

El principal beneficio de una arquitectura con un único LLM es su simplicidad, tanto en diseño como en implementación. Al no requerir múltiples modelos ni un sistema de orquestación, permite reducir la latencia y los costos de cómputo al minimizar la cantidad de llamadas a distintos componentes. Por ejemplo, si se requiere clasificar y resumir reseñas en un solo paso, basta una única invocación al modelo, evitando la sobrecarga de coordinación entre distintos módulos.

Sin embargo, este enfoque también presenta desventajas. A medida que la complejidad de las tareas y la cantidad de instrucciones a procesar aumentan, la efectividad del modelo puede verse afectada, ya que debe procesar múltiples requerimientos en un solo paso. Asimismo, la trazabilidad de los resultados se dificulta, pues todas las decisiones del modelo quedan implícitas en sus respuestas, sin una secuencia clara de entradas y salidas que permita seguir el flujo de las interacciones entre componentes. En escenarios con requerimientos acotados o en fases iniciales de un proyecto, esta arquitectura puede ser suficiente e incluso deseable, pero a medida que el sistema crece en complejidad, suelen volverse evidentes sus limitaciones en términos de mantenibilidad y control. (HatchWorksAI, 2024)

Múltiples LLMs

En este segundo enfoque, se utiliza un conjunto de LLMs, cada uno responsable de una tarea específica. La coordinación entre los módulos puede ser gestionada utilizando los mecanismos de orquestación provistos por el *framework* elegido.

En el paper de Khot et al., presentado en la Conferencia Internacional sobre Representaciones del Aprendizaje (ICLR 2023) (Khot y cols., 2023), se introduce el concepto de *Decomposed Prompting* como una estrategia para abordar tareas complejas mediante su descomposición en sub-tareas más simples, que se delegan a LLMs especializados con *prompts* optimizados. Esta estructura modular permite que cada *prompt* sea afinado y especializado para su tarea específica, y, de ser necesario, cada sub-tarea pueda ser subdividida aún más.

Este tipo de arquitecturas con múltiples LLMs promueven la escalabilidad, al permitir agregar y modificar componentes sin impactar al resto del sistema. Cada *prompt* o componente puede ser fácilmente reemplazado por una versión

mejorada, un modelo entrenado o incluso por una función, según sea necesario. Además, facilita la mantenibilidad al posibilitar pruebas y mejoras de cada módulo por separado, y permite la reutilización de los componentes encargados de cada sub-tarea en diferentes contextos.

Otro beneficio clave es el aislamiento de comportamientos, ya que se segmentan claramente las responsabilidades entre los distintos módulos. En contraposición, cuando se emplea un único *prompt* demasiado extenso, cualquier modificación puede afectar todos los comportamientos en simultáneo, además de favorecer contradicciones en las indicaciones, por ejemplo, exigir un estilo conciso pero también desarrollar en profundidad sobre cierto tema. Dichas inconsistencias pueden degradar la coherencia de las respuestas generadas, mientras que la arquitectura modular atenúa este riesgo al distribuir las directrices y la lógica entre componentes especializados.

El trabajo de Khot et al. demuestra cómo *Decomposed Prompting* supera las limitaciones del enfoque tradicional de *Standard Prompting*. Evaluaron esta técnica sobre ocho conjuntos de datos desafiantes, comparando su rendimiento con trabajos previos realizados bajo la arquitectura de *Standard Prompting*, obteniendo resultados superiores.

Por último, esta arquitectura también permite una mayor personalización del comportamiento. Es posible adaptar el tono, la estructura y el enfoque de las respuestas de cada LLM según la tarea, ajustándolos a las necesidades específicas del sistema.

Cada enfoque responde a necesidades distintas. Para aplicaciones con requerimientos muy acotados o en fases iniciales, la aproximación con un único LLM puede ser suficiente, puesto que permite prototipar de manera rápida. Sin embargo, en escenarios más complejos o con proyección de crecimiento, una arquitectura modular ofrece mayor robustez y adaptabilidad.

Cabe mencionar que las arquitecturas presentadas constituyen solo dos grandes enfoques, y que existen muchas otras alternativas. Por ejemplo, algunas emplean un único modelo que ejecuta múltiples pasos de forma iterativa, refinando su respuesta a medida que avanza. Este enfoque permite invocar varias herramientas en secuencia mediante *tool calling*, dejando que el modelo decida cuándo y cómo utilizar cada herramienta. Por ejemplo, el modelo podría llamar a una función para realizar un cálculo, luego a otra para recuperar información adicional y, finalmente, integrar todo en una respuesta coherente (LangChain, Inc., 2024).

Así como este, existen muchos otros enfoques posibles, e incluso diversas variaciones dentro de las arquitecturas presentadas, lo cual permite adaptarse a distintos escenarios y objetivos.

2.3.5. Evaluación

En esta sección se profundiza en los mecanismos de evaluación de los sistemas basados en IA generativa, presentando tanto un panorama general de los métodos utilizados históricamente, así como contribuciones recientes en el área.

Reseña histórica de evaluación de sistemas

Una descripción más detallada de los métodos de evaluación aplicados al PLN y a los LLMs, desde las métricas de coincidencia exacta hasta las evaluaciones basadas en similitud semántica, se presenta en el anexo D. Aquí basta señalar que la evaluación ha transitado desde enfoques léxicos tradicionales, como *Exact Match* y *F1 Score*, hacia métricas que explotan modelos de lenguaje y muestran una fuerte correlación con el juicio humano. Este último enfoque se desarrolla a continuación.

LLM as a Judge

En los últimos años se ha observado un creciente interés en utilizar modelos de lenguaje grandes no solo para generar respuestas, sino también como evaluadores automáticos que se aproximen a la valoración humana. El enfoque denominado *LLM-as-a-Judge* (Zheng y cols., 2023) consiste en emplear un LLM robusto (por ejemplo, GPT-4) para juzgar y calificar las respuestas producidas por distintos sistemas de diálogo o de preguntas y respuestas. Cuando la evaluación es realizada por un LLM, el resultado se obtiene a partir de un *prompt* que le indica cómo razonar y qué aspectos debe valorar. En su artículo presentado en la 61ª Conferencia Anual de la ACL (*Association for Computational Linguistics*), Zheng y cols. exponen tres estrategias de evaluación que pueden implementarse de forma independiente o combinada.

En el primer esquema llamado **comparación par a par**, se le presentan al LLM dos respuestas a la misma pregunta y se le solicita que determine cuál de ellas es superior. La decisión se basa en criterios como la relevancia, precisión, profundidad y coherencia con la consulta original. Este método resulta especialmente útil para identificar diferencias sutiles en la calidad de las respuestas generadas por dos modelos distintos.

En otro de los esquemas, llamado **calificación de respuesta única**, el juez (el LLM) asigna una puntuación a una respuesta concreta siguiendo una rúbrica preestablecida. Aunque puede no captar matices finos en la comparación directa entre respuestas, este enfoque es altamente escalable y permite obtener evaluaciones cuantitativas de forma rápida.

Finalmente, el esquema de **calificación guiada por referencia** se puede utilizar en tareas donde se dispone de una respuesta canónica, permitiendo comparar las respuestas generadas por dos modelos en función de su alineación con dicha referencia. Este enfoque facilita la identificación de errores o desviaciones en las respuestas y se basa en la similitud semántica, permitiendo valorar respuestas que, aunque no coincidan exactamente con la respuesta de referencia, mantengan coherencia y relevancia con la información esperada.

Entre las principales ventajas de emplear un LLM como juez se destacan la escalabilidad y la capacidad de ofrecer explicaciones detalladas sobre sus decisiones. Además de asignar puntajes, estos modelos generan justificaciones que hacen su salida más interpretable. Este enfoque permite reducir la dependencia de evaluaciones humanas, que suelen ser costosas y lentas, y al mismo tiempo

aporta mayor transparencia al proceso evaluativo. Los experimentos presentados en el estudio de Zheng y cols. muestran que, al utilizar GPT-4 como juez, se alcanza una tasa de acuerdo con evaluadores humanos superior al 80 %, equivarable a la concordancia observada entre evaluadores humanos.

Sin embargo, este enfoque también presenta ciertas limitaciones. Se han identificado sesgos, como el **sesgo de posición** (tendencia a favorecer la respuesta que se presenta primero), el **sesgo de verbosidad** (preferencia por respuestas excesivamente largas) y el **sesgo de auto-mejora** (tendencia a favorecer respuestas generadas por el propio modelo juez). Además, se ha observado que los LLMs presentan dificultades para evaluar preguntas que requieren razonamiento lógico o habilidades matemáticas. Incluso en problemas matemáticos simples —que el propio modelo podría resolver correctamente— pueden fallar al juzgar respuestas ajenas. Para mitigar estos problemas, se proponen estrategias como el intercambio de posiciones al presentar las respuestas y el uso de técnicas de *few-shot prompting* para orientar la evaluación del modelo.

Métricas comunes en la evaluación de sistemas RAG

Para evaluar los sistemas basados en Generación Aumentada por Recuperación (RAG), se han desarrollado métricas específicas que consideran tanto la calidad del contexto recuperado como la de las respuestas generadas. Estas métricas se centran en aspectos clave como la relevancia del contexto y la respuesta, la cobertura de la información necesaria y la fidelidad con respecto a las fuentes utilizadas. Si bien algunas pueden calcularse mediante métodos automáticos tradicionales, el enfoque *LLM-as-a-Judge* representa una alternativa versátil y automatizada, especialmente útil en escenarios donde no se tienen verdades de referencia. Además permite una evaluación más integral y contextual del proceso de recuperación (Yu y cols., 2025).

Estas métricas han sido presentadas y discutidas en trabajos recientes como la revisión de *benchmarks* de evaluación para sistemas RAG de (Yu y cols., 2025), la propuesta de RAGAs de (Es, James, Espinosa Anke, y Schockaert, 2024), y los recursos prácticos ofrecidos por Databricks (Databricks Partner Academy, 2024). A continuación, se describen algunas de las métricas más utilizadas.

Answer Correctness: Evalúa si la respuesta generada es factualmente correcta al compararla con una solución de referencia (*gold standard*).

Answer Relevancy: Evalúa qué tan completa y pertinente es la respuesta en relación a la consulta formulada. Es posible que una respuesta sea correcta en términos fácticos pero resulte incompleta si no aborda todos los aspectos relevantes de la pregunta.

Faithfulness: Mide el grado en que la respuesta se basa en la información proporcionada en el contexto, evitando la incorporación de datos inventados o alucinaciones. Este aspecto es especialmente relevante ya que los sistemas RAG suelen utilizarse en dominios y aplicaciones donde la consistencia factual con respecto a fuentes externas verificables resulta crítica (Es y cols., 2024).

Context Recall: Evalúa en qué medida el contexto recuperado incluye todas las entidades e información relevantes presentes en la respuesta de referencia. Un alto valor en esta métrica indica que una gran proporción del contexto recuperado es efectivamente pertinente. Esta métrica no solo refleja la efectividad de la recuperación, es decir, si efectivamente se están recuperando aquellos fragmentos con información relevante; sino que también puede servir como indicador de la completitud de la base de documentos curados.

Context Relevancy: Esta métrica evalúa si el contexto recuperado está compuesto por fragmentos precisos y focalizados para responder la pregunta del usuario, minimizando la presencia de datos irrelevantes. Alimentar al LLM con contextos demasiado extensos y poco enfocados puede incrementar el costo computacional y degradar la calidad de las respuestas, especialmente cuando la información útil se encuentra en posiciones intermedias dentro de los fragmentos recuperados (Liu y cols., 2024), (Es y cols., 2024).

En síntesis, el enfoque *LLM-as-a-Judge* representa una aproximación robusta y alineada con la valoración humana, que supera varias de las limitaciones de los métodos de evaluación tradicionales. Gracias a su flexibilidad, puede utilizarse no solo para evaluar métricas comunes en sistemas RAG, sino también para valorar otros componentes o funcionalidades del sistema en desarrollo. Esto lo convierte en una herramienta especialmente valiosa para guiar el proceso iterativo de desarrollo y mejora de sistemas basados en LLMs.

Capítulo 3

Chatbot para la Educación

Este capítulo, de carácter central en el desarrollo del proyecto, se encarga de describir de forma integral el proceso de construcción del prototipo, abarcando tres ejes fundamentales. En primer lugar, se exponen los *requerimientos del proyecto*, especificando las necesidades y restricciones que condicionan el diseño de la solución. Posteriormente, se incluye una *revisión de antecedentes*, explorando la literatura y los enfoques previos que sirvieron como base conceptual. Finalmente, se describe la *solución planteada*, abarcando tanto la arquitectura y el diseño como la implementación. De esta manera, el capítulo brinda una visión completa que conecta las necesidades identificadas, la investigación realizada y las decisiones de ingeniería tomadas.

3.1. Requerimientos del Proyecto

A continuación se presentan los requerimientos del proyecto, los cuales orientaron tanto la revisión de antecedentes como el diseño e implementación de la solución desarrollada. Estos requerimientos fueron relevados principalmente durante las reuniones iniciales con las tutoras del proyecto y las usuarias expertas, cuya experiencia resultó clave para delimitar el alcance y las prioridades del sistema.

3.1.1. Requerimientos Funcionales

RF-1. Interacción acotada a temas de inteligencia artificial

- Descripción: El chatbot deberá limitar sus respuestas exclusivamente a temas relacionados con la inteligencia artificial. En caso de que el usuario consulte sobre otros temas, el sistema deberá redirigir la conversación al área de la IA, pudiendo incluso sugerir el uso de otras herramientas para tratar el tema no soportado.

- **Importancia:** Esto garantiza que el chatbot se mantenga enfocado en su objetivo principal, reforzando la calidad y fiabilidad de la información al evitar ofrecer respuestas sobre temas para los cuales no está especializado.

RF-2. Interacción basada en fuentes curadas

- **Descripción:** El chatbot deberá integrarse con un repositorio de documentos curados que aborden los distintos temas comprendidos en el alcance temático (por ejemplo: fundamentos de la IA, aprendizaje automático, redes neuronales, ética y privacidad en el uso de la IA, aplicaciones, entre otros). Las respuestas deberán construirse exclusivamente a partir de la información contenida en este repositorio. En caso de no disponer de la información necesaria, el chatbot deberá informarlo al usuario y sugerir recursos alternativos. Además, cada respuesta que utilice información del repositorio deberá incluir la cita completa correspondiente.
- **Importancia:** Este requerimiento garantiza respuestas precisas y fundamentadas en fuentes curadas, lo que refuerza la credibilidad del sistema y su valor educativo. Además, la inclusión de citas completas permite a los usuarios verificar la información y aporta transparencia al proceso de generación de respuestas.

RF-3. Interacción basada en diálogo

- **Descripción:** El chatbot deberá ofrecer una interacción conversacional fluida, evitando respuestas excesivamente largas. Para eso, deberá mantener memoria del contexto de la conversación, permitiendo interpretar correctamente referencias a mensajes anteriores. Además, la información se deberá proporcionar de forma gradual, ampliándose a medida que el usuario lo solicite.
- **Importancia:** Este enfoque favorece un intercambio dinámico y bidireccional similar a una conversación humana. Evita la sobrecarga de información y promueve un aprendizaje adaptativo y centrado en el interés del interlocutor.

RF-4. Interacción basada en estrategias pedagógicas

- **Descripción:** El chatbot deberá incorporar estrategias pedagógicas, como la técnica de repregunta. Esto implica, en ocasiones, postergar la respuesta a la consulta del usuario para formular preguntas que estimulen el razonamiento y la reflexión.
- **Importancia:** Potencia el proceso de aprendizaje, ya que fomenta el pensamiento crítico y la participación activa del usuario.

RF-5. Interacción adaptable al perfil del usuario

- **Descripción:** El sistema deberá ajustar la complejidad, el nivel de detalle y el enfoque de las respuestas en función de la experiencia previa del usuario, determinada a través de un selector simple (por ejemplo: básico, medio y avanzado).

- **Importancia:** Permite personalizar la experiencia de aprendizaje, orientando la generación de respuestas más accesibles o más especializadas, según corresponda.

RF-6. Manejo de los historiales de conversación

- **Descripción:** El sistema deberá almacenar el historial de conversaciones de los usuarios para permitir la visualización, continuación y eliminación de conversaciones anteriores. El chatbot deberá utilizar este historial para evitar la repetición de información y mantener el contexto en las interacciones. Es fundamental informar explícitamente al usuario sobre el almacenamiento de sus conversaciones, aclarando que su propósito es mejorar la experiencia mediante la continuidad del diálogo, y siempre garantizando la confidencialidad de dichos datos.
- **Importancia:** Mejora significativamente la usabilidad del sistema y favorece interacciones contextualizadas, a la vez que cumple los estándares de seguridad y confidencialidad, aumentando así la confianza del usuario.

3.1.2. Requerimientos No Funcionales

RNF-1. Confidencialidad y seguridad de los datos

- **Descripción:** El sistema no deberá almacenar información personal de los usuarios sin su consentimiento explícito, garantizando así la confidencialidad y seguridad de los datos. Además, deberá utilizar herramientas de código abierto siempre que su uso no afecte significativamente la calidad del sistema y sea técnicamente viable.
- **Importancia:** Protege la privacidad del usuario y promueve un entorno seguro, aspectos esenciales en cualquier sistema interactivo. Además, el uso de herramientas de código abierto favorece la transparencia, lo que contribuye a fortalecer la confianza en el sistema.

RNF-2. Protección de la propiedad intelectual

- **Descripción:** Todos los documentos en el repositorio deberán incluir referencias completas (título, autores y año de publicación) y se deberá contar con los permisos y licencias adecuados para su uso. Cada vez que el chatbot utilice información de este repositorio, deberá proporcionar las citas correspondientes.
- **Importancia:** Garantiza el respeto a los derechos de autor y la correcta atribución de las fuentes, evitando problemas legales y promoviendo una gobernanza responsable de los datos.

RNF-3. Escalabilidad

- Descripción: La arquitectura del sistema deberá ser escalable, permitiendo futuras expansiones tanto en funcionalidad como en la incorporación de nuevos módulos o extensiones. Asimismo, el sistema deberá permitir cambio del dominio temático del repositorio de forma sencilla, ofreciendo una guía para llevar a cabo este proceso.
- Importancia: Facilita la adaptación del sistema a nuevos requerimientos o áreas temáticas, prolongando su vida útil y manteniendo su relevancia a lo largo del tiempo.

RNF-4. Mantenibilidad

- Descripción: El código del sistema deberá estar organizado con una clara separación de responsabilidades y contar con una documentación completa, de manera de facilitar su mantenimiento y futuras actualizaciones.
- Importancia: Asegura que el sistema pueda ser modificado o ampliado de forma eficiente, reduciendo los costos y el tiempo asociados a tareas de mantenimiento y mejora.

3.1.3. Análisis Técnico de los Requerimientos

Del análisis de los requerimientos funcionales se desprende la necesidad de incorporar modelos de lenguaje que no solo tengan un buen desempeño conversacional, sino que además cuenten con cierto grado de inteligencia para ser capaces de:

- Limitarse a responder exclusivamente cuestiones vinculadas con la inteligencia artificial (RF-1).
- Generar respuestas únicamente a partir de contenido proporcionado, evitando introducir información no respaldada (RF-2).
- Seguir instrucciones específicas sobre el estilo de respuesta, como presentar la información de manera progresiva, en función de las solicitudes del usuario (RF-3).
- Incorporar estrategias pedagógicas de forma pertinente y contextualizada, promoviendo la reflexión y el razonamiento sin afectar negativamente la experiencia del usuario (RF-4).
- Adaptar el nivel de complejidad de las respuestas según la experiencia declarada del usuario (RF-5).

Además, el modelo deberá ser capaz de decidir en cada interacción si corresponde continuar un diálogo casual, recuperar información del repositorio o aplicar estrategias pedagógicas como la repregunta, en función del contexto de la conversación y la pregunta del usuario.

Considerando la infraestructura de cómputo especializada necesaria para correr localmente modelos de lenguaje de tamaño mediano o grande, con capacidades avanzadas de razonamiento, y el rendimiento limitado observado en los modelos *open source* al momento de realizar la investigación (ver [Comparativa de Desempeño de Modelos de Lenguaje](#)), se concluye que la opción más viable es acceder a modelos mediante un proveedor de servicios.

Por otro lado, el sistema presenta diversas necesidades de personalización. En primer lugar, se requiere personalizar el comportamiento del modelo para que el estilo de sus respuestas se adecúe a los criterios definidos en el requerimiento [RF-3](#), lo cual podría lograrse mediante técnicas de *prompting* según lo expuesto en la sección [2.2.1](#) de métodos de personalización del comportamiento. Además, el requerimiento [RF-5](#) también podría abordarse utilizando esta técnica, incorporando al *prompt* información sobre el nivel de experiencia del usuario y el tipo de salida esperado en cada caso. Asimismo, los requerimientos de restringir las respuestas a temas relacionados con la IA ([RF-1](#)) y aplicar estrategias pedagógicas en determinadas situaciones ([RF-4](#)), pueden ser abordados mediante *prompt engineering*.

Además, cuando se le provee información al modelo, es necesario que sus respuestas se limiten estrictamente a esa información, con el fin de asegurar la fiabilidad de las respuestas ([RF-2](#)). Para ello, parece adecuado ajustar el parámetro *temperatura* a un valor bajo, lo que reduce la creatividad del modelo y favorece respuestas más precisas. En cambio, en situaciones donde el sistema debe mantener una conversación casual, y no se requiere fidelidad a fuentes recuperadas, puede aumentarse la temperatura para generar respuestas más variadas y menos repetitivas.

Tanto el *prompting* como el ajuste de parámetros de decodificación son técnicas apropiadas para este proyecto, ya que presentan bajos requerimientos computacionales, costos reducidos o nulos y pueden implementarse fácilmente utilizando algún *framework*. Si bien anteriormente se mencionó que el uso de *prompting* puede implicar ciertos costos en función de la longitud del *prompt*, en este caso particular no representa una limitación, dado que se espera un volumen acotado de consultas y una cantidad moderada de instrucciones, sin impactar significativamente en los costos.

A su vez, el sistema requiere una personalización del conocimiento, ya que las respuestas deben basarse en información contenida en un repositorio de documentos curados, incluyendo las citas correspondientes ([RF-2](#)). Tal como se detalla en la sección [2.2.2](#) sobre métodos de personalización del conocimiento, la arquitectura RAG permite incorporar conocimiento específico desde bases de datos externas, asegurando respuestas fundamentadas en información confiable. Esta estrategia, además de mitigar el riesgo de alucinaciones, garantiza que las respuestas puedan acompañarse de las fuentes utilizadas y sean trazables.

En contraste, el *fine-tuning*, implica costos computacionales y financieros elevados, fuera del alcance de este proyecto, y presenta mayores riesgos de alucinaciones, lo cual no es aceptable en el contexto del sistema a desarrollar.

Además, esta técnica tampoco ofrece mecanismos para incluir referencias, lo cual constituye un requerimiento clave del sistema.

Si bien la adopción de RAG puede incrementar los tiempos de respuesta, esta desventaja se considera aceptable en favor de una mayor confiabilidad y verificabilidad de las respuestas. Finalmente, el enfoque RAG proporciona una mayor flexibilidad en la actualización del conocimiento, al permitir expandir o cambiar el dominio temático de manera ágil, alineándose así con el requerimiento de escalabilidad (RNF-3).

Desde la perspectiva de la gobernanza de datos, el uso de documentos curados con licencias claras (por ejemplo, Creative Commons) y con datos de autoría completos contribuye a mitigar los riesgos de infracción de derechos de autor, reforzando el cumplimiento del requerimiento de protección de la propiedad intelectual (RNF-2). A su vez, resulta fundamental contar con un mecanismo adecuado de extracción de metadatos durante el procesamiento de los documentos, de modo que la información utilizada sea precisa, completa y trazable.

Para controlar el acceso al sistema y el uso de los servicios de modelos de lenguaje —los cuales, al estar provistos por un proveedor externo, pueden implicar costos—, es necesario contar con un sistema de autenticación. Además, esto permite asociar las conversaciones a cuentas individuales, habilitando funcionalidades como el guardado, la continuación y la eliminación de conversaciones por parte de cada usuario (RF-6). Es imprescindible que este sistema de autenticación garantice la persistencia segura de las credenciales para satisfacer el requerimiento de confidencialidad y seguridad de los datos (RNF-1). Asimismo, se deben implementar mecanismos de cifrado de la información en tránsito, y asegurar una gestión y almacenamiento adecuados de los historiales de conversación para satisfacer este requerimiento.

Además, dado que el sistema accede a modelos a través de servicios externos, es fundamental evaluar las políticas de privacidad y cifrado de cada proveedor, eligiendo aquellos que aseguren que las conversaciones no se utilizan para reentrenar sus modelos, y que ofrezcan cifrado de la información tanto en tránsito como en reposo.

Cabe aclarar que, si bien el sistema prioriza el uso de herramientas de código abierto conforme al criterio establecido en el requerimiento RNF-1, la decisión de utilizar modelos cerrados responde a la necesidad de garantizar un funcionamiento robusto y una experiencia satisfactoria para el usuario, por lo que se considera aceptable sacrificar este criterio en favor de un desempeño significativamente superior del sistema.

Por último, los requerimientos de escalabilidad y mantenibilidad (RNF-3, RNF-4) se abordan mediante buenas prácticas de programación y el diseño de una arquitectura modular y escalable, facilitando la incorporación futura de nuevos dominios temáticos o la sustitución de proveedores de IA.

La integración de los distintos criterios funcionales y no funcionales, junto

con las decisiones técnicas adoptadas, las consideraciones de gobernanza y los principios de ingeniería contemplados, aseguran un sistema robusto, alineado con los objetivos del proyecto y preparado para evoluciones futuras.

3.2. Revisión de Antecedentes

3.2.1. Análisis de soluciones existentes

Como se expone en la sección 1.1, uno de los objetivos del proyecto es realizar un relevamiento bibliográfico y un análisis de herramientas existentes relacionadas con chatbots educativos e inteligencia artificial generativa. En la siguiente sección se presenta un análisis detallado de los artículos considerados relevantes para la investigación. El primer artículo revisado, “Pyo, a Chatbot Assistant for Introductory Programming Students”, fue proporcionado por las tutoras y sirvió como punto de partida para el estudio. A partir de este artículo, se llevó a cabo un proceso de “*snowballing*” sobre sus referencias, identificando publicaciones clave que se alinean con los objetivos de la investigación. Complementariamente, se llevó a cabo una búsqueda en la base de datos Scopus ([Agencia Nacional de Investigación e Innovación, 2021](#)), accedida a través del portal Timbó, utilizando las siguientes consultas:

- TITLE-ABS-KEY ((“Retrieval-Augmented Generation” OR “RAG”) AND (“Fine-tune” OR “Fine-tuning”) AND (“LLM” OR “chatbot”))
- TITLE-ABS-KEY ((“chatbot”) AND (“education” OR “learning” OR “teaching”))

A continuación, se describen y analizan las publicaciones más relevantes encontradas, destacando sus aportes y su pertinencia para el desarrollo de un prototipo de chatbot con capacidad de recuperación de información, generación de respuestas fundamentadas y un enfoque pedagógico.

3.2.2. Artículos Revisados

Pyo, a Chatbot Assistant for Introductory Programming Students

Resumen

- Este artículo presenta el desarrollo y evaluación de *Pyo*, un chatbot diseñado para asistir a estudiantes en cursos de programación introductoria, específicamente en el lenguaje Python. La motivación principal es la alta tasa de abandono en estos cursos y la necesidad de apoyo constante para los estudiantes principiantes.
- El sistema no utiliza LLMs, ni recupera información dinámica desde bases de datos. En su lugar, parece basarse en flujos de conversación predefinidos mediante árboles de decisión, posiblemente asistidos por módulos de análisis semántico para condicionar dichos flujos.

- Publicación disponible en: (Carreira, Silva, Mendes, y Oliveira, 2022)

Puntos Importantes

- **Motivación:** Destaca que el alto índice de abandono en cursos de programación se debe, en parte, a la falta de apoyo para los estudiantes. Plantea que esta problemática podría abordarse mediante el uso un chatbot disponible en todo momento.
- **Trabajos Relacionados:** Revisa estudios previos y tecnologías relacionadas con el uso de chatbots en la educación.
- **Diseño del Chatbot:** Describe la arquitectura del chatbot, implementado en Python utilizando el *framework* RASA, e integrando técnicas de procesamiento de lenguaje natural para interpretar y responder preguntas. Detalla las estrategias pedagógicas implementadas para mejorar la enseñanza y el aprendizaje de la programación.
- **Estrategias Pedagógicas:** El chatbot implementa estrategias pedagógicas como realizar preguntas al estudiante o darle tiempo antes de proporcionar ayuda. Además, se menciona que el enfoque educativo adoptado se basa en la teoría Vygotskiana de la zona de desarrollo próximo.
- **Evaluación y Resultados:** Presenta los métodos utilizados para evaluar la efectividad de *Pyo* en un entorno educativo. Si bien la cantidad de datos recolectados no permite extraer conclusiones robustas, el enfoque propuesto puede servir como referencia para otros proyectos al momento de definir métodos de evaluación.

Python-Bot: A Chatbot for Teaching Python Programming

Resumen

- Este artículo describe el desarrollo e implementación de *Python-Bot*, un chatbot educativo para enseñar programación en Python. Utiliza la plataforma SnatchBot para interactuar con los estudiantes y proporcionar apoyo en tiempo real.
- Al igual que el sistema anterior, se basa en flujos conversacionales programados, y no emplea LLMs, ni mecanismos de recuperación de información de una base de datos.
- Publicación disponible en: (Okonkwo y Ade-Ibijola, 2020)

Puntos Importantes

- **Antecedentes y Trabajos Relacionados:** Define qué es un chatbot y describe su uso en diversos campos como la educación, la salud y las finanzas. Revisa trabajos previos relevantes en la educación asistida por chatbots.

- **Diseño de Python-Bot:** Describe la arquitectura, el uso de la API de SnatchBot, y el flujo de trabajo del sistema, incluyendo cómo se utilizan las capacidades de procesamiento de lenguaje natural para procesar las consultas de los usuarios. Además, especifica su despliegue como aplicación web móvil.
- **Interacción con Usuarios:** Proporciona ejemplos de interacciones entre el chatbot y los usuarios, demostrando cómo *Python-Bot* explica conceptos de programación y resuelve problemas.
- **Evaluación:** Describe el método de evaluación utilizado, que incluye una encuesta a estudiantes de la Universidad de Johannesburgo. Presenta los resultados de la encuesta, que indican una alta aceptación y percepción positiva de *Python-Bot* por parte de los estudiantes.

Adapting LLMs for Efficient, Personalized Information Retrieval: Methods and Implications

Resumen

- Este artículo explora cómo los Grandes Modelos de Lenguaje pueden mejorar los sistemas de Recuperación de Información (IR) al proporcionar respuestas personalizadas y contextualmente relevantes.
- Se examinan los desafíos y soluciones propuestas para la integración de LLMs con sistemas de IR. Si bien en este trabajo no se implementa un chatbot, resulta igualmente pertinente para esta investigación por su enfoque en la recuperación dinámica de información desde bases de datos.
- Publicación disponible en: ([Ghodratnama y Zakershahrak, 2023](#))

Puntos Importantes

- **Motivación:** Introduce la creciente necesidad de sistemas eficientes de recuperación de información debido al aumento masivo de datos digitales. Plantea las problemáticas de los sistemas de IR tradicionales y cómo los LLMs pueden proporcionar una comprensión más profunda y contextual de las consultas de los usuarios.
- **Desafíos de los LLMs:** Analiza los principales desafíos de los LLMs, en particular su limitación en la generación de respuestas precisas y relevantes debido a la falta de conocimiento actualizado.
- **Generación Aumentada por Recuperación (RAG):** Discute la tecnología de Generación Aumentada por Recuperación y cómo esta puede superar las limitaciones de los LLMs —cuyo conocimiento es estático— al incorporar datos dinámicos para mejorar la precisión de las respuestas.
- **Criterios de Selección de Modelos:** Comenta los factores clave para seleccionar un modelo adecuado, entre ellos, la especificidad del dominio, la escalabilidad, el sesgo y la privacidad de los datos.

- **Orquestación y Despliegue de Aplicaciones Basadas en LLMs:** Explora las herramientas y marcos disponibles para desarrollar y desplegar aplicaciones basadas en LLMs, como Langchain, LlamaIndex y DSPy. Describe las capacidades y metodologías de cada herramienta, destacando sus fortalezas y aplicaciones específicas.
- **Evaluación de Desempeño:** Discute la importancia de establecer métricas de evaluación adecuadas para los modelos y cómo estas métricas pueden aplicarse en diversos contextos. Presenta un enfoque híbrido de evaluación que combina métricas de capacidad existentes con nuevas métricas basadas en preferencias humanas.

Conditioning Chat-GPT for Information Retrieval: The Unipa-GPT Case Study

Resumen

- El artículo presenta *Unipa-GPT*, un chatbot basado en GPT-3.5-turbo, desarrollado para asistir a estudiantes de la Universidad de Palermo. Se analizan tanto el enfoque de Generación Aumentada por Recuperación como el *fine-tuning* para personalizar los conocimientos del LLM.
- Utiliza LLMs y recupera información dinámica de una base de datos.
- Publicación disponible en: (Siragusa y Pirrone, 2023)

Puntos Importantes

- **Motivación:** Expone el interés creciente en los modelos de lenguaje y la inteligencia artificial, destacando la popularidad de ChatGPT y su potencial en aplicaciones educativas. Presenta el objetivo de crear *Unipa-GPT* para ayudar a estudiantes de nivel secundario a elegir cursos de grado y responder preguntas relacionadas con la Universidad de Palermo.
- **Arquitectura del Sistema:**
 - **Herramientas:** Detalla el uso de herramientas como LangChain y FAISS.
 - **Enfoque RAG y Fine-Tuning:** Compara enfoques relevantes para la actualización del conocimiento del modelo, y la creación de un corpus específico para cada uno.
 - **Componentes principales:** *Unipa-GPT* se desarrolla como una arquitectura de tipo RAG, compuesta por dos módulos principales: el módulo de recuperación y el módulo generador.
 - **Módulo de Recuperación:** Utiliza la base de datos vectorial FAISS, integrada a través de LangChain, para la búsqueda de información. Los documentos del *unipa-corpus* (construido mediante técnicas de *scraping* en el sitio web de la universidad) fueron segmentados en

fragmentos, a partir de los cuales se generaron *embeddings* utilizando el modelo `text-embedding-ada-002` de OpenAI.

- **Módulo Generador:** Utiliza `GPT-3.5-turbo` con un *prompt* personalizado que guía el comportamiento del chatbot, integrando tanto la conversación previa como la nueva pregunta. También se implementó una versión alternativa mediante *fine-tuning*, utilizando una adaptación del mismo corpus.
- **Resultados Experimentales:** Presenta los resultados de pruebas cualitativas con estudiantes de secundaria, evaluando la precisión y utilidad de las respuestas generadas por diferentes configuraciones del sistema. Estas configuraciones incluyen tres corpus diferentes para la recuperación, modelos con y sin *fine-tuning*, y dos variantes para el *prompt*. Los resultados muestran que la configuración que combinó el modelo sin *fine-tuning*, un *prompt* condensado y el corpus `unipa-corpus-clear` alcanzó el mejor desempeño. Por otro lado, el uso de *fine-tuning* no aportó mejoras significativas y a veces produjo respuestas largas y repetitivas.
- **Conclusiones y Trabajos Futuros:** Resume los hallazgos principales, destacando que el uso de *prompts* condensados y un corpus ligero proporciona mejores resultados. Propone líneas futuras de trabajo, como la mejora del *prompt* mediante *prompt-tuning* o *prompt engineering*, selección adaptativa del corpus y el desarrollo de versiones de *Unipa-GPT* que mantengan la misma arquitectura pero usen distintos LLMs.

CarExpert: Leveraging Large Language Models for In-Car Conversational Question Answering

Resumen

- Este artículo introduce *CarExpert*, un sistema de pregunta-respuesta conversacional específico para automóviles. Se discuten los desafíos y soluciones para generar respuestas seguras y específicas del dominio automotriz.
- Utiliza LLMs y recupera información dinámica de una base de datos.
- Publicación disponible en: (Rony y cols., 2023)

Puntos Importantes

- **Motivación:** Detalla la motivación detrás de *CarExpert*, destacando los problemas de los LLMs en la generación de respuestas en dominios específicos, como la tendencia generar respuestas incorrectas o irrelevantes (alucinaciones) debido a la falta de conocimiento actualizado o del contexto adecuado.
- **Arquitectura del Sistema:**

- **Estructura General:** La arquitectura propuesta se organiza en torno a cuatro sub-tareas clave para la generación de respuestas fundamentadas en información: orquestación, búsqueda semántica, generación de la respuesta y moderación de la respuesta.
 - **Orquestador:** Controla la entrada del usuario y maneja escenarios de múltiples intercambios. Filtra preguntas inseguras y fuera de contexto.
 - **Búsqueda Semántica:** Utiliza bases de datos vectoriales para buscar documentos relevantes en base a las consultas de los usuarios.
 - **Generación de Respuestas:** Emplea tanto modelos extractivos como generativos para generar respuestas a partir de los documentos recuperados.
 - **Moderación de Respuestas:** Selecciona la mejor respuesta (extractiva o generativa) utilizando criterios como la similitud de coseno y el puntaje de extracción (*extraction score*).
- **Evaluación Experimental:**
- **Evaluación Cuantitativa:** Comparación de modelos de *embeddings* (como BM25, DPR, Sentence-transformers) y modelos de lenguaje (GPT-3.5-turbo, Luminous-extended) en métricas como MRR, F1-Score y Exact Match.
 - **Evaluación Cualitativa:** Comparación de respuestas proporcionadas por *CarExpert* y por GPT-3.5-turbo con y sin acceso a documentos.
- **Resultados:** *CarExpert* supera a GPT-3.5-turbo (tanto con como sin documentos) en generar respuestas precisas y específicas del dominio automotriz. Se destaca la importancia de la moderación de respuestas y el uso de *prompts* diseñados específicamente.

3.2.3. Comentarios de la Revisión

La revisión de antecedentes muestra casos relevantes tanto en el uso de chatbots en el ámbito educativo como en el desarrollo de sistemas de recuperación de información. Los primeros artículos analizados, como “Pyo, a Chatbot Assistant for Introductory Programming Students” y “Python-Bot: A Chatbot for Teaching Python Programming”, se enfocan en chatbots con flujos de decisión predefinidos y limitadas capacidades de procesamiento de lenguaje natural. Sin embargo, artículos más recientes como “Adapting LLMs for Efficient, Personalized Information Retrieval: Methods and Implications”, “Conditioning ChatGPT for Information Retrieval: The Unipa-GPT Case Study” y “CarExpert: Leveraging Large Language Models for In-Car Conversational Question Answering” muestran un avance hacia la integración de Grandes Modelos de Lenguaje y Generación Aumentada por Recuperación (RAG), proporcionando respuestas

más precisas y contextualmente relevantes. Además de mejorar la capacidad de recuperación de información, los LLMs permiten interacciones más humanas, facilitando la implementación de estrategias pedagógicas. Este avance se alinea estrechamente con los objetivos del presente proyecto de grado. Por ello, los últimos artículos han tenido una mayor influencia en la elaboración de la solución planteada en la siguiente sección, aunque los primeros artículos también han sido valiosos al proporcionar ejemplos de la aplicación de técnicas pedagógicas y su implementación en el ámbito educativo.

3.3. Solución Planteada

A continuación se presenta la solución planteada, para la cual se describe tanto el diseño y arquitectura (3.3.1) como la implementación (3.3.2) que responde a los requerimientos del cliente y al análisis de antecedentes realizado. Es importante destacar que, durante el desarrollo del proyecto y en las reuniones con las usuarias expertas, fueron surgiendo nuevas ideas que podrían ampliar y fortalecer la solución. Sin embargo, debido a limitaciones de tiempo y alcance, algunas de estas propuestas se han reducido o postergado. Aún así, se han contemplado en el diseño y la arquitectura para que, si en el futuro se considera oportuno, puedan implementarse sin alterar de forma sustancial la estructura propuesta.

3.3.1. Diseño y Arquitectura

En esta sección se describe la solución propuesta a nivel de diseño y arquitectura, en función de los requerimientos funcionales (3.1.1) y no funcionales (3.1.2) planteados. En la Figura 3.1 se muestra el diagrama de componentes en alto nivel, en la Figura 3.2 se ilustra con mayor detalle la estructura del *Frontend* y sus pantallas principales, en la Figura 3.3 se presenta la composición interna del *Backend* con sus distintos LLMs y, finalmente, en la Figura 3.4 se describe de forma más detallada la componente RAG ubicada en el *Backend*. Se comenzará proporcionando comentarios generales sobre las principales componentes del sistema (Figura 3.1) para luego profundizar en cada una de ellas.

- **Separación de responsabilidades entre *Frontend* y *Backend*:** Se establece una separación definida entre *Frontend* y *Backend*, para que la solución sea más mantenible (RNF-4) y escalable (RNF-3), según lo solicitado en los requerimientos no funcionales. Al desacoplar la capa de presentación de la capa lógica, se facilita la integración de múltiples componentes de *Frontend* (por ejemplo, una aplicación móvil que consuma el mismo *Backend*).
- **Uso de proveedor externo de LLMs:** Como se expuso en el Análisis Técnico de los Requerimientos (3.1.3), se optó por incorporar un proveedor de modelos de lenguaje en lugar de utilizar un modelo que pueda ejecutarse localmente en el *Backend* o en una componente aparte. Dado que

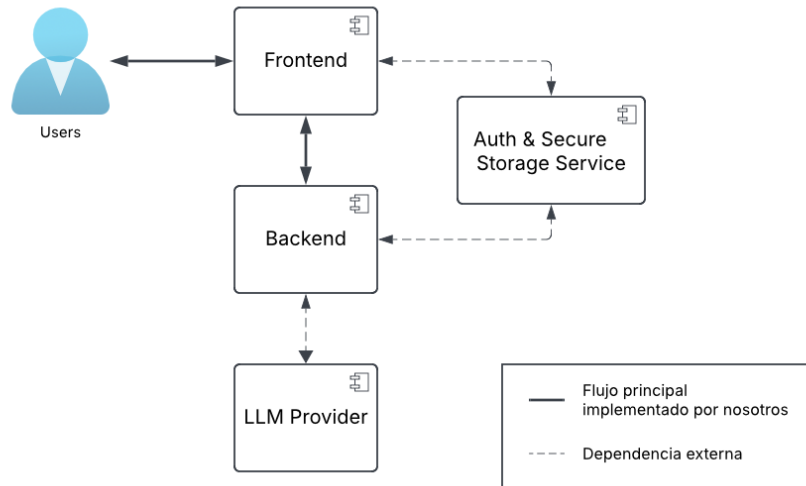


Figura 3.1: Diagrama de componentes en alto nivel.

uno de los requerimientos es garantizar la confidencialidad y seguridad de los datos (RNF-1), es importante utilizar un proveedor que asegure una gestión adecuada de la información sensible que se envía a sus APIs.

- **Integración de servicio de autenticación y almacenamiento seguro:** Se decidió integrar un servicio externo encargado de la autenticación y del almacenamiento seguro de los datos de los usuarios, en lugar de implementarlo directamente en el *Backend*. Esto se debe a la complejidad técnica que supone y a que no es el foco de la tesis. Sin embargo, sigue siendo esencial contar con algún sistema de autenticación para permitir que solo los usuarios registrados puedan acceder, evitando exponer el chatbot a accesos no autorizados que podrían generar costos significativos. Además, es necesario que este sistema sea seguro para cumplir el requerimiento no funcional de confidencialidad y seguridad de los datos (RNF-1).

A un nivel de abstracción más bajo, la componente de **Frontend** (Figura 3.2) se compone de:

Login Screen

- Es la primera pantalla, que se presenta al usuario en una nueva sesión. Permite al usuario ingresar las credenciales que se le han brindado, además de contener información sobre el proyecto. En un futuro se podría incorporar la opción de registrarse; sin embargo, para los fines de este proyecto se mantiene de esta manera para limitar el acceso y evitar exponer el chatbot al público.

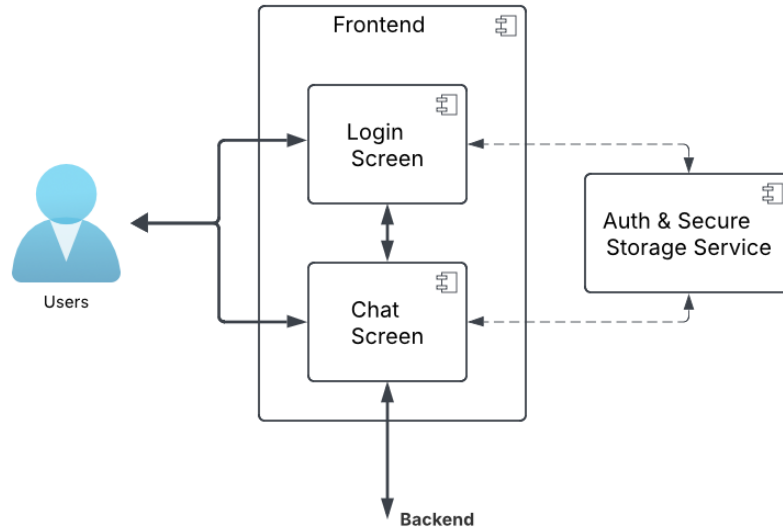


Figura 3.2: Estructura del *Frontend* con sus pantallas principales (*Login* y *Chat*).

- Se comunica con el servicio de autenticación para corroborar que las credenciales sean correctas. Si la autenticación es exitosa, se guarda un *token* de autenticación y se navega a la siguiente pantalla (*Chat Screen*).

Chat Screen

- Es la pantalla que se presenta al usuario luego de una autenticación exitosa. En ella, el usuario puede interactuar con el chatbot mediante una interfaz de chat y acceder a información sobre el proyecto. También puede cerrar la sesión (limpiando el *token* de autenticación) para volver a la pantalla de inicio.
- Se comunica con el servicio de almacenamiento seguro de datos para obtener el historial de chats del usuario. Este puede continuar conversaciones pasadas, iniciar una nueva, o eliminar cualquiera de ellas, cumpliendo así con el requerimiento funcional de manejo del historial de conversación (RF-6). Asimismo, contribuye al cumplimiento del requerimiento de confidencialidad y seguridad de los datos (RNF-1), al ofrecer un mecanismo explícito para borrar las conversaciones en caso de que el usuario no quiera que permanezcan almacenadas.
- En caso de querer mandarle un mensaje al chatbot, la pantalla se comunica con el *Backend* enviando el contenido, el identificador de conversación y el *token* de autenticación. El *Backend* responde con un nuevo mensaje o con un error en caso de haberlo al procesar la solicitud.

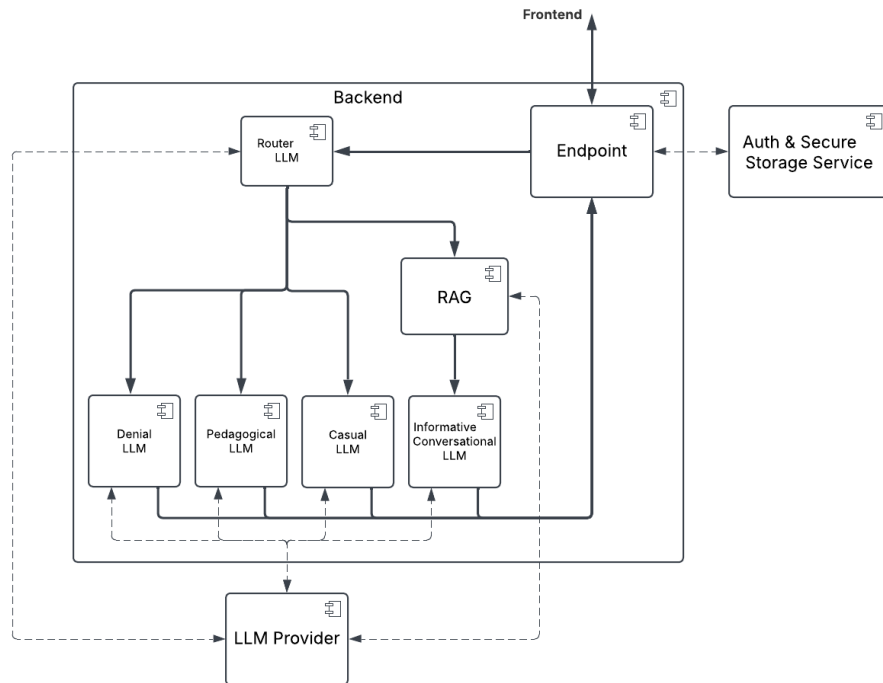


Figura 3.3: Estructura interna del *Backend* y sus distintos LLMs.

A un nivel de abstracción más bajo, la componente de **Backend** (Figura 3.3) se compone de:

Endpoint

- Es el punto de entrada al *Backend* con el que interactúa la *Chat Screen* del *Frontend*. Se comunica con el servicio de autenticación y almacenamiento seguro para validar que el *token* de autenticación recibido sea válido y para recuperar el historial de conversación correspondiente al identificador recibido (así como para guardar el nuevo mensaje generado al final del flujo).
- Llama al *Router LLM* con el mensaje ingresado y el historial de conversación para que decida el camino a tomar para procesar la solicitud. Una vez generada la respuesta, independientemente del camino, la devuelve al *Frontend*.

Router LLM

- Componente encargada de decidir, entre un conjunto predefinido de caminos, cuál es el más acorde en base a un historial de conversación y un

nuevo mensaje. Consiste en un LLM configurado con salida estructurada para que *únicamente* devuelva el identificador de uno de los caminos definidos, aprovechando la inteligencia del LLM pero limitando sus respuestas a un formato específico que permita integrarlo en un flujo programático.

- El conjunto de caminos fue definido en base a los requerimientos funcionales. Cada camino se desarrolla en componentes separadas, detalladas a continuación.

RAG

- Componente encargada de devolver información relevante (si existe) del repositorio de documentos curados, en base a un historial de conversación y un nuevo mensaje. Devuelve, por un lado, información que responde la consulta del usuario y, por otro, las fuentes correspondientes a dicha información.
- Más adelante (Figura 3.4) se profundiza en el diseño de la solución para la implementación de esta componente. Un *RAG* es un tema lo suficientemente amplio como para ser objeto de un proyecto de grado completo (como el de Ignacio Sastre (Sastre, 2024)), por lo que en esta solución se aísla en una componente específica, para facilitar su reemplazo o mejora futura y cumplir con los requerimientos no funcionales de mantenibilidad (RNF-4) y escalabilidad (RNF-3).

Informative Conversational LLM

- Componente encargada de devolver una respuesta basada en información del repositorio de documentos curados. Consiste en un LLM fuertemente instruido para cumplir con el requerimiento funcional de interacción basada en diálogo (RF-3) y responder *únicamente* con la información proporcionada por el RAG, garantizando también así el requerimiento de basar las respuestas en fuentes curadas (RF-2).
- Este camino se elige cuando el usuario envía un mensaje relacionado con inteligencia artificial o su uso en la educación, cuya respuesta puede obtenerse a partir del repositorio de documentos curados mediante el RAG, y no se considera oportuna una estrategia pedagógica. La respuesta de esta componente, junto a las fuentes que devuelve el RAG, se envía al *Endpoint* para su devolución al *Frontend*.

Casual LLM

- Componente encargada de manejar interacciones informales con el usuario sin necesidad de recuperar información del repositorio de documentos curados. Consiste en un LLM instruido para generar respuestas naturales que fomenten la interacción y mantengan la conversación enfocada en

inteligencia artificial, sin introducir información nueva no mencionada en el historial, cumpliendo así con la necesidad de basarse únicamente en fuentes curadas.

- Este camino se elige principalmente cuando el usuario envía un mensaje casual (por ejemplo, un saludo o un agradecimiento) sin intención de solicitar algo específico. También se selecciona cuando el usuario solicita una aclaración o reformulación de algún mensaje del chatbot, ya que en estos casos no es necesario recuperar nueva información del repositorio de documentos curados. Además, si el usuario realiza preguntas sobre el chatbot, también se toma este camino para responderlas. La respuesta de esta componente, se envía a la componente *Endpoint* para su devolución al *Frontend*.

Pedagogical LLM

- Componente encargada de responder aplicando estrategias pedagógicas. Consiste en un LLM instruido para cumplir el requerimiento de interacción basada en estrategias pedagógicas (RF-4), en particular implementando la estrategia de repregunta.
- Este camino se selecciona cuando el mensaje del usuario puede responderse eficazmente mediante una estrategia pedagógica, y se considera más adecuado aplicar este enfoque para promover el pensamiento crítico, en lugar de responder con información del repositorio de documentos curados. La respuesta de esta componente, se envía a la componente *Endpoint* para su devolución al *Frontend*.

Denial LLM

- Componente encargada de comunicar al usuario que el enfoque del chatbot está limitado a temas relacionados con la inteligencia artificial. Consiste en un LLM instruido para recordar el propósito de este chatbot, cumpliendo con el requerimiento de interacción restringida a temas de inteligencia artificial (RF-1).
- Este camino se elige principalmente cuando el usuario intenta utilizar el chatbot para un tema ajeno a la inteligencia artificial. La respuesta de esta componente, se envía a la componente *Endpoint* para su devolución al *Frontend*.

A continuación, se profundiza en la composición de la componente **RAG**, la cual, en términos generales, combina:

- Una base de datos vectorial que almacena los documentos fragmentados y transformados en su representación vectorial (*embeddings*).

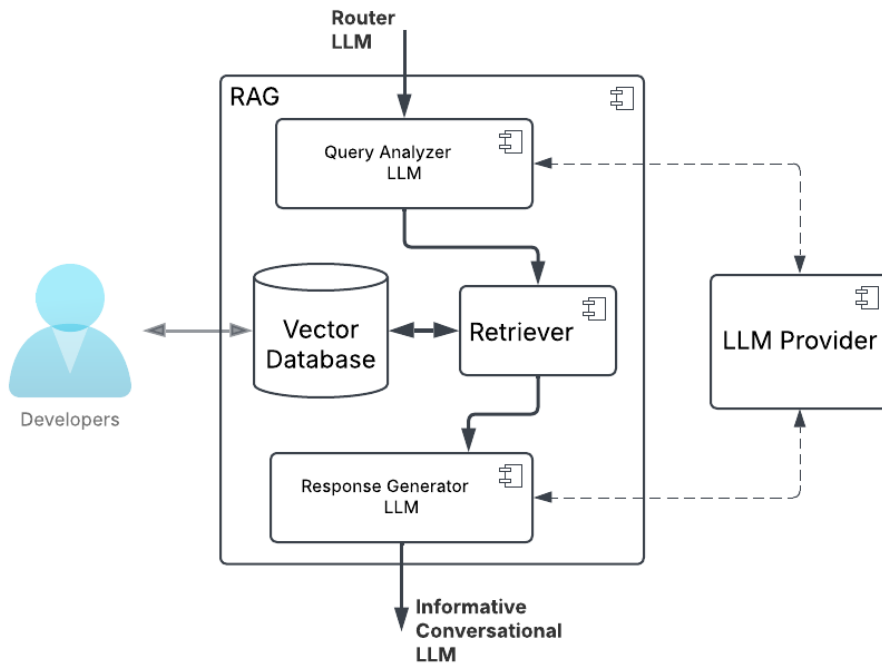


Figura 3.4: Detalle de la componente RAG con sus subcomponentes (Query Analyzer, Retriever, Vector Database y Response Generator).

- Una componente de recuperación, responsable de convertir las consultas en su representación vectorial y buscar, mediante algoritmos de búsqueda por similitud, fragmentos semánticamente relacionados que puedan dar respuesta al usuario.
- Dos LLMs adicionales para preprocesar (antes de la recuperación) y post-procesar (luego de la recuperación) la información.

La composición específica de la componente **RAG** en esta solución se presenta en la Figura 3.4 y se detalla a continuación:

Query Analyzer LLM

- Componente que preprocesa el mensaje enviado por el usuario junto al historial de conversación para optimizar la recuperación de información relevante del repositorio de documentos curados. No es suficiente con enviar únicamente el último mensaje al *Retriever* (responsable de la recuperación) por varias razones:
 - El mensaje puede hacer referencia a mensajes anteriores, por lo que el usuario no siempre repite todos los detalles relevantes a la consul-

ta en el último mensaje, lo que podría dificultar la recuperación de información de manera adecuada.

- El mensaje puede contener varias consultas o elementos semánticos diferentes, que al momento de convertirse en la representación vectorial, puede alejarse del espacio vectorial de cada subconsulta/elemento independiente, empeorando la recuperación.
 - El mensaje puede contener acrónimos o abreviaciones que pueden empeorar la recuperación, dado que 2-3 letras no tienen la misma carga semántica que el significado del acrónimo. Por ejemplo, “LLM” podría generar una peor recuperación que “Large Language Model”, ya que el modelo de embeddings utilizado podría no haber aprendido correctamente la representación vectorial del acrónimo en el espacio vectorial de los grandes modelos de lenguaje.
- Es por estos motivos que el *Query Analyzer LLM* es instruido a generar subconsultas que aumenten la probabilidad de encontrar documentos relevantes (se puede ahondar sobre técnicas de *query analysis* en el anexo E). Estas subconsultas deben ser independientes entre sí, abordando las distintas partes del mensaje original, en caso de que incluya múltiples ideas o preguntas. Además deben resolver referencias a mensajes anteriores, e incorporar el contexto necesario para ser auto-contenidas. Cuando sea pertinente, deben incluir tanto los acrónimos como sus formas extendidas, con el fin de maximizar la cobertura semántica durante el proceso de recuperación.
 - Además, el *Query Analyzer LLM* es instruido a generar una versión actualizada de la consulta original del usuario, reformulándola para que sea auto-contenida. Esta reformulación incorpora el contexto relevante del historial de conversación y resuelve cualquier referencia a mensajes anteriores, con el objetivo de que pueda ser interpretada de forma aislada. Esta consulta actualizada es enviada al *Response Generator LLM*, encargado de generar la respuesta final de la componente RAG. Dado que la función del *Response Generator LLM* es construir una respuesta exclusivamente a partir de los fragmentos recuperados, no recibe el historial de conversación, por lo cual necesita una consulta auto-contenida, que incluya toda la información necesaria para interpretar la intención del usuario.
 - Para devolver tanto el arreglo de *subconsultas* como la consulta *actualizada*, este LLM se configuró con salida estructurada (*Structured Output*), de modo que su respuesta se ajuste a un formato predefinido y pueda ser utilizada directamente por los componentes que la consumen: las *subconsultas* son utilizadas por el *Retriever* y la *consulta actualizada* por el *Response Generator LLM*.

Retriever

- Componente encargada de realizar la recuperación de fragmentos de documentos relacionados con las subconsultas devueltas por el *Query Analyzer LLM*.

Vector Database

- Componente que almacena las representaciones vectoriales de los fragmentos del repositorio de documentos curados, junto con sus versiones en lenguaje natural.
- La carga de documentos del repositorio en la base de datos se realiza mediante un *script* que lleva a cabo los siguientes pasos:
 1. Carga los documentos a procesar
 2. Extrae la metadata de manera automatizada, utilizando un LLM para obtener los datos requeridos (título, autores y año)
 3. Fragmenta los documentos (*chunking*)
 4. Obtiene la representación vectorial de cada fragmento usando el mismo modelo de *embeddings* empleado por el *Retriever*, como fue explicado al introducir la arquitectura RAG en la sección 2.2.2.
 5. Incorpora los fragmentos, junto con su representación vectorial y metadata, a la base de datos vectorial.

Observar que el paso 2, correspondiente a la extracción de metadata, permite asociar cada fragmento con sus datos de autoría y almacenarlos en la base de datos. Esta información puede ser utilizada posteriormente para acompañar las respuestas generadas con sus respectivas citas. Sumado a la selección previa de recursos educativos abiertos, libres y accesibles, esto contribuye al cumplimiento del requerimiento de protección de la propiedad intelectual (RNF-2).

Response Generator LLM

- Componente encargada de postprocesar los fragmentos recuperados por el *Retriever*, seleccionando aquellos más relevantes para responder la consulta actualizada (generada por el *Query Analyzer LLM*). El LLM se configuró con salida estructurada para que *únicamente* devuelva un formato predefinido que contenga:
 1. La respuesta final basada en la información recuperada.
 2. Los fragmentos utilizados, acompañados de sus respectivas fuentes (título, autor, año y nombre del documento).
- La respuesta del *Response Generator LLM* se envía al *Informative Conversational LLM* para ser adaptada a un formato conversacional, cumpliendo con el requerimiento funcional de interacción basada en diálogo (RF-3), y

respetando la información original sin añadir detalles adicionales. Por otro lado, las fuentes se envían directamente al *Endpoint* para ser mostradas como citas en el *Frontend*.

- En caso de que los fragmentos recuperados por el *Retriever* no contengan información relevante para responder la consulta, la componente devuelve como respuesta “*No information found*” en el campo destinado a la respuesta, dejando vacío el campo correspondiente a los fragmentos y sus fuentes.

En síntesis, esta arquitectura busca cumplir con los requerimientos funcionales 3.1.1 y no funcionales 3.1.2, separando cada función crítica en componentes específicos que pueden mejorarse o reemplazarse con relativa facilidad.

3.3.2. Implementación

En esta sección se describe la implementación de la solución, tomando como punto de partida el diseño y la arquitectura presentada previamente. En la Figura 3.5 se muestra el diagrama de componentes a alto nivel con las tecnologías y dependencias reales utilizadas.

Elección de Tecnologías:

- **Python para el *Backend*:** Se eligió *Python* para el desarrollo del *Backend*, debido a su popularidad en el desarrollo de aplicaciones basadas en IA generativa y por la abundante documentación y soporte en frameworks relevantes (por ejemplo, *LangChain*). Este lenguaje ofrece un entorno muy completo para implementar la lógica de negocio y la interacción con el proveedor de LLM.
- **React para el *Frontend*:** Se optó por *React* para la implementación del *Frontend* debido a la experiencia previa del equipo y a su adecuación para la creación rápida de interfaces relativamente simples, como las pantallas de *Login* y *Chat*. Además, su ecosistema facilita el mantenimiento y la extensión de funcionalidades si se quisiera evolucionar la solución en el futuro.
- **Docker para la Contenerización y Despliegue:** Se agruparon los principales componentes del sistema (*Backend*, *Frontend* y la base de datos *Milvus*) dentro de contenedores de *Docker* y se realizó el despliegue en una máquina virtual provista por *Antel*. Esto permitió simplificar el proceso de despliegue y asegurar la reproducibilidad del entorno en la máquina remota en la cual se disponibilizó la implementación.
- **Firestore (Google) para Autenticación y Almacenamiento Seguro:** Para cubrir el servicio de autenticación y almacenamiento seguro de datos descrito en el capítulo de diseño, se utilizó *Firestore*, un servicio en

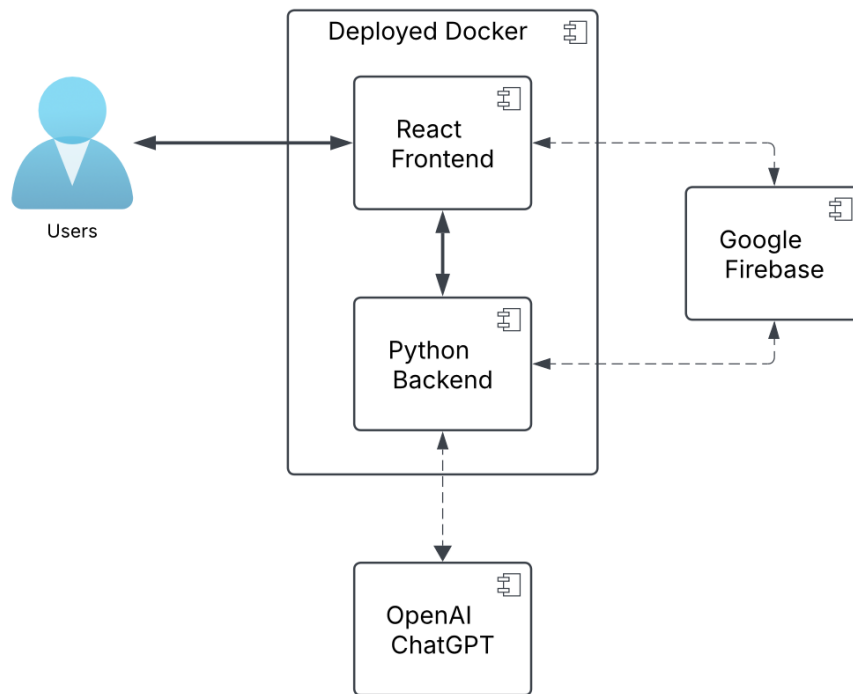


Figura 3.5: Diagrama de componentes en alto nivel con las tecnologías y dependencias utilizadas.

la nube de Google que ofrece distintos módulos como **FirebaseAuth** y **Firestore**. Esta solución se eligió aprovechando sus garantías de seguridad y su facilidad de uso.

- **OpenAI como Proveedor de LLMs:** Para el proveedor de LLMs, se utilizó *OpenAI*, que provee algunos de los modelos más potentes en la industria actualmente. Un factor clave en su selección, además de su desempeño, fue el manejo de los datos de los usuarios. Según la documentación oficial de OpenAI ([OpenAI, 2023a](#)), no retienen los datos de los usuarios ni los utilizan para entrenar modelos cuando se consulta por API, salvo que el usuario lo autorice explícitamente, cumpliendo así con el requerimiento de confidencialidad y seguridad de los datos ([RNF-1](#)).

A continuación, se detalla cada componente y cómo se implementó efectivamente, manteniendo la misma estructura utilizada en la presentación de la arquitectura:

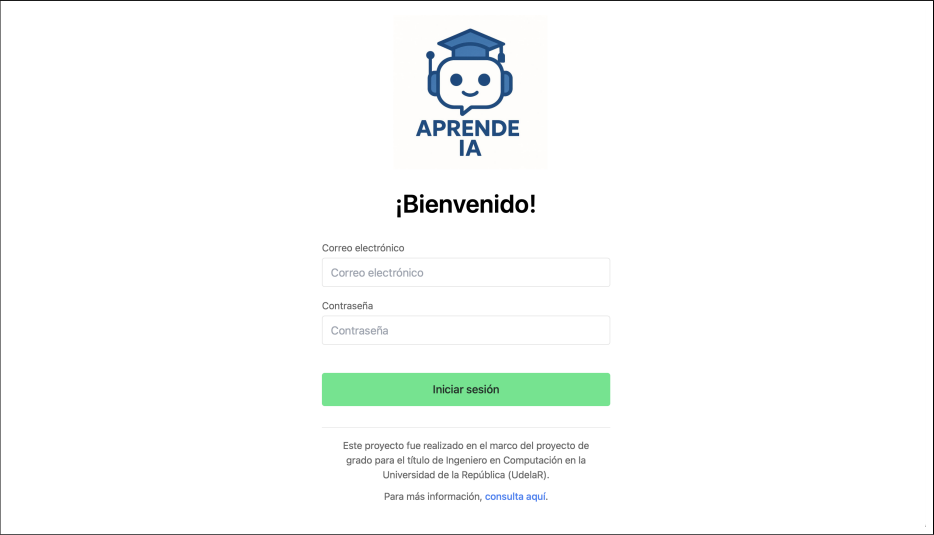
Frontend

La implementación del *Frontend* se destaca por ser moderna, desarrollada con **React** y **TypeScript**, estilizada con **Tailwind CSS**. La arquitectura se basa en principios de modularidad y reutilización, con un fuerte énfasis en la seguridad a través de rutas privadas y manejo seguro de variables de entorno. Además, se integra con el sistema robusto de autenticación de **Firebase**, y se optimiza la experiencia de usuario por medio de notificaciones, estados de carga y diseño *responsive*. El uso de **TypeScript**, contextos para el estado global y formularios validados contribuye a mantener altos estándares de calidad en el código.

En las Figuras 3.6 y 3.7 se puede observar cómo lucen las dos pantallas principales.

Login Screen

- Es la pantalla inicial a la que se redirige al usuario cuando *no* está autenticado. Se desarrolló con **React** y **React Router** para la navegación, implementando rutas privadas que exigen un *token* de autenticación para acceder a otras pantallas como la de *Chat*.
- El usuario ingresa sus credenciales, que se validan contra **FirebaseAuth**. Al autenticar con éxito, **Firebase** devuelve un *token* de sesión que se almacena temporalmente en la aplicación.
- Dicho *token* se incluye en las cabeceras de las solicitudes al Backend en cada interacción posterior, asegurando que sólo usuarios autenticados puedan acceder a la funcionalidad del chatbot.



The screenshot shows a login interface for 'APRENDE IA'. At the top is a logo of a blue robot head wearing a graduation cap, with the text 'APRENDE IA' below it. Below the logo is the heading '¡Bienvenido!'. There are two input fields: 'Correo electrónico' and 'Contraseña'. Below these is a green button labeled 'Iniciar sesión'. At the bottom, there is a small text block stating: 'Este proyecto fue realizado en el marco del proyecto de grado para el título de Ingeniero en Computación en la Universidad de la República (Udelar). Para más información, [consulta aquí](#).'

Figura 3.6: *Login Screen* implementada en React.

Chat Screen

- Luego de una autenticación exitosa, el usuario accede a la pantalla de *Chat*, en la cual se muestra un campo de texto para introducir mensajes y un listado de mensajes del historial actual.
- En cada interacción, se envía al *Backend* el *token* de sesión y el *historial de conversación* (mantenido localmente en el estado de la aplicación) para obtener la respuesta del chatbot.
- Debido a la limitación de tiempo del proyecto, se tuvo que repriorizar y reevaluar el alcance, y el requerimiento funcional del manejo del historial de conversación (RF-6) se pospuso como trabajo futuro. En consecuencia, *no* se está almacenando ni recuperando el historial de *Firestore* como se había contemplado en el diseño. En esta implementación, el historial es efímero y se reinicia al refrescar la página o cerrar la sesión.

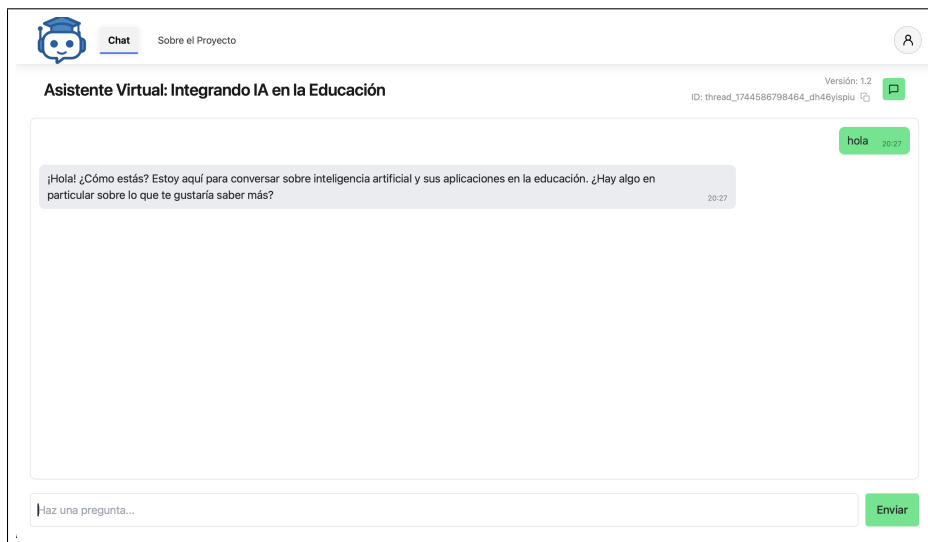


Figura 3.7: Chat Screen implementada en React.

Backend

La implementación del *Backend* está basada en **Python** y sigue un enfoque con múltiples LLMs para cubrir los distintos comportamientos especificados en los requerimientos funcionales. En el contexto de este proyecto, este enfoque resulta más adecuado que utilizar un único LLM que asuma todas las responsabilidades, ya que permite definir componentes especializadas, cada una orientada a cumplir un requerimiento funcional específico. Esto facilita afinar su comportamiento de forma independiente durante el desarrollo y evaluar de manera

aislada el grado de cumplimiento de cada requerimiento en la etapa de evaluación. Además, como se explicó en la sección 2.3.4, este tipo de arquitecturas promueve la escalabilidad y la mantenibilidad del sistema, al permitir agregar o modificar componentes de manera modular sin impactar al resto de la arquitectura. Esto contribuye directamente al cumplimiento de los requerimientos no funcionales de escalabilidad (RNF-3) y mantenibilidad (RNF-4).

Se destaca también la elección de **LangChain** como *framework* de desarrollo. Tal como se detalló en la sección 2.3.3, LangChain representa una opción robusta por su capacidad de integración con diversos modelos y bases de datos, el soporte activo de su comunidad y la abundante documentación disponible. A esto se suma la herramienta complementaria **LangSmith**, que facilita la trazabilidad, el monitoreo y la depuración del comportamiento del sistema durante el desarrollo. Finalmente, la experiencia previa del equipo con este *framework* fue un factor adicional para su elección, dado que permite aprovechar el conocimiento adquirido en proyectos anteriores y acelerar el proceso de implementación.

El código del *Backend*, al igual que el del *Frontend*, hace uso de *variables de entorno* para gestionar configuraciones sensibles de manera segura.

Endpoint

- Se utiliza el *framework* **FastAPI** para exponer una única ruta `POST /invoke_agent`, que recibe el *token* de autenticación y el mensaje del usuario, junto con el historial efímero suministrado por el *Frontend*.
- Se valida el *token* de sesión con **FirebaseAuth**; si es inválido, se devuelve un error, y si es válido, se invoca la lógica del *Router LLM* para procesar la solicitud. La respuesta generada por el flujo desencadenado se devuelve al *Frontend*, cumpliendo así con el ciclo de interacción entre el usuario y el sistema.

LLMs

- Para cada componente del sistema que requiere un LLM (por ejemplo, *Router LLM*, *Casual LLM*, *Pedagogical LLM*, etc.), se utilizó **LangChain** para instanciar modelos específicos de OpenAI (el proveedor de LLMs elegido) y gestionar su integración en los flujos programáticos.
- Se diseñaron *prompts* personalizados (detallados en el anexo B) para cada *LLM*, de modo que el modelo reciba instrucciones claras y específicas sobre el comportamiento esperado. Esto incluye indicaciones como mantener la conversación en un dominio restringido, hacer preguntas de seguimiento, responder sólo con información proporcionada, etc. Además, se especificaron intencionalmente los parámetros de decodificación, siguiendo criterios como configurar una *temperatura* baja cuando se quiere generar respuestas apoyadas en información recuperada, favoreciendo la precisión, y una *temperatura* más alta cuando se quiere que sostengan interacciones casuales, permitiendo mayor diversidad en las respuestas.

- Para los casos en que se requirió una *salida estructurada* (por ejemplo, un identificador de un camino en el caso del *Router LLM*), se implementaron las clases necesarias utilizando la librería **Pydantic**. Esto permitió forzar al modelo a devolver una respuesta en el formato deseado, aprovechando las ventajas de validación de datos y tipado que ofrece **Pydantic**. De esta manera, se enlazó de forma directa la salida del LLM con la lógica de la aplicación, asegurando su uso correcto en el flujo conversacional.
- La interacción con cada modelo se parametrizó mediante el uso de variables de entorno (por ejemplo, **OPENAI_API_KEY**), lo que permite resguardar la configuración sensible y facilita la conmutación a otros proveedores de *LLMs* si fuera necesario en el futuro. Esta práctica también mejora la seguridad y la flexibilidad del sistema al gestionar configuraciones y credenciales de forma centralizada y segura.
- El uso de **LangChain** como *framework* intermediario para gestionar la interacción con los *LLMs* permite abstraer esta lógica, a la vez que mantiene una arquitectura modular. Esta aproximación asegura un acoplamiento mínimo con las APIs específicas de cada proveedor, lo que facilita la extensibilidad del sistema. Así, en caso de querer cambiar o integrar nuevos proveedores de *LLMs* en el futuro, no será necesario reescribir grandes partes del código que interactúa con los modelos. Esta arquitectura no solo mejora la escalabilidad ([RNF-3](#)), sino que también simplifica el mantenimiento a largo plazo y garantiza una mayor flexibilidad ([RNF-4](#)).

RAG

- Se eligió **Milvus** como base de datos vectorial para almacenar las representaciones vectoriales (*embeddings*) de los documentos curados. Se comenzó el desarrollo utilizando **Faiss** ([Faiss, 2017](#)) pero rápidamente se migró a **Milvus** pues es un motor de base de datos más potente y especializado, alineado con el requerimiento de escalabilidad ([RNF-3](#)). También se contó con **attu** ([Milvus, 2023](#)), una herramienta que permite visualizar, consultar y modificar vectores de la base. Esta herramienta fue vital para validar que los fragmentos y sus metadatos estuvieran siendo correctamente almacenados en la base.
- Previo a la carga de documentos en la base de datos, se aplicó un proceso de *chunking semántico* para dividir los textos en fragmentos más pequeños y coherentes, basados en su contenido (se puede ahondar sobre técnicas de *chunking* en el anexo [F](#)). Para el cálculo de *embeddings* de cada fragmento se utilizó la API de **OpenAI**.
- Por último, en relación con la construcción del *repositorio de documentos curados*, para que un recurso sea incluido, se exige que sea de *acceso abierto*, es decir, que permita la reproducción o adaptación del contenido, siempre con la correspondiente cita a los autores. Se excluyen los textos

sin mención de licencia hasta tanto se compruebe su uso legítimo. Esta política garantiza el respeto por los derechos de autor y contribuye al cumplimiento del requerimiento de protección de la propiedad intelectual (RNF-2).

Deployed Docker

La estrategia de **despliegue** para esta solución se ha basado en la utilización de **Docker** para encapsular los distintos servicios que componen el sistema. El *Frontend*, el *Backend* y la base de datos vectorial se ejecutan en contenedores independientes que se inician de forma coordinada. De este modo, el *Backend* depende de la disponibilidad de *Milvus* para la funcionalidad de recuperación de información de la componente del *RAG*, y el *Frontend* precisa del *Backend* para gestionar las comunicaciones del usuario con el chatbot. El uso de Docker permite garantizar un entorno controlado y reproducible, minimizando problemas de compatibilidad entre librerías y facilitando la portabilidad y escalabilidad del sistema hacia otros servidores o entornos, cumpliendo así nuevamente el requerimiento no funcional de escalabilidad (RNF-3).

El despliegue remoto se llevó a cabo en una máquina virtual provista por *Antel*, la cual cuenta con una dirección IP pública que hace accesible el sistema a usuarios externos. Tras obtener el acceso inicial a la máquina (agregando las claves públicas SSH de los integrantes de la tesis), se procedió a clonar el repositorio de código alojado en *GitHub* (el servicio remoto de almacenamiento y gestión de repositorios de código elegido para el proyecto). Para ello, se generó un par de claves asimétricas en la máquina remota y se configuró la clave pública como *Deploy Key* en el repositorio, habilitando la clonación y actualización de código sin requerir autenticaciones manuales adicionales. A continuación, se siguieron los pasos descritos en el archivo *README* (documentación detallada sobre cómo ejecutar la implementación), se instaló Docker y se iniciaron los contenedores, exponiendo el *Frontend* en un puerto concreto de la IP pública. De esta forma, se habilitó el acceso al sistema sencillamente introduciendo la URL (<http://179.27.96.162:8081>).

Con el objetivo de evitar la necesidad de conectarse manualmente a la máquina virtual mediante SSH para actualizar los contenedores en cada nueva versión del sistema, se implementó un flujo de **CI/CD (Continuous Integration/Deployment)** basado en *GitHub Actions*. Para esto, se configuró una *action* que se ejecutase automáticamente cuando se detectasen cambios en la rama principal. Esta acción replica los pasos manuales: actualización del código en la máquina de despliegue con la última versión del repositorio remoto y reinicio de los contenedores. Para ello, se generó un nuevo par de claves asimétricas, colocando la clave privada como un *secret* en el repositorio de *GitHub* y registrando la clave pública como una llave SSH válida en la máquina de *Antel*. Así, la *action* puede conectarse de forma segura a la máquina remota e iniciar el proceso de despliegue sin intervención humana.

En términos de **versionado**, el proyecto adoptó un modelo con ramas **main** y **develop**. La rama **develop** reúne el desarrollo continuo y a partir de ella se

crean ramas de características (*feature branches*), que son objeto de revisiones (*pull requests*) antes de incorporarse a **develop**. Cuando se requiere una nueva versión estable para desplegar, se realiza un *merge* desde **develop** a **main**. El sistema de versionado está basado en un esquema **X.Y**, donde **X** indica la versión mayor y **Y** la versión menor. Cada vez que se fusiona a **main**, se incrementa en uno la parte **Y**. Para reflejar esta actualización, se mantiene un *archivo de texto* en el repositorio que se actualiza automáticamente en la acción de CI. El mismo proceso realiza un *commit* con dicha actualización y genera un *tag*, de forma que la versión queda registrada en el *Frontend* (mostrándose en la interfaz) y en las trazas capturadas por *LangSmith*, ya que es enviada como metadata.

Desde el punto de vista de **seguridad**, se hizo un ajuste selectivo de los puertos expuestos por Docker, de manera que *Milvus* quedase accesible exclusivamente para el *Backend* y no para clientes externos, protegiendo así sus operaciones internas y los datos indexados. Además, en el entorno de producción se suprimen los mensajes de error detallados que podrían revelar información sensible sobre el sistema. No obstante, una limitación relevante es que actualmente el acceso a la aplicación se realiza mediante el protocolo HTTP (sin cifrado), lo cual implica que las comunicaciones no están aseguradas, exponiendo potencialmente el contenido de las conversaciones a ataques del tipo *man in the middle*. Para resolver este problema, podría configurarse un certificado *SSL/TLS* (emitido por una autoridad certificadora), que asegure la autenticidad del servidor y cifre las comunicaciones (HTTPS). Al usar un certificado válido, los navegadores reconocerían la página como segura, y se protegerían los datos en tránsito. Debido a restricciones de alcance y tiempo en el desarrollo de este proyecto de grado, se decidió postergar esta tarea pero dejándola documentada para su posible inclusión en futuras iteraciones.

En conclusión, la implementación desarrollada cumple con la mayoría de los requerimientos definidos en conjunto con las usuarias expertas. No obstante, alguno de ellos fue postergado debido al tiempo acotado que se espera que se le dedique al proyecto. Dichos requerimientos podrían ser retomados y completados en iteraciones futuras, siguiendo las especificaciones definidas tanto en el diseño como en la sección 6.2 de trabajos futuros. La implementación completa puede ser consultada en el repositorio público del proyecto, disponible en: <https://github.com/Fedee120/ProyGrad2024>.

En el siguiente capítulo se profundizará sobre cómo se evaluó la calidad del prototipo desarrollado y en qué medida se alcanzaron los requerimientos establecidos.

Capítulo 4

Evaluación y Resultados

En este capítulo se expondrá el enfoque metodológico propuesto para evaluar la calidad y el correcto funcionamiento del prototipo de chatbot generativo para la educación, tanto durante su desarrollo como en la etapa de despliegue final. El objetivo principal de esta evaluación es asegurar que cada uno de los requerimientos sea cubierto de manera consistente y medible, así como identificar oportunidades de mejora. Se contemplan evaluaciones *offline*, *online* y con usuarios finales, promoviendo una visión integral del rendimiento y del cumplimiento de los requerimientos definidos.

4.1. Evaluación *Offline*

La evaluación *offline* se centra en medir y asegurar el correcto funcionamiento de cada uno de los componentes de la arquitectura de manera aislada, sin involucrar directamente a usuarios finales ni a entornos de producción. Para ello, se ejecuta un *script* que realiza pruebas sobre los distintos subsistemas, utilizando conjuntos de datos (*datasets*) predefinidos para evaluar su desempeño y analizar los resultados generados.

Este proceso se lleva a cabo antes de integrar cualquier cambio en la implementación, garantizando así la estabilidad y fiabilidad del sistema. De esta manera, se adopta un enfoque de desarrollo basado en métricas (*Metric-Driven Development*), que permite tomar decisiones de diseño fundamentadas en los resultados de las evaluaciones y garantizar que cada modificación introducida contribuya a mejorar las métricas definidas.

4.1.1. Objetivos de la Evaluación *Offline*

- **Evaluar el desempeño de cada componente:** Permite verificar que cada bloque del sistema (por ejemplo, el *Router LLM* o el *Response Generator LLM*) cumple con su función definida, garantizando la precisión y coherencia de las respuestas generadas.

- **Prevenir regresiones:** Contar con un conjunto de pruebas predefinido y un proceso de evaluación automatizado permite detectar y evitar degradaciones en el desempeño del sistema ante modificaciones en los *prompts*, parámetros de los modelos o lógica general del sistema, asegurando que los cambios no afecten negativamente ni a los componentes individuales ni al sistema en su conjunto.
- **Persistir resultados y tendencias:** Cada ejecución del *script* genera un archivo con las métricas obtenidas (p. ej., un JSON) que refleja el desempeño de cada componente. Esto permite llevar un histórico de mediciones y comparar la evolución del sistema a lo largo del tiempo.

4.1.2. Mecánica de la Evaluación *Offline*

La evaluación *offline* se desarrolla en varias etapas: primero, se definen métricas específicas para cada componente según su comportamiento esperado; luego, se diseñan conjuntos de pruebas representativos; posteriormente, se emplean LLMs como jueces para automatizar la verificación de métricas; y finalmente, los resultados se registran en un archivo JSON. A continuación, se detallan estos pasos.

Definición de métricas específicas por componente

Se establecen métricas para evaluar el desempeño de cada componente, considerando su función y el comportamiento esperado dentro del sistema.

- *Router LLM*
 - *Routing_Accuracy*: Evalúa si el *Router* selecciona el camino correcto acorde al comportamiento esperado especificado en el diseño.
- *Casual LLM*
 - *Remembers_Chat_History*: Mide la capacidad del modelo para mantener el contexto de la conversación a lo largo de las interacciones.
 - *Returns_No_Information*: Verifica que no introduzca información nueva que no se haya mencionado previamente en la conversación.
- *RAG*

Como se explicó en la sección de Diseño y Arquitectura, la componente *RAG* se divide en varias subcomponentes, cada una con métricas específicas para evaluar su desempeño.

 - *Query Analyzer LLM*
 - *Resolves_References*: Evalúa la capacidad del modelo para resolver referencias a mensajes anteriores al reformular la consulta del usuario y generar subconsultas para la búsqueda en la base de datos. Un ejemplo de conversación en la que el modelo resuelve una referencia implícita a los modelos de lenguaje es el siguiente:

AI: “Los modelos de lenguaje de gran tamaño, también conocidos como LLMs, son capaces de comprender y generar texto en lenguaje natural...”

Usuario: “¿Podrían llegar a tener sesgo de información?”

En este caso, la consulta actualizada podría ser:

“¿Los modelos de lenguaje podrían llegar a tener sesgo de información?”

De esta manera, el modelo resuelve el sujeto implícito de la pregunta del usuario. Del mismo modo, las subconsultas generadas harían referencia explícita a los modelos de lenguaje, permitiendo una recuperación de información más efectiva.

- *Expands_Acronyms:* Verifica que las subconsultas incluyan tanto el acrónimo como su forma expandida.
- *Includes_Context:* Confirma que el modelo incorpore información relevante del historial al actualizar la consulta del usuario y generar subconsultas. A modo de ejemplo, se considera la siguiente conversación:

Usuario: He leído que los modelos de lenguaje pueden producir información incorrecta e inventar contenido al generar texto.

AI: Es cierto, algunos modelos de lenguaje pueden generar información imprecisa o sesgada, lo cual puede deberse a varios factores relacionados con los datos, el diseño del modelo y su entrenamiento.

Usuario: Entonces, ¿deberíamos desconfiar de la inteligencia artificial?

En este caso, si el modelo no incorpora el contexto previo, la consulta reformulada podría quedar simplemente como:

“¿Deberíamos desconfiar de la inteligencia artificial?”

Sin embargo, para asegurar que la consulta incluya el contexto relevante, la versión actualizada debería ser:

“¿Deberíamos desconfiar de la inteligencia artificial debido a los riesgos de producir información incorrecta e inventar contenido al generar texto?”

De esta manera, el modelo preserva el significado original de la conversación y permite recuperar documentos que aborden específicamente la confiabilidad de la IA en relación con la generación de contenido inexacto. Las subconsultas generadas también deberían reflejar este contexto.

- *Retriever*

Las métricas utilizadas para evaluar esta componente son *Context Relevancy* y *Context Recall*, corresponden a algunas de las métricas comunes en la evaluación de sistemas RAG descritas en la sección [2.3.5](#). Para una descripción detallada, consultar dicha sección.

- *Response Generator LLM*

Para la evaluación de esta componente se utilizaron tanto métricas comunes en la evaluación de sistemas RAG: *Answer Relevancy*, *Answer Correctness* y *Faithfulness*; como métricas específicas diseñadas para este proyecto. A continuación, se describen estas últimas, mientras que las métricas comunes pueden consultarse en la sección 2.3.5.

- *Acknowledges_Contradiction*: Evalúa si la respuesta reconoce la existencia de contradicciones dentro de los fragmentos recuperados y si menciona explícitamente cuando hay información conflictiva, contrastando las ideas de las fuentes en desacuerdo. Esta métrica es relevante porque la base de documentos podría contener información contradictoria, y en esos casos, el comportamiento esperado no es, por ejemplo, que el modelo seleccione arbitrariamente una de las posturas basándose en lo aprendido previamente, sino que reconozca y explice la contradicción, ofreciendo una respuesta que muestre la diversidad de perspectivas en los documentos.
- *Citations_Real_and_Used*: Dado que la componente *Response Generator LLM* devuelve los fragmentos utilizados para generar la respuesta junto con sus respectivas fuentes, es crucial que los fragmentos devueltos sean efectivamente los que fueron utilizados. Esta métrica evalúa tanto que los fragmentos estén presentes en el conjunto de fragmentos proporcionados (sin ser inventados), como que realmente hayan sido utilizados para generar la respuesta (y no se devuelvan fragmentos que no fueron utilizados).

- *Informative Conversational LLM*

- *RAG_Response_Unaltered*: Verifica que la respuesta se base exclusivamente en la información proporcionada por el RAG, sin agregar, modificar ni inferir contenido adicional.
- *Adapts_to_User_Profile*: Evalúa si la complejidad y forma de las respuestas se ajustan al nivel de experiencia previa del usuario. Esto garantiza el cumplimiento del requerimiento funcional de interacción adaptable al perfil del usuario (RF-5).
- *Is_Conversational*: Evalúa si el chatbot mantiene una interacción natural, evitando respuestas excesivamente largas y presentando la información de manera progresiva, ampliándola a medida que el usuario lo solicita.

- *Pedagogical LLM*

- *Returns_no_information*: Verifica que el modelo no proporcione información en su respuesta y que evite responder la pregunta del usuario.

- *Prompts_Reasoning*: Evalúa si la repregunta generada por el modelo es pertinente con respecto a la consulta inicial del usuario y si fomenta el razonamiento y la reflexión. Se espera que la repregunta guíe al usuario en la construcción del conocimiento en lugar de desviarlo del tema o resultar irrelevante.

- *Denial LLM*

- *Returns_no_information*: Verifica que el modelo no proporcione información en su respuesta y que rechace las consultas por ser ajenas al ámbito de la inteligencia artificial.

Preparación de *datasets*

Se diseñan conjuntos de pruebas que incluyan mensajes representativos de los diferentes escenarios contemplados por los requerimientos, de modo que sea posible evaluar cada componente de forma focalizada. Por ejemplo, para la evaluación del *Router LLM*, se incluyen instancias relacionadas con temas fuera del ámbito de la IA, preguntas que requieran recuperación de información, consultas donde se deba aplicar una estrategia pedagógica, etc., con el fin de validar que en cada caso se seleccione el camino esperado.

Evaluación de métricas con LLMs como jueces

Dado que la mayoría de las métricas para evaluar textos arbitrarios generados por LLMs carecen de un único *gold standard* de referencia, fue necesario emplear modelos de lenguaje que actúen como jueces (véase 2.3.5). En este enfoque, para cada prueba se utiliza un LLM que compara la respuesta obtenida contra la respuesta o comportamiento esperado y retorna una salida estructurada. Esta salida incluye, por un lado, un campo booleano que indica si la respuesta cumple o no con el criterio de evaluación, y por otro, un listado de pasos de razonamiento que justifican la decisión del modelo. Estos pasos aportan trazabilidad y permiten ajustar el *prompt* del evaluador en caso de detectar inconsistencias.

Este enfoque no solo permite automatizar la verificación de métricas como *Answer_Correctness*, *Faithfulness* o *Remembers_Chat_History*, sino que también facilita el entendimiento del proceso de decisión del LLM evaluador y la mejora continua de las indicaciones de *prompting*. Como excepción, en la métrica *Routes_Correctly* se cuenta con un *gold standard* preciso que permite evaluar el comportamiento sin la intervención de un LLM. En este caso, cada instancia (historial de conversación) tiene un camino esperado definido en función del último mensaje del usuario, lo que posibilita una evaluación determinista.

Para llevar a cabo la evaluación completa, se ejecuta un *script* que recorre todas las instancias del conjunto de prueba y aplica las métricas correspondientes a cada componente.

Un ejemplo de *prompt* generado para estas evaluaciones puede consultarse en el anexo B (correspondiente a la métrica *Acknowledges Contradictions*), aunque

todos los *prompts* se encuentran disponibles en el repositorio de GitHub del proyecto.

Registro de resultados en archivo JSON

Cada ejecución del *script* almacena los resultados en un archivo estructurado que incluye la fecha de la prueba, los valores globales de cada métrica (promedios sobre todas las instancias) y el detalle de la evaluación de cada instancia individual. Para cada una se registra la etiqueta de cumplimiento junto con los pasos de razonamiento correspondientes, lo cual ofrece un historial detallado de la evolución del sistema y facilita la identificación de oportunidades de mejora.

4.1.3. Aspectos no implementados y trabajos futuros

Debido a limitaciones de tiempo y al alcance definido para el proyecto, no fue posible implementar todas las métricas automatizadas previstas para la evaluación *offline*. En particular, se priorizó la implementación de métricas para el *Router LLM* y para las componentes del *RAG*, implicadas en el camino de recuperación de información, por considerarse componentes centrales en el funcionamiento del sistema y especialmente críticas en lo que refiere a la fidelidad de las respuestas generadas.

La elección de este enfoque respondió, por un lado, a la importancia de asegurar que el sistema seleccione siempre el camino adecuado según las especificaciones definidas en el diseño. El correcto funcionamiento del *Router LLM* es clave para garantizar que el sistema actúe conforme al comportamiento esperado: que recupere información cuando corresponde, que aplique una estrategia pedagógica ante una consulta que lo justifique, que deniegue una respuesta cuando el tema está fuera del dominio definido, y que responda de manera apropiada ante mensajes casuales. Esto resulta esencial para el buen funcionamiento general del chatbot, ya que, por ejemplo, una repregunta generada por el camino pedagógico solo debería formularse cuando realmente pueda enriquecer el proceso de aprendizaje, y no de forma arbitraria, ya que eso podría frustrar al usuario al recibir muchas preguntas y pocas respuestas. De igual manera, ante una consulta fuera del dominio, no sería deseable que el sistema intentara recuperar información o diera una respuesta basada en la inteligencia del propio modelo, sino que debería denegar adecuadamente. En el caso de un saludo o un comentario informal, tampoco tendría sentido procesar la entrada como si requiriera búsqueda de información o aplicar estrategias pedagógicas, ya que eso implicaría una pérdida de tiempo en recuperación y procesamiento, además de devolver una respuesta inadecuada.

Por otro lado, dado que una de las principales fortalezas del chatbot desarrollado radica en su capacidad de proporcionar respuestas basadas en documentos curados y de devolver la fuente de donde proviene la información, también se consideró fundamental evaluar la trayectoria de recuperación de información llevada a cabo por la componente *RAG*. Esta evaluación resulta crítica para asegurar que las respuestas generadas se apoyen efectivamente en los fragmen-

tos recuperados de la base, sin modificarlos ni introducir información que no esté respaldada por esos documentos. En este sentido, se definieron métricas específicas para evaluar distintos aspectos clave del funcionamiento de esta trayectoria: la calidad de la información almacenada en la base, la relevancia de los fragmentos recuperados, el uso real y fiel de dichos fragmentos en la respuesta final, y la correspondencia entre los fragmentos citados y los efectivamente utilizados. Asegurar un comportamiento adecuado en estas dimensiones es esencial para garantizar la confiabilidad del sistema y cumplir con uno de los requerimientos centrales del proyecto: generar respuestas exclusivamente a partir de información presente en el repositorio de documentos curados (RF-2).

Si bien la evaluación automatizada cubrió los aspectos más críticos para garantizar la fidelidad y confiabilidad del sistema, como se mencionó anteriormente, algunas componentes quedaron fuera de este proceso, lo que representa una oportunidad clara para futuras mejoras. En particular, no se llegaron a implementar las métricas automatizadas para los restantes caminos que puede tomar el *Router LLM*, incluyendo las componentes *Casual LLM*, *Denial LLM* y *Pedagogical LLM*. Tampoco se desarrollaron las métricas definidas para el *Informative Conversational LLM*, encargado de tomar la respuesta final del *RAG* y convertirla en un formato más conversacional y dialogado. Estos caminos fueron evaluados mediante pruebas manuales *online*, generando conversaciones que llevarán por cada uno de estos caminos y analizando las trazas correspondientes, pero sin contar con un mecanismo automatizado que verificara su comportamiento de forma sistemática. La implementación de estas métricas se plantea como una línea de trabajo futura en la sección 6.2, que permitiría ampliar la cobertura de la evaluación *offline* y profundizar el análisis sobre la calidad de las respuestas producidas por estas componentes.

4.1.4. Resultados

A continuación se presentan los resultados obtenidos en la evaluación *offline* de las distintas componentes del sistema, utilizando las métricas definidas para cada una. Estos resultados permiten analizar el desempeño individual de cada componente y orientar posibles mejoras.

- *Router LLM*

La métrica de *Routing Accuracy* alcanzó un valor final de **0.99** sobre un total de 81 ejemplos evaluados, los cuales abarcan los distintos caminos posibles, con historiales de conversación variados tanto en contenido como en longitud. Este resultado se obtuvo tras un proceso iterativo de ajuste del *prompt* del *Router LLM*, en el cual se analizaron los errores cometidos por el modelo en cada uno de los caminos y se refinaron las instrucciones en consecuencia. Asimismo, se observó una mejora significativa en el desempeño al reemplazar el modelo GPT-4o mini por GPT-4o, impactando positivamente en la capacidad del sistema para seleccionar el camino adecuado. Dos estrategias que resultaron particularmente efectivas fueron el

uso de la técnica *chain of thought* para fomentar el razonamiento explícito del modelo (Wei y cols., 2022), y la incorporación de ejemplos representativos a través de *few-shot prompting* (Brown y cols., 2020)..

■ *Response Generator LLM*

Para la evaluación de esta componente se utilizaron 10 ejemplos diferentes para cada métrica. El resultado para *Answer Relevancy* fue de **1.0**, lo que indica que en todos los casos evaluados las respuestas generadas fueron consideradas relevantes en relación con las preguntas planteadas. Este resultado refleja una adecuada comprensión de las consultas por parte del modelo y un uso efectivo del contexto proporcionado para generar respuestas pertinentes y alineadas con la intención del usuario. Cabe señalar que no todos los fragmentos incluidos en los contextos de los ejemplos son útiles para responder las preguntas, lo cual podría ocurrir en un escenario real, en el que la base de datos no necesariamente contiene información que responde la consulta, o en el que se recuperan fragmentos poco relevantes. Incluir este tipo de casos en la evaluación permite validar la capacidad del modelo para identificar y utilizar únicamente la información pertinente dentro de contextos más amplios o con ruido.

Por otro lado, la métrica *Answer Correctness* obtuvo un valor de **0.9**, lo que indica que en la mayoría de los casos las respuestas generadas fueron correctas en relación con la respuesta esperada (*ground truth*). Nuevamente los ejemplos utilizados incluyeron tanto contextos con fragmentos relevantes como irrelevantes para la consulta, y en este caso también ejemplos en los que ningún fragmento era útil para responder adecuadamente. En estos últimos, la respuesta esperada por parte del *Response Generator LLM* era “*No information found*”, evaluando así su capacidad para reconocer la ausencia de información pertinente y evitar generar respuestas incorrectas. Este enfoque permitió validar no solo la precisión del modelo en la generación de respuestas, sino también su habilidad para gestionar adecuadamente los casos en que la información proporcionada no es útil.

Para la métrica *Faithfulness*, se obtuvo un valor de **1.0**, lo que indica que en todos los casos evaluados las respuestas generadas se derivaron del contexto proporcionado. Este comportamiento es clave para asegurar la confiabilidad del sistema, ya que garantiza que las respuestas estén efectivamente respaldadas por las fuentes recuperadas.

Para la métrica *Acknowledges Contradiction*, se obtuvo un valor de **0.9**, lo que refleja una buena capacidad del modelo para contrastar diferentes opiniones e informaciones presentes en los fragmentos recuperados, resolviendo correctamente los casos en los que hay contradicciones. Este aspecto es relevante ya que no sería adecuado, por ejemplo, que el modelo descartara automáticamente fragmentos por considerarlos incorrectos según su conocimiento previo; por el contrario, debe ser capaz de reflejar la diversidad

de perspectivas presentes en el contexto. En esta evaluación se utilizaron ejemplos diseñados con fragmentos intencionalmente contradictorios.

Por último, la métrica *Citations Real and Used* alcanzó un valor de **1.0**, lo que confirma que en todos los casos evaluados se citaron exclusivamente los fragmentos que fueron efectivamente utilizados para generar la respuesta. Este aspecto es clave para garantizar la confiabilidad del sistema en cuanto a la atribución de fuentes, ya que citar referencias no utilizadas u omitir aquellas que sí fueron utilizadas comprometería la integridad académica y el respeto por la propiedad intelectual. En los ejemplos para esta métrica, los contextos incluyen la metadata necesaria para que el *Response Generator LLM* pueda devolver correctamente las citas.

■ *Retriever*

Las métricas utilizadas para evaluar esta componente son *Context Relevancy* y *Context Recall*. La evaluación se realizó sobre un conjunto de 50 ejemplos con preguntas vinculadas a la inteligencia artificial, su aplicación en la educación, su uso en distintos ámbitos, aspectos éticos, entre otros temas que podrían ser de interés para un docente y que se espera que estén cubiertos por la base de datos del sistema.

La métrica *Context Relevancy*, que mide la proporción de fragmentos relevantes para responder la consulta entre el total de fragmentos recuperados, obtuvo un valor de **0.58**. Este resultado refleja que, en promedio, el 58 % del contenido recuperado fue pertinente para responder las preguntas planteadas.

Por otro lado, para la métrica *Context Recall* —que evalúa en qué medida los fragmentos recuperados cubren las distintas afirmaciones que están presentes en la respuesta de referencia— se le solicitó al LLM evaluador que clasificara el grado de cubrimiento de cada afirmación como “total”, “parcial” o “nulo”, en lugar de limitarse a una respuesta binaria (sí/no). A partir de estas clasificaciones, se calcularon dos variantes de la métrica: una que considera una afirmación cubierta si fue marcada como “total” o “parcial”, y otra ponderada (*Weighted Context Recall*), en la que las afirmaciones totalmente cubiertas aportan 1 punto, las parcialmente cubiertas 0.5 y las no cubiertas 0. El resultado para la primera variante fue de **0.73**, mientras que el valor de la métrica ponderada fue de **0.47**.

Los resultados obtenidos para estas métricas refuerzan la importancia de seguir optimizando el proceso de recuperación. Algunas posibles líneas de mejora incluyen la experimentación con diferentes criterios o algoritmos de búsqueda, con el objetivo de mejorar la calidad de los fragmentos recuperados; la incorporación de más documentos a la base de datos, para aumentar la cobertura temática; y el ajuste de parámetros como la cantidad de fragmentos devueltos o los hiperparámetros específicos del algoritmo de búsqueda, entre otros. Estas acciones podrían contribuir a incremen-

tar la relevancia y cobertura del contexto recuperado, mejorando así el desempeño general del sistema.

■ *Query Analyzer LLM*

Para las tres métricas utilizadas en la evaluación de esta componente —*Resolves References*, *Expands Acronyms* e *Includes Context*—, se utilizaron 10 ejemplos diseñados específicamente para poner a prueba cada una de ellas. Estos ejemplos incluyen la consulta del usuario y, cuando corresponde, un historial de conversación, las subconsultas esperadas para la búsqueda en la base de datos y la consulta actualizada correspondiente.

La métrica *Resolves References* obtuvo un valor de **1.0**, evidenciando una sólida capacidad del modelo para resolver referencias a mensajes anteriores al reformular la consulta del usuario y generar las subconsultas correspondientes. Este aspecto es fundamental para garantizar una recuperación adecuada en un sistema conversacional, ya que, de no resolverse correctamente las referencias, podrían generarse consultas ambiguas o incompletas; por ejemplo, consultas sin un sujeto explícito, que afectarían negativamente la calidad de las respuestas. En este caso, fue suficiente con indicarle al modelo explícitamente en el *prompt* que resolviera las referencias para que lo lograra y con este excelente desempeño. Para esta métrica, los ejemplos incluyen la consulta original del usuario, que hace referencia a algo mencionado anteriormente en la conversación, y el historial de conversación necesario para permitir la resolución.

Por otro lado, la métrica *Expands Acronyms* obtuvo un resultado final de **0.6**, lo que indica que en el 60 % de los casos las subconsultas generadas incluyen tanto el acrónimo como su forma expandida. En esta evaluación, los ejemplos contienen únicamente la consulta original del usuario con al menos un acrónimo, pero no incluyen un historial de conversación. Esto puede haber afectado el desempeño del modelo, ya que se en el *prompt* se le indica expandir un acrónimo cuando le es familiar o su significado puede ser inferido del contexto de la conversación y del tema tratado. Para mejorar esta métrica y tener ejemplos más representativos del uso real del chatbot, podría ampliarse el conjunto de ejemplos incorporando historiales de conversación, de manera que el modelo disponga del contexto necesario para resolver los acrónimos con mayor precisión.

Por último, para la métrica *Includes Context* se obtuvo un valor de **1.0**, lo que confirma que el modelo incorpora efectivamente información relevante del historial al actualizar la consulta del usuario y generar las subconsultas correspondientes. Nuevamente, este aspecto resulta fundamental en un sistema conversacional como el de este proyecto, ya que permite recuperar información en base a consultas completas, así como actualizar la consulta del usuario teniendo en cuenta el contexto de la conversación, de manera que el *Response Generator LLM* cuente con toda la información necesaria para formular la respuesta. En este caso, también fue suficiente con

indicarle al modelo en el *prompt* que debía incorporar el contexto y proporcionarle algunos ejemplos mediante la técnica de *Few Shot* para lograr este buen desempeño.

En términos generales, los resultados obtenidos son alentadores y reflejan un funcionamiento adecuado del sistema en sus distintas componentes. El *Router LLM* se comporta de acuerdo a lo esperado, mostrando una capacidad sólida para redirigir las consultas según corresponda. Tanto el *Response Generator LLM* como el *Query Analyzer LLM* presentan un muy buen desempeño en los ejemplos evaluados, aunque el número reducido de casos (10 por métrica) limita el alcance de las conclusiones que pueden obtenerse. Sería recomendable ampliar la cantidad y variedad de ejemplos para contar con una evaluación más robusta. Por su parte, la componente *Retriever* mostró resultados aceptables, pero con margen para mejoras. Dada su influencia directa en la calidad de las respuestas generadas, y su importancia dentro del sistema, sería bueno continuar trabajando en su optimización mediante la incorporación de nuevos documentos, la exploración de diferentes algoritmos de búsqueda y el ajuste de parámetros relevantes. En conjunto, estos resultados proporcionan una buena base sobre la cual seguir iterando y fortaleciendo el sistema.

4.2. Evaluación *Online*

En las evaluaciones *online*, se examina de manera manual el rendimiento y la corrección del sistema en su entorno de ejecución real. Para eso se analizan los rastros de ejecución (*traces*) e históricos de conversaciones generados durante las interacciones con el chatbot, los cuales pueden visualizarse a través de las herramientas de monitoreo que ofrece *LangSmith*.

Esta evaluación es realizada tanto por el equipo de desarrollo, que genera escenarios específicos para observar comportamientos particulares del sistema, como por las usuarias expertas del proyecto, quienes interactúan libremente con el chatbot y aportan casos de uso reales desde su propia perspectiva. Para ello, se incorporó una funcionalidad que permite dejar comentarios directamente desde la interfaz de la aplicación. Esta opción se habilita a través de un botón que despliega un modal con un campo de texto para ingresar retroalimentación. Los comentarios se almacenan en *Firebase* junto con un conjunto de metadatos que incluyen: la versión de la aplicación en la que se generó el comentario, la dirección de correo electrónico del usuario (utilizada para identificar a quien dejó el comentario), la fecha y el identificador de la conversación (*thread ID*). Este último permite vincular el comentario con su conversación correspondiente en *LangSmith*, facilitando el análisis contextual de los casos reportados.

En la figura 4.1 se muestra cómo se visualiza el modal de retroalimentación al hacer click en el botón ubicado en la esquina superior derecha de la interfaz.

Las evaluaciones *online* se realizan de forma continua a lo largo del proceso de desarrollo, especialmente tras la incorporación de nuevas funcionalidades o ajustes en el sistema. Son clave para observar el comportamiento del sistema

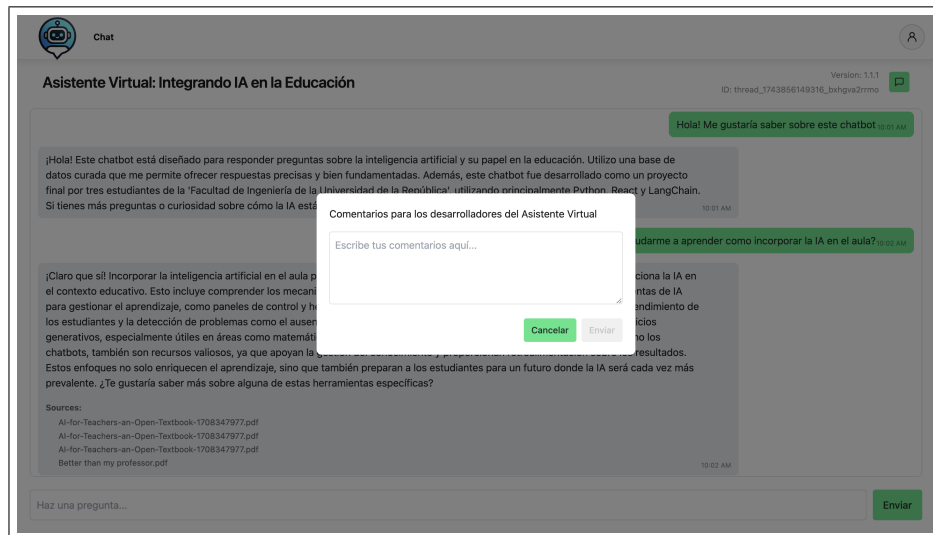


Figura 4.1: Modal para dejar retroalimentación a los desarrolladores.

en condiciones reales de uso, donde las personas pueden enviar mensajes ambiguos, con errores tipográficos o gramaticales, o formular consultas de manera poco clara; así como generar escenarios inesperados que no siempre están contemplados en las pruebas *offline*. Analizar estas situaciones permite detectar fallos y ajustar el sistema para mejorar su robustez y capacidad de respuesta frente a los diversos escenarios que podrían presentarse.

4.2.1. Objetivos de la Evaluación *Online*

- **Validar el comportamiento integral del sistema:** Este tipo de evaluación permite comprobar la efectividad de la orquestación de todas las componentes, incluyendo factores como la latencia, el manejo de errores y la interacción con servicios externos, como la API de los LLMs o la base de datos.
- **Detección de problemas no previstos:** Algunas fallas emergen únicamente ante consultas atípicas o usos inesperados por parte de usuarios reales. La evaluación *online* permite detectar estos casos a través del análisis de trazas en producción, así como incorporar la perspectiva de las usuarias expertas del proyecto, quienes probablemente realicen pruebas distintas a las contempladas por el equipo de desarrollo.

4.2.2. Mecánica de la Evaluación *Online*

A continuación se describen las acciones concretas que conforman la evaluación *online*, incluyendo el diseño de consultas y escenarios de prueba por parte

del equipo de desarrollo, la interacción libre realizada por las usuarias expertas del proyecto, y el análisis posterior de los rastros de ejecución y comentarios obtenidos.

Diseño de consultas y escenarios de pruebas

Como se mencionó previamente, la evaluación *online* incluye tanto las pruebas realizadas por el equipo de desarrollo como aquellas llevadas a cabo por las usuarias expertas del proyecto. En el caso de las usuarias, la interacción con el sistema es completamente libre, lo que les permite explorar el chatbot desde su propia perspectiva, formulando las consultas que consideren más pertinentes para evaluar su utilidad, claridad y relevancia. Este enfoque aporta una mirada complementaria, al generar escenarios de uso reales que pueden diferir de los previstos por el equipo de desarrollo.

Por su parte, el equipo de desarrollo diseña un conjunto amplio y variado de consultas que permitan observar el comportamiento del sistema en condiciones reales y variadas, contemplando tanto preguntas típicas como atípicas. Las preguntas típicas pueden incluir, por ejemplo, aquellas que contienen subconsultas que deberían ser correctamente identificadas y separadas por el *Query Analyzer* para recuperar información de la base de datos, o consultas con acrónimos, donde se espera que el sistema considere también su forma expandida durante la recuperación. En cambio, las preguntas atípicas pueden presentar contenido contradictorio, formulaciones ambiguas, errores, mensajes desordenados o sin una intención claramente definida. Este conjunto diverso de consultas permite monitorear qué tan bien responde el sistema y su capacidad para adaptarse a distintos contextos de uso.

Análisis de *traces* y retroalimentación recibida

El sistema guarda los rastros de cada interacción usando *LangSmith*, que resulta una herramienta valiosa ya que facilita la detección de errores y el proceso de depuración (*debugging*) al mostrar en detalle todos los pasos involucrados en cada interacción. Entre estos pasos se incluyen las funciones ejecutadas con sus respectivas entradas y salidas, el camino seleccionado por el *Router LLM*, la decisión del modelo que motivó esa elección, las subconsultas generadas para la recuperación de información, los fragmentos recuperados de la base y los que fueron efectivamente utilizados en la respuesta, el tiempo de respuesta total y por componente, entre otras cosas. La figura 4.2 ilustra cómo se ve una traza en *LangSmith*.

Este análisis técnico se complementa con la revisión de los comentarios enviados por las usuarias expertas a través del modal de retroalimentación integrado en la aplicación. Al estar cada comentario vinculado a su conversación correspondiente, es posible interpretar con mayor precisión la situación reportada, ya sea un error técnico, una limitación funcional o una dificultad en la interacción. Este tipo de retroalimentación sirve como insumo para orientar mejoras concretas. De esta manera, el equipo de desarrollo puede iterar sobre el sistema

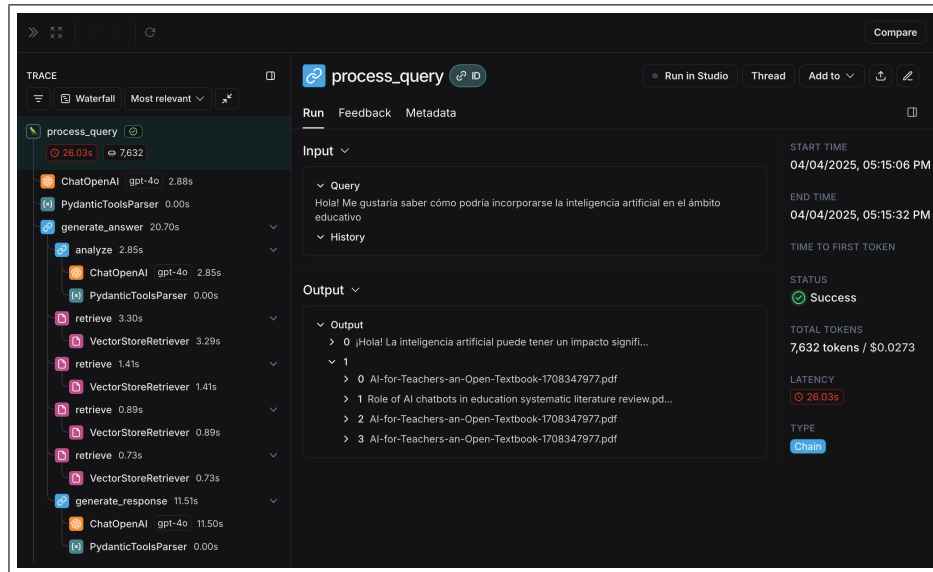


Figura 4.2: Ejemplo de traza de *Langsmith*.

incorporando la perspectiva de las usuarias y enriquecer así la evaluación técnica con experiencias reales de uso.

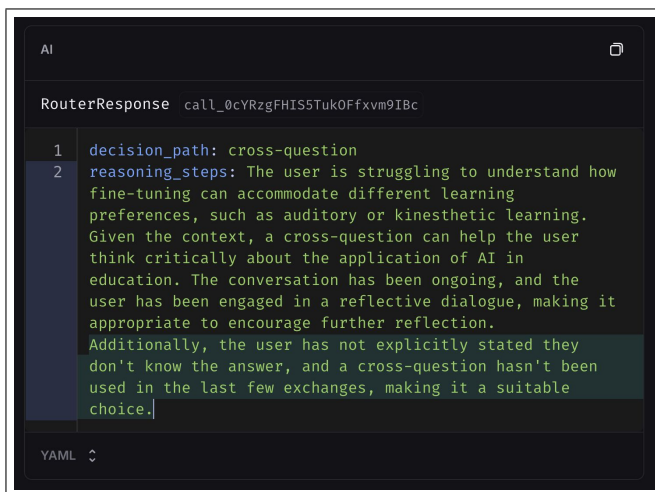
4.2.3. Resultados

El análisis de las evaluaciones *online*, sustentado en las trazas de LangSmith, resultó decisivo para depurar y optimizar la solución desarrollada.

Uno de los aspectos evaluados en esta etapa fue la frecuencia de repreguntas durante una conversación promedio. Se espera que estas contribuyan a fomentar la reflexión sin generar molestia ni perjudicar la experiencia de aprendizaje del usuario. Por más que la evaluación *offline* sugiera que el *Router LLM* toma buenas decisiones respecto a cuándo aplicar esta estrategia pedagógica, resulta fundamental validar este comportamiento en conversaciones reales, más extensas y variadas. Los casos utilizados en la evaluación *offline* suelen contener historiales breves con mensajes simples, lo cual permite tener un panorama del rendimiento, pero algunas veces puede diferir del comportamiento observado en escenarios reales.

En este análisis se detectaron situaciones donde se realizaban repreguntas en exceso y de forma consecutiva, lo cual no se considera adecuado. Al examinar las trazas de LangSmith, particularmente los pasos de razonamiento del *Router LLM*, se identificó que el modelo no interpretaba correctamente la instrucción de evitar repreguntar si ya se había hecho recientemente. En su razonamiento, el modelo concluía erróneamente que no se había hecho ninguna repregunta previa, por lo que volvía a elegir ese camino. La Figura 4.3 muestra un ejemplo concreto

de este comportamiento, observado en las trazas de ejecución. Para corregirlo, se llevó a cabo un proceso iterativo de refinamiento sobre el *prompt* del *Router*, hasta lograr que su comportamiento se alinea con el esperado.



```
AI RouterResponse call_0cYRzgFHIS5Tuk0FFxvm9IBc

1 decision_path: cross-question
2 reasoning_steps: The user is struggling to understand how fine-tuning can accommodate different learning preferences, such as auditory or kinesthetic learning. Given the context, a cross-question can help the user think critically about the application of AI in education. The conversation has been ongoing, and the user has been engaged in a reflective dialogue, making it appropriate to encourage further reflection. Additionally, the user has not explicitly stated they don't know the answer, and a cross-question hasn't been used in the last few exchanges, making it a suitable choice.

YAML ↺
```

Figura 4.3: Razonamiento del *Router LLM* observado en LangSmith. En este ejemplo, se observa que el modelo no detecta correctamente una repregunta reciente, lo que lleva a una decisión incorrecta.

A su vez, fue fundamental comprobar la calidad de las repreguntas realizadas. No basta con que el modelo decida hacer una repregunta en el momento adecuado; también es necesario que el contenido de esa repregunta sea pertinente para el contexto de la conversación. En este sentido, se pudo confirmar que las repreguntas generadas cumplían con este criterio.

Otro caso evaluado fue el comportamiento del chatbot frente a afirmaciones erróneas por parte del usuario, como por ejemplo: “el aprendizaje automático siempre hace predicciones perfectas”. Se observó que, ante este tipo de afirmaciones, el sistema respondía cuestionando al usuario y orientándolo a considerar por qué dicha afirmación no es correcta. Se probaron múltiples ejemplos, y en todos los casos el comportamiento fue consistente, alineándose con lo esperado y validando el funcionamiento del sistema en este tipo de situaciones. La Figura 4.4 muestra la respuesta generada por el chatbot ante dicha afirmación.

También se evaluó la respuesta del chatbot cuando un usuario se identifica como estudiante. Aunque los modelos fueron configurados con un *prompt* de sistema que indica que el asistente está diseñado principalmente para ayudar a profesores y educadores, se espera que no rechace responder a estudiantes siempre que las preguntas estén vinculadas con la temática de la inteligencia artificial. Se observó que, en estos casos, el chatbot responde de forma adecuada, ofreciendo apoyo también a estudiantes interesados en la IA. A continuación, se presenta un ejemplo de respuesta obtenida:

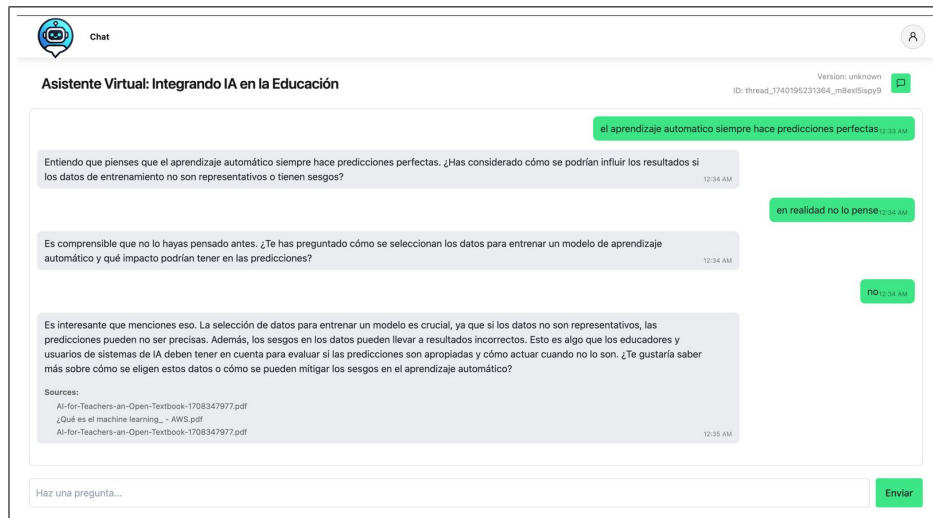


Figura 4.4: Respuesta del chatbot ante una afirmación incorrecta.

“Estoy aquí principalmente para ayudar a profesores y educadores interesados en cómo la inteligencia artificial puede integrarse en el ámbito educativo. Sin embargo, si eres estudiante y tienes curiosidad sobre el tema, estaré encantado de conversar contigo y explorar juntos cómo la IA puede impactar en el aprendizaje. ¿Tienes alguna pregunta específica sobre la IA en educación?”

Este comportamiento es el esperado, ya que el sistema dispone de los recursos necesarios para atender consultas sobre esta temática, independientemente del rol del usuario, brindando soporte tanto a docentes como a estudiantes interesados en IA.

Las pruebas mencionadas fueron solo una parte del conjunto de evaluaciones realizadas. También se llevaron a cabo otras pruebas, como la verificación del estilo de las respuestas que deniegan la conversación por estar fuera de la temática de la IA, las cuales se espera que sean amables, claras y que sugieran recursos alternativos. Asimismo, se revisó el cumplimiento de las instrucciones indicadas en los *prompts*, observándose que, en algunas situaciones, ciertas indicaciones eran omitidas por los modelos. Como resultado del análisis realizado, se realizaron diversos ajustes, incluyendo la refinación de los *prompts* de varios modelos, tanto en las instrucciones como en los ejemplos de *few-shot*, los cuales demostraron tener una influencia significativa en la salida del modelo.

Finalmente, la retroalimentación directa de las usuarias expertas, a través del botón de comentarios en el chatbot, puso en evidencia un problema presente en una implementación intermedia del *Router* (correspondiente a la segunda arquitectura desarrollada, descrita en la sección 5.1.2). En dicha versión, la selección del camino a seguir se realizaba de forma aleatoria, lo que conducía con

frecuencia a respuestas inadecuadas. Este comportamiento generó frustración en usuarios reales y confirmó la necesidad de contar con un mecanismo más inteligente. En respuesta, se reemplazó la lógica aleatoria por el *Router LLM* de la arquitectura actual.

Además, los comentarios de las usuarias dieron lugar a varias ideas para enriquecer el prototipo, las cuales fueron documentadas en la sección de trabajos futuros, junto con otras propuestas que quedaron fuera del alcance inicial del proyecto.

En conjunto, las pruebas internas y los comentarios de las usuarias expertas no solo permitieron validar la solidez de la arquitectura general, sino que también aportaron información valiosa para ajustar los *prompts*, afinar los comportamientos de los modelos y perfeccionar la orquestación general del sistema, asegurando que el prototipo mantenga estándares adecuados de calidad y usabilidad en entornos reales.

4.3. Evaluación con Usuarios

Finalmente, se contempló una instancia de evaluación del sistema con usuarios reales, la cual se realizó al cierre del proceso de desarrollo, utilizando la versión final del prototipo. Su propósito es comprobar que el sistema cumpla no solo la corrección técnica y formal, tal como fue abordado en las evaluaciones anteriores mediante métricas específicas y pruebas controladas, sino que también satisfaga la experiencia de uso y los requerimientos acordados desde la perspectiva de quienes lo utilizan. Este tipo de evaluación es clave para recoger impresiones, observaciones y sugerencias por parte de personas externas al equipo de desarrollo, lo que permite detectar aspectos de usabilidad, comprensión o interacción que podrían pasar desapercibidos en las evaluaciones *online* realizadas por el propio equipo.

Dado que esta instancia requiere la participación de múltiples usuarios y tiene como objetivo principal documentar los resultados finales del sistema, se lleva a cabo únicamente al finalizar el desarrollo, utilizando la última versión del prototipo.

4.3.1. Objetivos de la Evaluación con Usuarios

- **Validar el grado de cumplimiento de los requerimientos:** Confirmar, desde la perspectiva de los usuarios, si el sistema cumple con los objetivos planteados inicialmente y responde adecuadamente a las necesidades para las cuales fue diseñado.
- **Obtener una impresión final sobre el nivel de satisfacción de los usuarios:** Recoger opiniones sobre la experiencia general de uso del sistema, incluyendo su utilidad, facilidad de interacción y claridad de las respuestas, con el fin de contar con una evaluación externa y directa del prototipo desarrollado.

4.3.2. Enfoque Propuesto

Se planteó un enfoque que combina pruebas guiadas y de uso libre, con el fin de obtener tanto información detallada sobre el cumplimiento de los requerimientos como una visión general de la experiencia de los usuarios al interactuar con el sistema.

- **Pruebas guiadas con un conjunto de preguntas:** Se solicita a un grupo de usuarios que interactúe con el chatbot siguiendo una guía de preguntas predefinidas, diseñadas para evaluar diferentes áreas de funcionalidad (respuesta basada en información, aplicación de estrategias pedagógicas, adaptación al nivel de conocimiento del usuario, entre otras). Luego, se les solicita completar un formulario donde valoran en qué medida consideran que el sistema cumple con los requerimientos establecidos.
- **Pruebas de uso libre:** Los usuarios interactúan libremente con el chatbot, formulando las consultas que consideren relevantes para evaluar la utilidad general del sistema. Esto permite detectar eventuales limitaciones o comportamientos inesperados que no estaban contemplados en la evaluación guiada. También brinda información valiosa sobre la percepción general de los usuarios en términos de usabilidad, utilidad y accesibilidad.

4.3.3. Rúbrica de Evaluación

A continuación se presenta la rúbrica concreta diseñada para la evaluación. Esta se organiza en bloques, cada uno de los cuales aborda un requerimiento específico e incluye una descripción detallada, junto con las instrucciones a seguir para evaluarlo.

Para cuantificar las respuestas, se emplea una escala de 1 (muy en desacuerdo) a 5 (muy de acuerdo), seguida de una sección de comentarios destinada a recoger una perspectiva cualitativa. También se solicita el ID de la conversación (visible en la interfaz del chatbot) para tener trazabilidad.

Cabe señalar que el bloque 4 evalúa dos de los requerimientos de forma conjunta, y que el bloque 5 no está asociado a un requerimiento específico, sino que busca recoger una valoración global del sistema.

1. **RF-1. Interacción acotada a temas de inteligencia artificial**

- **Descripción:** El chatbot limita sus respuestas a temas relacionados con IA y redirige las consultas sobre otros tópicos.
 - **Instrucciones:** Mantenga una conversación; pregunte primero sobre un tema ajeno a la IA y luego sobre uno relacionado con la IA.
- 1.1. Identificador (ID) de la conversación
 - 1.2. El chatbot responde consultas dentro del dominio de IA. (1-5)
 - 1.3. El chatbot evita responder fuera del dominio de IA. (1-5)
 - 1.4. El chatbot sugiere redirigir la conversación cuando el tema es ajeno a IA. (1-5)

1.5.	Comentarios	
2.	RF-2. Interacción basada en fuentes curadas	
	■ Descripción: Las respuestas se construyen únicamente con información del repositorio curado e incluyen citas completas. Si el chatbot no dispone de la información necesaria, lo indica y sugiere recursos alternativos. ■ Instrucciones: Converse sobre IA. Pregunte también sobre un concepto inventado para observar cómo reacciona ante información ausente en el repositorio.	
2.1.	Identificador (ID) de la conversación	
2.2.	El chatbot cita las fuentes que respaldan la información.	(1-5)
2.3.	Las citas son completas, claras, legibles y fácilmente identificables. .	(1-5)
2.4.	Cuando no posee la información, lo indica y sugiere recursos alternativos. (1-5)	
2.5.	Comentarios	
3.	RF-5. Interacción adaptable al perfil del usuario	
	■ Descripción: El sistema ajusta la complejidad y el detalle según la experiencia del usuario. ■ Instrucciones: Realice dos consultas sobre IA, indicando distintos niveles de experiencia: una como principiante y otra como experto.	
3.1.	Identificador (ID) de la conversación	
3.2.	Para principiantes, las respuestas son simples y claras.	(1-5)
3.3.	Para expertos, las respuestas son técnicas y profundas.	(1-5)
3.4.	Comentarios	
4.	RF-3 y RF-4. Interacción basada en diálogo y estrategias pedagógicas	
	■ Descripción: Las respuestas son claras, concisas y graduales. El chatbot mantiene el contexto de la conversación e incorpora repreguntas como estrategia pedagógica. ■ Instrucciones: Converse libremente (≥ 10 mensajes), verifique si el chatbot conserva el contexto y realiza repreguntas.	
4.1.	Identificador (ID) de la conversación	
4.2.	Las respuestas son claras, concisas y graduales.	(1-5)
4.3.	El chatbot interpreta referencias a mensajes anteriores.	(1-5)

- 4.4. Utiliza el contexto previo de la conversación para responder con coherencia. (1-5)
- 4.5. Incorpora repreguntas para estimular la reflexión. (1-5)
- 4.6. Las repreguntas son pertinentes y enriquecen la experiencia. (1-5)
- 4.7. Tras la repregunta, brinda la respuesta si el usuario lo solicita. (1-5)
- 4.8. Comentarios

5. Valoración global

- **Descripción:** Impresión general sobre utilidad, usabilidad y satisfacción.

- 5.1. La interfaz es intuitiva y fácil de usar. (1-5)
- 5.2. El chatbot es útil para aprender o consultar sobre IA. (1-5)
- 5.3. No experimentó fallas o comportamientos erróneos frecuentes. (1-5)
- 5.4. No identifica mejoras críticas y está satisfecho/a con el sistema. (1-5)
- 5.5. Comentarios y sugerencias generales

4.3.4. Resultados

La rúbrica propuesta en la sección anterior se compartió con diez docentes-evaluadores. Los ítems analizados permiten extraer un balance general muy favorable, aunque con algunas áreas de mejora que podrían guiar trabajos futuros. El conjunto completo de respuestas anonimizadas puede encontrarse en el anexo C.

Respecto del cumplimiento de los requerimientos funcionales, el desempeño fue alentador. El primero de ellos, la restricción temática al dominio de la inteligencia artificial (RF-1), se cumplió de manera consistente. Hubo consenso por parte de los evaluadores en que el chatbot se mantiene dentro de la temática de inteligencia artificial y, cuando se le consulta sobre otros asuntos, intenta redirigir la conversación hacia su dominio temático. Algunos evaluadores dejaron comentarios detallando su experiencia, por ejemplo:

«Intenté consultar sobre una receta de galletas con miel. Con amabilidad, me aclaró que se especializa en ayudar con temas relacionados con la inteligencia artificial y aplicaciones en educación y me sugirió plataformas para recetas de cocina. Procedí entonces a hacerle una pregunta concreta sobre AI y educación y me contestó en forma detallada, citando además la fuente consultada. Le agradecí y quedó a disposición para futuras preguntas.»

Por otro lado, el segundo requerimiento (RF-2), referente al uso exclusivo de fuentes curadas para generar las respuestas y la inclusión de citas completas, obtuvo una puntuación sobresaliente. Nueve evaluadores confirmaron que las

respuestas ofrecieron referencias claras y legibles. Cuando la información solicitada no estaba disponible, el sistema, por lo general, informó esta limitación y propuso recursos alternativos. No obstante, tres evaluadores señalaron que el chatbot no siempre dejó claro que no contaba con la información, lo que revela una oportunidad de mejora en la comunicación del desconocimiento. Por ejemplo, un evaluador probó inventar una teoría con un nombre ficticio. Inicialmente, el chatbot asumió que se trataba de una teoría real y, por momentos, conversó sobre cómo podría aplicarse en contextos educativos. Sin embargo, recién cuando se le pidió que enunciara dicha teoría, el sistema indicó no disponer de información al respecto.

Adicionalmente, dos usuarios propusieron la posibilidad de mejorar las citas incluyendo información sobre la publicación (por ejemplo, editorial) o incluir un *link* que permitiera ver o descargar el documento citado. Este tema se retomará en la sección trabajos futuros 6.2.

En cuanto a la interacción conversacional (RF-3), los evaluadores coincidieron en que el prototipo fue capaz de recordar el contexto dentro de la sesión y usarlo para mantener coherencia en las respuestas. Además, la capacidad para desarrollar las explicaciones de forma gradual y el grado de concreción en las respuestas fueron bien valorados en general, aunque dos evaluadores asignaron una puntuación de 3 sobre 5 a este aspecto, lo que sugiere que, en algunos casos, la extensión de las respuestas podría no haber sido del todo adecuada.

En relación con el cuarto requerimiento (RF-4), vinculado a la aplicación de la estrategia pedagógica de repregunta, los resultados fueron en general positivos: la mayoría percibió que el chatbot, en algunas ocasiones, postergaba la respuesta, motivando la reflexión y el razonamiento antes de ofrecer la explicación completa. Solo un usuario indicó estar en desacuerdo con el cumplimiento de este requerimiento, lo que sugiere que no recibió repreguntas durante su interacción. Dado que este comportamiento no es determinista, es esperable que pueda no darse en todos los casos.

Además, la mayoría coincidió en que las repreguntas fueron pertinentes y enriquecieron la experiencia, evitando usos excesivos o fuera de lugar. Solo un evaluador asignó una puntuación de 3 sobre 5 a este aspecto, lo cual indica que las repreguntas realizadas por el chatbot no le resultaron del todo adecuadas. A su vez, hubo consenso en que, tras realizar una repregunta, el chatbot brinda la respuesta correspondiente si el usuario indica que no puede deducirla o solicita ayuda. Como comentario adicional, en las respuestas cualitativas vinculadas a este requerimiento se señaló que la estrategia de repregunta fue una de las funcionalidades más novedosas, ya que algunos evaluadores no esperaban ese tipo de intervención reflexiva por parte del chatbot.

La adaptación al perfil del usuario (RF-5) mostró resultados mixtos. Cuando los evaluadores se presentaron como principiantes, todos recibieron respuestas sencillas, con lenguaje claro y ejemplos básicos. Sin embargo, al declararse como usuarios avanzados, ocho de ellos experimentaron un incremento notorio en la tecnicidad del discurso. Los dos restantes percibieron diferencias insuficientes, lo que indica que la lógica de adaptación al perfil podría requerir un ajuste más fino para acentuar el contraste entre niveles de conocimiento. Además, esta

limitación en la capacidad de responder con profundidad y tecnicismo podría estar relacionada con una cantidad insuficiente de documentos con foco técnico en los temas analizados, lo que sugiere que el repositorio podría ampliarse para abarcar mayor cantidad de consultas. Esta necesidad se ve reforzada por comentarios un evaluadores, quienes señalaron que, si bien el chatbot ofrecía información general adecuada, en ciertos casos no pudo proporcionar ejemplos específicos o definiciones suficientemente detalladas. Por ejemplo, uno de ellos señaló la falta de ejemplos concretos en ciertos temas, mientras que otro consideró que la explicación dada para una definición fue demasiado general. En la sección de trabajos futuros se contempla la ampliación del corpus de documentos, lo cual contribuiría en gran medida a enriquecer la calidad y profundidad de las respuestas del chatbot.

En términos generales, la aceptación global de la herramienta fue alta. Todos los participantes la consideraron útil para aprender o consultar sobre inteligencia artificial y, salvo un caso aislado que asignó un 3, el resto declaró sentirse satisfecho con el estado actual del sistema. Asimismo, los evaluadores coincidieron en que la interfaz del chatbot resulta intuitiva y fácil de usar.

Sin embargo, tres de los diez evaluadores reportaron fallas durante alguna de sus pruebas. Si bien se trató de errores transitorios que se resolvieron al refrescar el chat, su aparición evidencia oportunidades de mejora en la estabilidad del sistema. En un caso, se produjo un error de *parsing* del frontend (*“Unexpected token ... is not valid JSON”*), en otro un *timeout* aislado y en el tercero un error no especificado. Para cada uno de esos problemas se propondrá soluciones en la sección de trabajos futuros 6.2. Del mismo modo, un usuario manifestó la necesidad de contar con una funcionalidad que permita guardar y reabrir conversaciones pasadas, aspecto contemplado por el requerimiento RF-6, pero que no fue implementado en esta versión del prototipo. Este aspecto también será abordada en la sección de trabajos futuros.

Finalmente, los comentarios generales recogidos en el bloque de valoración global reflejan una percepción positiva del prototipo. Algunos evaluadores dejaron comentarios detallando su experiencia, por ejemplo:

«Esta experiencia fue muy interesante y un incentivo para continuar incorporando conocimientos al respecto de la IA en la educación y otras áreas.»

«En general me parece muy bueno el chatbot. Felicitaciones. Es fácil de usar, tiene una estética de letras y colores que está muy bien. No tuve demasiado tiempo para probarlo, pero puede seguir las indicaciones sin problemas. Revisé alguna de las referencias y son reales. Tiene un tono agradable para la interacción. Éxitos en el cierre del proyecto!»

«El trabajo que han hecho es excelente. Han escuchado a las clientas :-). Gracias por eso. Voy primero con las sugerencias de mejora: Desde el punto de vista técnico, como usuaria, sólo decir que extraño poder guardar la conversación para poder retomarla después... Sería muy valioso, que, además de la cita, brindara el enlace al documento en el repositorio... Por último decirles, que en esta conversación que tuve con el chatbot de ustedes se me ocurrieron 3 aplicaciones para 3 diferentes contextos de clase. ¡Así que miren si sirve! ... ¡Muchas felicitaciones! Excelente trabajo.»

En síntesis, la estrategia de evaluación propuesta comprende un ciclo completo de validación, que incluye pruebas *offline* automatizadas para evaluar cada componente por separado, el seguimiento *online* de la solución en el entorno productivo y la apreciación práctica por parte de los usuarios finales. Este enfoque integral busca asegurar no solo la corrección y robustez técnica del chatbot, sino también su capacidad para cumplir los requerimientos definidos y ofrecer una experiencia de uso satisfactoria.

Capítulo 5

Experimentación

A lo largo del desarrollo del sistema, se exploraron distintas alternativas arquitectónicas antes de llegar a la solución presentada en el capítulo 3.3.1. Aunque el esquema general de la solución se mantuvo consistente con lo expuesto en el diagrama de alto nivel de la Figura 3.1, la componente *Backend* experimentó diversas modificaciones.

Paralelamente, se llevaron a cabo experimentaciones en torno a la componente *RAG*, incluyendo la elección del modelo de lenguaje, los modelos de *embeddings* y los métodos de *chunking*.

El propósito de este capítulo es proporcionar una visión detallada del proceso de experimentación, destacando los aprendizajes obtenidos en cada iteración y justificando las decisiones que condujeron al diseño e implementación actuales. Para ello, se documentan dos enfoques previos para la componente *Backend*, analizando sus fortalezas y limitaciones, y comparándolos con la arquitectura finalmente adoptada. Asimismo, se detallan las pruebas realizadas, que permitieron fundamentar la selección del modelo de lenguaje, el modelo de *embeddings* y la técnica de *chunking*, así como el ajuste de sus parámetros.

5.1. Experimentación sobre la arquitectura

5.1.1. Primera arquitectura planteada

La primera arquitectura propuesta para el *Backend* se ilustra en la Figura 5.1. En esta aproximación, tanto el *Endpoint* como el *RAG* operan de la misma manera que se describió en la sección de Diseño y Arquitectura.

Por su parte, el *Router LLM* evalúa si es necesario recuperar información de la base de datos antes de generar una respuesta. Si lo es, invoca al *RAG* para obtener los datos pertinentes. Finalmente, el *Response LLM* genera la respuesta definitiva para el usuario, integrando la información recuperada cuando corresponda.

A continuación, se detallan estas dos componentes y su papel dentro de la arquitectura.

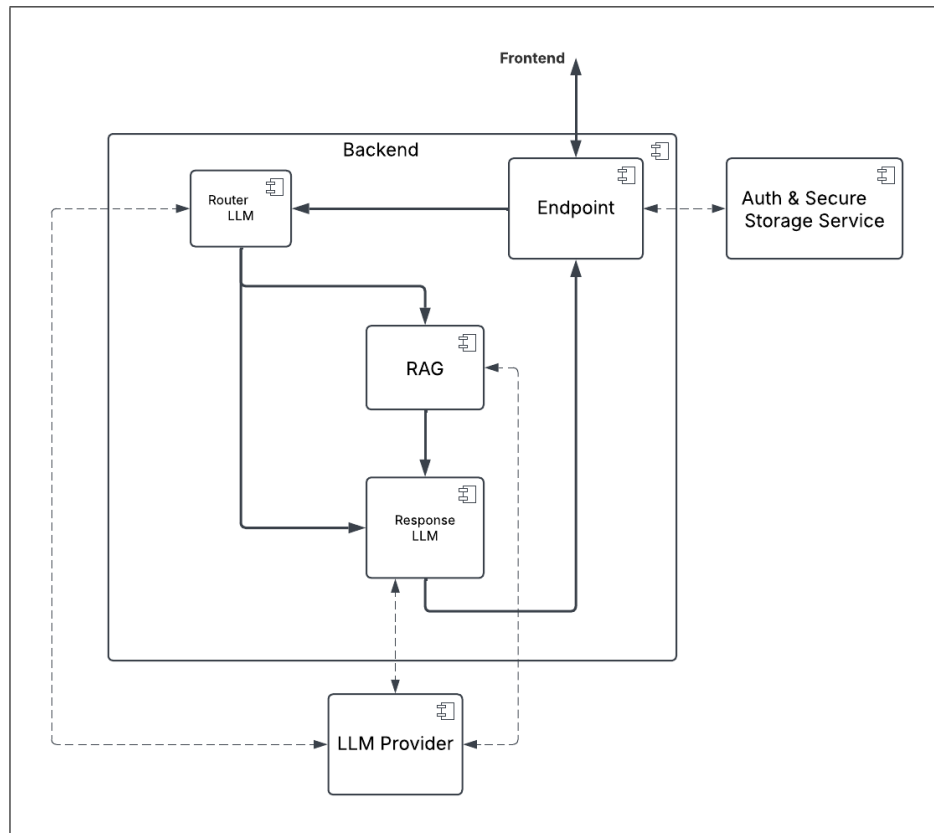


Figura 5.1: Estructura interna del *Backend* y sus distintos LLMs en la primera arquitectura.

Router LLM

Esta componente está encargada de determinar si la consulta del usuario requiere recuperar información de la base vectorial o si puede ser respondida directamente. Consiste en un LLM que utiliza el mecanismo de *tool calling*, el cual no solo permite generar texto, sino también estructurar su salida para indicar la invocación de funciones predefinidas, como se explicó en la sección 2.3.2. La función del *Router LLM* es analizar el contenido de la consulta y decidir si debe invocar la herramienta encargada de buscar información en la base de datos. Si la consulta involucra definiciones, explicaciones, reflexiones o temas que pueden beneficiarse de información adicional, devuelve una salida estructurada indicando que se debe invocar la herramienta de búsqueda. En cambio, si la consulta corresponde a saludos o conversación trivial, determina que no es necesario recuperar datos. Es importante destacar que el *Router LLM* no proporciona la respuesta final a la consulta, sino que únicamente decide si es necesario llamar a la herramienta de búsqueda según un conjunto de instrucciones definidas en

su *prompt*. En los casos donde no se requiere recuperar información, también proporciona una justificación de esta decisión.

Es importante notar que la función del *Router LLM* en esta arquitectura difiere de la presentada en la sección de Diseño y Arquitectura. En este caso, solo se contemplan dos posibles decisiones: recuperar información o no hacerlo. En cambio, en la arquitectura original, el *Router LLM* manejaba cuatro caminos: recuperación de información (*RAG* seguido del *Informative Conversational LLM*), denegación de respuesta por ser ajena a la IA (*Denial LLM*), aplicación de estrategias pedagógicas (*Pedagogical LLM*) y generación de respuestas a mensajes triviales (*Casual LLM*). Esta diferencia se debe a que, en la presente arquitectura, las tres últimas decisiones recaen sobre el *Response LLM*, mientras que la recuperación de información sigue el flujo *RAG* antes de pasar al *Response LLM*. A continuación, se detalla el funcionamiento de la componente *Response LLM*.

Response LLM

Esta componente se encarga de generar la respuesta final del chatbot. Consiste en un LLM que recibe el historial de la conversación, la nueva consulta del usuario, y, cuando corresponde, la información recuperada. A partir de esto, produce una respuesta en lenguaje natural que se envía al usuario como respuesta del chatbot.

Su comportamiento está definido por un *prompt* detallado que orienta la forma en que interactúa y estructura sus respuestas. Entre sus principales directrices se encuentran:

- **Enfoque en IA (RF-1)**: El modelo debe responder exclusivamente preguntas relacionadas con inteligencia artificial. Si el usuario consulta sobre otro tema, debe redirigir amablemente la conversación hacia la IA o su aplicación en la educación, explicando que solo puede asistir en este ámbito. De esta manera, el *Response LLM* cumple con la función de denegar respuestas fuera del dominio del chatbot, una responsabilidad que según lo definido en el Diseño y Arquitectura correspondía al *Denial LLM*.
- **Fomento de la reflexión (RF-4)**: Algunas veces, en lugar de ofrecer respuestas directas, el modelo debe incentivar el pensamiento crítico, formulando preguntas que guíen al usuario hacia la reflexión. De este modo, el *Response LLM* cumple también con la función de aplicar estrategias pedagógicas, en particular la repregunta, asignada al *Pedagogical LLM* en el Diseño y Arquitectura.
- **Uso de la información recuperada y estilo conversacional (RF-2 y RF-3)**: Cuando el *Router LLM* indica que es necesario recuperar información, el *Response LLM* debe generar su respuesta exclusivamente a partir de la información obtenida, sin inferir ni inventar contenido. Además, debe ir entregando la información de manera progresiva para no

sobrecargar al usuario, con la orden de mantener siempre un tono conversacional natural. De este modo, cumple la función de generar respuestas basadas en la información recuperada por el *RAG*, manteniendo un tono conversacional (*Informative Conversational LLM*).

- **Interacción amigable y didáctica:** Además de las directrices anteriores, el modelo es indicado para responder de manera amigable y didáctica, adaptándose al contexto de la conversación. Esto le permite también manejar interacciones informales, como saludos o mensajes triviales, siempre dentro del marco de un chatbot enfocado en IA. Así, también cumple con la función de generar respuestas a mensajes triviales (*Casual LLM*).

Como se puede observar, esta arquitectura cuenta con tres LLMs principales: el *Router LLM*, el *Response LLM* y el LLM del *RAG*. Si bien podría clasificarse dentro del enfoque de múltiples LLMs explicado en la sección 2.3.4, el *Response LLM* en particular concentra múltiples responsabilidades, como fue mencionado anteriormente, y podría ser descompuesto en distintos modelos especializados para optimizar su desempeño y mejorar la modularidad del sistema.

De hecho, el principal problema de esta solución fue que el *Response LLM* no siempre seguía las indicaciones definidas en el *prompt*. A pesar de múltiples iteraciones y ajustes en el diseño del *prompt* para reforzar las instrucciones que no se estaban cumpliendo, no se logró un comportamiento completamente alineado con los requerimientos. Se generaron diversas versiones de *prompts* intercambiables para optimizar su desempeño, pero ninguna consiguió abarcar satisfactoriamente todas las responsabilidades asignadas. Este problema puede atribuirse a la carga excesiva de responsabilidades sobre un único modelo, lo que resultó en un *prompt* extenso con las dificultades inherentes a esta aproximación, como se discutió en la sección de Arquitecturas.

Si bien esta implementación inicial presentaba una menor complejidad en comparación con la solución final, se llegó a la conclusión de que una arquitectura tan simplificada no sería suficiente para satisfacer todos los requerimientos. Por esto, se decidió descomponer las tareas del *Response LLM* y distribuir las entre componentes especializadas, asegurando así una mayor robustez y control sobre cada funcionalidad.

5.1.2. Segunda arquitectura planteada

La segunda arquitectura propuesta para el *Backend* se ilustra en la Figura 5.2. Nuevamente, en esta aproximación, tanto el *Endpoint* como el *RAG* operan de la misma manera que se describió en la sección de Diseño y Arquitectura (3.3.1).

La principal diferencia en esta nueva arquitectura radica en el *Router*. En lugar de ser un LLM que evalúa si es necesario recuperar información de la base de datos para generar la respuesta, esta versión implementa una lógica aleatoria para decidir entre dos estrategias: recuperar información o aplicar una estrategia pedagógica. Si el *Router* determina que se debe recuperar información,

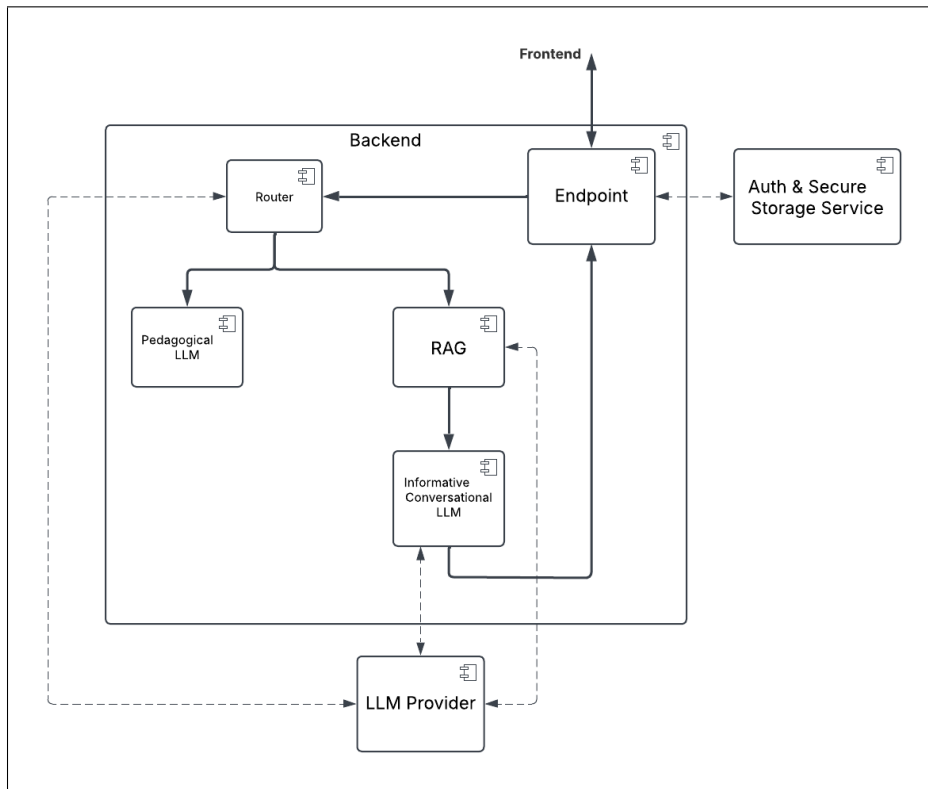


Figura 5.2: Estructura interna del *Backend* y sus distintos componentes en la segunda arquitectura.

se invoca al *RAG* y, posteriormente, el *Informative Conversational LLM* genera la respuesta final para el usuario, integrando la información obtenida. En caso contrario, se invoca al *Pedagogical LLM*, encargado de formular una repregunta para fomentar la reflexión y el aprendizaje del usuario.

A continuación, se detallan estas tres componentes y su papel dentro de la arquitectura.

Router

- Componente encargada de decidir si la respuesta al usuario incluirá información recuperada o si, en su lugar, se aplicará una estrategia pedagógica. A diferencia de la solución anterior y de la versión final, en este caso la decisión se toma de manera aleatoria, con un 50% de probabilidad para cada opción.

Esta aproximación surgió de la idea de aplicar estrategias pedagógicas de forma controlada mediante un factor aleatorio, con la posibilidad de ajustar dicho factor para reducir la frecuencia de repreguntas por debajo

del 50 %. Incluso se consideró una variante en la que este factor aumentara progresivamente a lo largo de la conversación (hasta un límite máximo), permitiendo inicialmente respuestas informativas para captar el interés del usuario y, a medida que avanzara la interacción, incorporar un mayor número de repreguntas. Esta estrategia fue discutida en reuniones con las usuarias expertas.

- Notar que, en esta versión, el *Router* también considera solamente dos posibles caminos, en vez de contemplar los cuatro definidos en el Diseño y Arquitectura.

Informative Conversational LLM

- Componente encargada de generar la respuesta final del chatbot cuando el *Router* elige el camino de recuperar información de la base de datos. Consiste en un LLM que recibe el historial de la conversación, la nueva consulta del usuario, y la información recuperada, y produce una respuesta que se envía al usuario como respuesta del chatbot.
- El *prompt* que define el comportamiento del modelo es generado mediante la concatenación de distintas directrices, las cuales están centralizadas en un único archivo y son utilizadas tanto por el *Informative Conversational LLM* como por el *Pedagogical LLM*. Esta reutilización de directrices permite mantener coherencia en ciertos comportamientos compartidos entre ambos modelos. A continuación, se detallan las principales directrices que rigen el comportamiento del *Informative Conversational LLM*:
 - **Contexto general del asistente:** El modelo debe comportarse como un asistente conversacional diseñado para ayudar a docentes interesados en la inteligencia artificial generativa. Todas sus respuestas deben alinearse con este propósito, asegurando que su comportamiento sea coherente con su rol y objetivos.
 - **Enfoque en IA (RF-1):** Similar a la directriz utilizada en la primera arquitectura, el modelo debe responder exclusivamente preguntas relacionadas con inteligencia artificial. Si el usuario consulta sobre otro tema, debe redirigir la conversación de manera amable hacia la IA o su aplicación en educación, aclarando que su alcance está limitado a este ámbito. En caso de que el usuario insista en temas ajenos a la IA, el chatbot puede sugerir otras herramientas más adecuadas para su consulta.
 - **Respuestas basadas en la información recuperada (RF-2):** El modelo debe generar respuestas únicamente a partir de la información recuperada, sin añadir contenido adicional. Si no se encuentra información relevante en la base de datos, debe indicar explícitamente esta limitación y recordar al usuario que puede formular otras consultas.

- **Interacción conversacional (RF-3)**: Para evitar sobrecargar al usuario con información excesiva, el modelo debe presentar los contenidos de manera progresiva, manteniendo un tono natural y de diálogo. A medida que la conversación avanza y se detecta un mayor interés del usuario, puede proporcionar detalles adicionales de forma gradual.

Pedagogical LLM

- Componente encargada de generar la respuesta final del chatbot cuando el *Router* elige el camino de aplicar una estrategia pedagógica. Consiste en un LLM que recibe el historial de la conversación, la nueva consulta del usuario, y la información recuperada, y produce una respuesta diseñada para fomentar la reflexión y el aprendizaje.
- Como se mencionó anteriormente, el *prompt* que define el comportamiento del *Pedagogical LLM* se genera mediante la concatenación de varias directrices, algunas de las cuales son compartidas con el *Informative Conversational LLM*. En particular, se comparten las directrices de **“Contexto general del asistente”**, que proporciona el contexto necesario para las respuestas del modelo, y **“Enfoque en IA”**, que asegura que la conversación se mantenga dentro del ámbito de la inteligencia artificial.

Adicionalmente, el *Pedagogical LLM* incorpora la siguiente directriz específica para este modelo:

- **Formulación de preguntas reflexivas (RF-4)**: En lugar de proporcionar una respuesta directa a la consulta del usuario, debe formular una pregunta que lo motive a razonar y reflexionar sobre el tema. Esto debe hacerse siempre sin ignorar la consulta original y reconociendo lo que el usuario ha expresado previamente.

Es importante destacar que, en esta arquitectura, independientemente del camino que elija el *Router*, los LLMs de ambos caminos comparten las mismas directrices respecto al **“Enfoque en IA”**. Esto permite que la funcionalidad de denegar la respuesta en caso de que la consulta no esté relacionada con la inteligencia artificial (tarea asignada al *Denial LLM* en el diseño) pueda ser cumplida por cualquiera de los dos LLMs, ya sea que el *Router* elija el camino del *Pedagogical LLM* o del *Informative Conversational LLM*.

Asimismo, dado que ambos LLMs están contextualizados de igual manera mediante las directrices de **“Contexto general del asistente”**, ambos son igualmente capaces de ofrecer respuestas adecuadas en el contexto de una conversación casual, cumpliendo así con las responsabilidades del *Casual LLM* definidas en el diseño.

De este modo, la arquitectura satisface todas las responsabilidades planteadas en la arquitectura original, asociadas a los cuatro caminos definidos.

Otro punto importante a mencionar es que, en esta segunda arquitectura, se logró una mejora significativa en comparación con la primera arquitectura en relación con los problemas derivados de tener un único LLM (el *Response LLM*) encargado de manejar los cuatro tipos de respuestas que el chatbot puede ofrecer. Si bien en esta segunda arquitectura las responsabilidades podrían descomponerse aún más, ya que tanto la tarea de denegar respuestas como la de responder a mensajes casuales están a cargo de ambos LLMs y no de LLMs específicos, la separación de responsabilidades entre el LLM encargado de aplicar estrategias pedagógicas y el que se encarga de responder con información mejoró notablemente.

En particular, los cambios realizados en los *prompts* se reflejaron de manera más evidente en las respuestas del chatbot al tener *prompts* separados para las distintas responsabilidades. Esto contrasta con los esfuerzos previos —en la primera arquitectura—, donde a pesar de numerosos intentos de modificar el *prompt* del *Response LLM*, muchas veces los cambios no se reflejaban en la generación del modelo, lo que dificultaba la obtención de los comportamientos deseados.

Sin embargo, a pesar de que esta arquitectura es más modular y presenta ventajas en ese sentido, también tiene asociados algunos problemas importantes que impidieron alcanzar el comportamiento esperado, lo que llevó a la necesidad de seguir iterando hasta llegar a la arquitectura final. Más concretamente, se identificaron inconvenientes derivados de la naturaleza aleatoria del *Router*. La decisión aleatoria sobre el camino a seguir podría dar lugar —como se verificó al probar el chatbot y fue señalado por las usuarias expertas en su evaluación (4.2.3)—, a múltiples repreguntas consecutivas al usuario. Esto no solo genera insatisfacción, sino que también impide poder mantener una conversación fluida y provechosa.

Además, la aleatoriedad podría provocar que se formularan repreguntas en momentos poco adecuados, cuando es más apropiado proporcionar información directamente. Por ejemplo, ante preguntas informativas concretas como “¿En qué consiste la arquitectura de los transformadores en el contexto de la inteligencia artificial?”, resulta difícil plantear una repregunta que fomente la reflexión, dado que la consulta es meramente informativa. Por esta razón, la formulación de repreguntas no siempre es adecuada, aunque el *Router* la seleccione aleatoriamente.

Asimismo, otro problema asociado a la aleatoriedad del *Router* es que podría decidir realizar una búsqueda de información incluso ante un saludo o una conversación casual. Esto implica un esfuerzo innecesario en términos de cómputo y búsqueda en la base de datos, ya que, al intentar recuperar información en base a un mensaje casual, no se obtendrá ningún dato relevante de una base de datos centrada en inteligencia artificial. Este comportamiento también provoca que respuestas que podrían generarse rápidamente, sin necesidad de realizar búsquedas, se vean demoradas significativamente, ya que deben pasar por todo el proceso de análisis de la consulta, generación de sub-consultas, recuperación de datos, entre otros.

Los problemas identificados llevaron a que, en la solución final, la componen-

te del *Router* estuviera compuesta por un LLM que, en lugar de tomar decisiones de manera aleatoria, eligiera el camino a seguir en base a indicaciones específicas. Además, al observar los beneficios de una arquitectura más modular, se decidió continuar con la separación de responsabilidades, lo que dio lugar a la creación de los cuatro caminos presentes en la solución final. Esto permitió aprovechar las ventajas de las arquitecturas con múltiples LLMs mencionadas en la sección 2.3.4, facilitando la optimización y evaluación del desempeño del modelo en cada camino.

5.2. Experimentación sobre la Componente *RAG*

5.2.1. Técnicas de *Chunking*

Durante el desarrollo del prototipo se experimentó con las distintas técnicas de *chunking* mencionadas en el anexo F. Para las pruebas de *chunking* semántico se utilizó el método basado en desviación estándar. Se evaluaron distintos umbrales para identificar cuál proporcionaba mejores resultados en las métricas de *Context Relevancy* y *Context Recall*. La tabla 5.1 resume los resultados obtenidos para los distintos valores de umbral así como una comparación con el *baseline* (*Recursive chunking*)¹.

Método	Context Relevancy	Context Recall
Recursive	0.96	0.73
Semantic 0.25	0.98	0.84
Semantic 0.5	1.0	0.91
Semantic 0.75	1.0	0.91
Semantic 1.0	0.99	0.95
Semantic 3.0	0.98	0.92

Tabla 5.1: Evaluación de métodos de chunking

Se puede observar que tanto el *Context Relevancy*, como el *Context Recall*, mejoran sustancialmente respecto al *baseline*. Es importante notar que cuanto más grande el umbral elegido para esta técnica menor cantidad de *chunks* habrá en la base de datos vectorial, y en consecuencia la búsqueda será mas rápida. También es relevante mencionar que debido a la densidad semántica de los textos objetivo de este prototipo, para valores de desvío superiores a 1.0 los *chunks* comprenden una página entera. Tras este análisis se eligió como técnica para este proyecto el ***chunking* semántico con desviación estándar de 1.0**.

¹Los valores presentados para *context relevancy* y *context recall* difieren con los presentados en 4.1.4 debido a que se ejecutaron con una primera versión del *dataset* de evaluación.

5.2.2. Modelos de *Embeddings* Evaluados para la Recuperación

La Tabla 5.2 recoge los modelos de *embeddings* más relevantes (*open-source* y comerciales) para sistemas como el desarrollado en este proyecto. En la tabla se incluyen fecha de lanzamiento, dimensión del vector, puntuación declarada en el *benchmark* MTEB para tareas de recuperación y costo por millón de *tokens*.

Modelo	Fecha	Dim.	MTEB	Costo ²
OpenAI text-embedding-3-small	2024-01-25	1536	62.3	\$0.02
OpenAI text-embedding-3-large	2024-01-25	3072	64.6	\$0.13
OpenAI text-embedding-ada-002	2022-12	1536	61.0	\$0.10
Cohere embed-english-v3.0	2023-11-02	1024	64.5	\$0.10
Nomic embed-text-v1	2024-02	768	62.39	Libre
E5 multilingual-large-instruct	2024	1024	64.4	Libre
Sentence-T. all-mpnet-base-v2	2021	768	63.3	Libre

Tabla 5.2: Comparativa de modelos de embeddings textuales para tareas de recuperación. Valores tomados en mayo de 2025 de: (PingCAP, 2024), (OpenAI, 2024), (OpenAI, 2024), (Cohere, 2023), (Nomic AI, 2024), (Wang y cols., 2024), (SBERT, 2024)

Sería posible y prudente elegir modelos *open weights* como *embed-text-v1* de Nomic, sin embargo, debido a que se cuenta con una máquina virtual con recursos limitados se debe optar por modelos de proveedores. De todos modos, se presenta esta comparativa para dejar en evidencia que sería perfectamente posible usar un modelo abierto para esta tarea.

Entre las alternativas analizadas, *text-embedding-3-small* de OpenAI proporciona el mejor equilibrio entre precisión y costo: mejora a *ada-002* en MTEB (+1.3 puntos) mientras reduce el precio 5 veces respecto a los demás modelos de proveedores escrutados. Adicionalmente, este modelo es de la familia A.0.3 con lo que trae consigo las propiedades mencionadas en dicha sección.

5.2.3. Comparativa de Desempeño de Modelos de Lenguaje

Para fundamentar la elección del modelo de lenguaje a utilizar, se consolidaron los puntajes *zero-shot* en cinco de los *benchmarks* más utilizados en la literatura: WINOGRAD, HELLASWAG, MMLU, ARC-CHALLENGE (conjunto difícil) y ARC-AGI. Los valores de la tabla 5.3 se obtuvieron de los informes y artículos originales de cada proveedor: Mixtral 8×7B (Jiang y cols., 2024), GPT-3.5-turbo-0125 (OpenAI, 2023c), GPT-4o-mini (Menick y cols., 2024), GPT-4o (Hurst y cols., 2024), GPT-4 (Achiam y cols., 2023), Claude 2, 2.1 e Instant 1.2 (Anthropic, 2024b) y Llama 3 (Grattafiori y cols., 2024).

²Costo expresado en USD por millón de tokens de entrada

Modelo	Winograd	HellaSwag	MMLU	ARC-Challenge	ARC-AGI
Mixtral 8×7B	77.2 %	84.4 %	70.6 %	85.8 %	—
GPT-3.5-turbo-0125	67.0 %	—	—	—	—
GPT-4o-mini	—	—	82.0 %	—	—
GPT-4o	—	—	87.2 %	—	—
GPT-4	83.0 %	82.0 %	86.4 %	84.0 %	2.0 %
Claude 2.1	—	—	78.5 %	—	—
Claude 2	—	—	75.5 %	—	—
Claude Instant 1.2	—	—	67.5 %	—	—
Llama 3 8B	77.1 %	82.0 %	66.5 %	59.2 %	—
Llama 3 13B	79.0 %	80.5 %	63.5 %	56.0 %	—
Llama 3 70B	85.0 %	88.0 %	77.5 %	72.0 %	—

Tabla 5.3: Exactitud (%) *zero-shot* en cinco *benchmarks* globales.

A simple vista, los modelos de la familia Llama 3 alcanzan precisiones notables en WINOGRAD y HELLASWAG, aprovechando su afinamiento en tareas de razonamiento y sentido común. No obstante, cuando se requiere un conocimiento amplio de múltiples disciplinas, como en MMLU, o un razonamiento multihop más profundo (ARC-CHALLENGE), su rendimiento cae por debajo de lo que ofrecen las variantes de OpenAI. GPT-4, en particular, exhibe una consistencia superior al 80 % en casi todos los escenarios, y es la única instancia que supera la barrera del 80 % en MMLU. GPT-3.5, en contraste, se mantiene alrededor del 67 % en WINOGRAD y carece de datos *zero-shot* en las otras pruebas, lo que subraya sus limitaciones sin ayuda de *few-shot* o CoT.

Desempeño en Español

Otro aspecto esencial a confirmar es que las fortalezas observadas en inglés se trasladen al español. Para ello, se evaluaron las versiones traducidas de WINOGRAD (**wnli-dev-es**) y HELLASWAG (**hella-swag-es**). La tabla 5.4 resume los puntajes de F1 y el promedio total (*Total AVG*) para cada modelo.³

Modelo	wnli-dev-es (F1)	hella-swag-es (F1)	Total AVG
GPT-4o	0.35	0.94	0.65
GPT-4-Turbo-2024-04-09	0.28	0.91	0.60
Mixtral 8×7B	0.40	0.70	0.55
Claude Instant 1.2	0.41	0.68	0.55
Llama 3 70B (8192 tokens)	0.22	0.81	0.52
Claude 2.1	0.30	0.73	0.51
Gemma 8B-It	0.51	0.48	0.50
Llama 2 70B-Chat	0.26	0.73	0.49
Claude 2	0.26	0.70	0.48
GPT-3.5-turbo-0125	0.30	0.48	0.39

Tabla 5.4: F1 en versiones españolas de WINOGRAD y HELLASWAG, con promedio simple.

³Las métricas presentadas fueron obtenidas mediante evaluaciones locales, realizadas en un entorno reproducible y consistente, garantizando la equidad en la comparación entre modelos.

El cuadro muestra que GPT-4o lidera con claridad en castellano, alcanzando un F1 de 0.94 en HELLASWAG-ES y 0.35 en WINOGRAD-ES, lo que se traduce en un promedio de 0.65. Este, supera con margen al segundo clasificado, GPT-4-Turbo, y supera ampliamente a los modelos *open-source* como Mixtral o Llama 3. El hecho de que GPT-4o preserve su ventaja sin necesidad de ajustes técnicos refuerza su robustez multilingüe y confirma que su solidez en inglés se traslada al español cuando se evalúa razonamiento contextual y sentido común.

Desempeño en HotpotQA

Para cerrar el análisis, HotpotQA en su modalidad *distractor* mide la capacidad de integrar dos evidencias separadas en una única respuesta coherente. La tabla 5.5 recoge las métricas de *Exact Match* (EM) y F1 en configuración *zero-shot* para los modelos con datos publicados.

Modelo	EM	F1
GPT-4o	72.2 ± 0.6	82.7 ± 0.7
GPT-4	69.9 ± 1.5	82.3 ± 1.3
GPT-3.5-turbo-0125	58.6 ± 0.9	69.1 ± 1.1
GPT-4o-mini	≈ 58	—
Mixtral 7×8 B	—	—
Claude 2 / 2.1 / Instant 1.2	—	—
Llama 3 (8 B / 13 B / 70 B)	—	—

Tabla 5.5: Resultados zero-shot en HotpotQA (*distractor setting*).

Las cifras de HotpotQA enfatizan la superioridad de GPT-4o en tareas de razonamiento *multihop*. Con un F1 de 82.7 %, supera por más de trece puntos a GPT-3.5 y aventaja ligeramente a GPT-4. Esta diferencia no es menor. Sintetizar información dispersa y desechar distractores exige al modelo ser sensible a señales y poco sensible al ruido, lo que señala a GPT-4o como la mejor opción para la tarea propuesta.

Capítulo 6

Conclusiones y Trabajos Futuros

En este capítulo se presentan las principales conclusiones del proyecto, así como posibles líneas de trabajo futuras que permitirían continuar y profundizar en los desarrollos realizados.

6.1. Conclusiones

Este proyecto se originó con el propósito inicial de desarrollar un chatbot generativo de código abierto. Sin embargo, rápidamente se evidenció que uno de los principales desafíos no solo residía en cumplir con los requerimientos técnicos o investigar un campo en constante evolución, sino también en gestionar dinámicamente las expectativas del cliente y ajustar de manera continua el alcance del proyecto según sus necesidades reales y los recursos disponibles, especialmente el tiempo.

Durante varios meses se realizaron iteraciones sobre los requerimientos y objetivos del sistema, adaptándolos a medida que se recopilaba retroalimentación y se validaba técnicamente cada idea propuesta. En este proceso se definieron cuatro hitos fundamentales: investigar el estado del arte, diseñar una arquitectura robusta que cumpliera con los requisitos funcionales y no funcionales previamente delimitados, desarrollar un prototipo escalable y, finalmente, llevar a cabo evaluaciones del sistema con usuarios reales.

A continuación, se detalla cómo se cumplieron dichos objetivos, en qué aspectos se superaron o quedaron pendientes, los logros alcanzados y las principales lecciones aprendidas a lo largo de este proceso.

Síntesis del cumplimiento de objetivos

1. **Relevamiento bibliográfico y análisis crítico.** Se revisaron principalmente cinco trabajos relevantes sobre chatbots educativos, sistemas RAG

y personalización de LLMs, seleccionados mediante *snowballing* y búsquedas sistemáticas en Scopus. El análisis permitió:

- Identificar brechas significativas entre métodos tradicionales basados en flujos de decisión y enfoques modernos apoyados en LLMs.
- Justificar técnicamente la elección de RAG como estrategia principal para especializar el chatbot en un dominio específico y garantizar la correcta atribución y trazabilidad de las fuentes empleadas.
- Integrar buenas prácticas identificadas sobre evaluación automática y con usuarios, enriqueciendo la metodología propuesta.

2. **Diseño arquitectónico alineado a requerimientos.** Se propuso una arquitectura modular multi-LLM que separa claramente la orquestación (*Router LLM*), la recuperación (subsistema RAG) y la generación según cuatro estilos de respuesta (informativa, causal, pedagógica y de denegación). Este diseño:

- Garantiza la escalabilidad y modularidad solicitadas, facilitando la sustitución y actualización independiente de componentes.
- Cubre la mayoría de los requerimientos funcionales definidos, asegurando una respuesta adecuada a los distintos escenarios de uso.
- Responde al objetivo pedagógico mediante la incorporación de componentes especializadas que apoyan el aprendizaje.
- Asegura la trazabilidad factual y rigor académico mediante referencias explícitas a fragmentos extraídos de un repositorio curado.

3. **Implementación con tecnologías actuales y código abierto.** El prototipo se implementó utilizando tecnologías consolidadas en la industria (Python con FastAPI, React con Tailwind, Milvus, LangChain y Firebase), y fue puesto a disposición pública en GitHub para fomentar la colaboración comunitaria. Esta implementación:

- Cumple en gran medida con los requisitos no funcionales definidos.
- Garantiza la confidencialidad y seguridad de los datos de los usuarios mediante la elección de un modelo propietario que no reutiliza las entradas para su entrenamiento ni almacenamiento, conforme a los términos de uso del proveedor.
- Protege la propiedad intelectual de las fuentes utilizadas mediante la extracción automática y almacenamiento de los metadatos de autoría para su inclusión explícita en las respuestas generadas.
- Contribuye a la escalabilidad y mantenibilidad del sistema mediante el uso de tecnologías como el *framework* LangChain para la orquestación y Docker para el despliegue y portabilidad del *backend*.

4. **Evaluación integral.** Se adoptó un enfoque exhaustivo de evaluación, compuesto por tres niveles complementarios: pruebas automatizadas *offline*, monitoreo y depuración *online*, y validación directa con usuarios del público objetivo. Estas capas permitieron identificar y corregir fallos técnicos tempranamente, optimizar la eficiencia operativa del sistema y garantizar que el chatbot cumpliera eficazmente con su propósito educativo en contextos reales.

Aportes distintivos y valor agregado

Este proyecto introduce un chatbot pedagógico que, a diferencia de las soluciones genéricas, incorpora técnicas de repregunta diseñadas para fomentar el pensamiento crítico en los usuarios. Este enfoque permite emular más de cerca el comportamiento de un asistente humano y refuerza el proceso de aprendizaje. En lugar de limitarse a entregar grandes bloques de información, el sistema estimula la reflexión del usuario, evitando una experiencia pasiva que puede resultar monótona y propensa al olvido.

Otra de sus fortalezas es la entrega de respuestas fundamentadas y la inclusión de referencias junto con las respuestas, lo que no solo brinda mayor confianza al usuario, sino que también aborda uno de los principales desafíos de los modelos de lenguaje: las alucinaciones. Esta característica permite, además, asegurar la trazabilidad de las fuentes utilizadas y facilita al usuario la posibilidad de profundizar en los contenidos. Durante la etapa de evaluación, se realizaron esfuerzos específicos para garantizar que la información recuperada de los documentos no fuera alterada por el modelo de lenguaje, preservando así la fidelidad al contenido original y asegurando la validez de las citas presentadas en las respuestas.

Además, el hecho de que el sistema se mantenga rigurosamente dentro del dominio de la inteligencia artificial —gracias a la componente de ruteo *Router LLM*—, asegura que cada interacción se mantenga alineada con su propósito educativo y que el chatbot no intente responder consultas para las cuales no fue diseñado ni cuenta con respaldo documental.

Otra característica importante es que se ha construido un *pipeline* de evaluación totalmente reproducible: los *scripts* y conjuntos de datos entregados automatizan la verificación de catorce métricas. Entre ellas, se incluyen métricas propias desarrolladas especialmente para este proyecto, como “*Acknowledges Contradiction*” y “*Citations Real and Used*”, que permiten evaluar aspectos clave del comportamiento esperado.

Por último, el proyecto promueve la transparencia y la colaboración comunitaria al ofrecer en el repositorio público guías detalladas que permiten cambiar el dominio temático al que se restringe el chatbot o ajustar el contenido del repositorio de documentos, facilitando así su adopción y extensión por parte de otros grupos interesados en el ámbito educativo.

Comparación con trabajos previos

En la revisión de antecedentes se analizaron cinco trabajos relevantes, destacando particularmente el estudio de Siragusa y Pirrone (*Unipa-GPT*) (Siragusa y Pirrone, 2023), debido a que es el único que comparte, de manera integral, los aspectos esenciales del prototipo desarrollado en esta investigación: (i) un modelo de lenguaje (LLM) de uso general, (ii) una arquitectura RAG que limita sus respuestas a un corpus cuidadosamente seleccionado, y (iii) un objetivo explícitamente pedagógico dentro de un contexto universitario. Los trabajos iniciales revisados (Pyo y *Python-Bot*) se fundamentan en flujos rígidos de decisión, carentes de mecanismos dinámicos de recuperación; por su parte, *CarExpert* y el estudio sobre recuperación de información (IR) mediante LLMs, aunque valiosos para inspirar metodologías de ingesta documental y evaluación métrica, se ubican en dominios distintos al ámbito educativo. En consecuencia, la comparación detallada con *Unipa-GPT* permite evaluar logros y limitaciones bajo supuestos operativos equiparables.

El prototipo propuesto presenta mejoras significativas respecto a lo identificado por Siragusa y Pirrone. Aunque ambos proyectos comparten la premisa básica de restringir respuestas a un corpus curado y ofrecer apoyo académico universitario, divergen en dos aspectos fundamentales. En primer lugar, en cuanto a la orquestación: mientras *Unipa-GPT* depende de un único prompt condensado para guiar las respuestas del modelo, el sistema presentado introduce una componente *Router LLM* que distribuye la generación de respuestas en cuatro rutas especializadas. Este mecanismo alcanzó una precisión de ruteo (*Routing Accuracy*) de **0,99**, lo que permite que el sistema adapte su comportamiento de forma coherente con cada tipo de interacción y contexto. En segundo lugar, respecto al enfoque pedagógico, el prototipo incorpora un camino explícito de repreguntas orientado a fomentar la reflexión del usuario, lo cual sucedió en el 90 % de los casos durante la evaluación con usuarios, una funcionalidad ausente en *Unipa-GPT*.

En resumen, la solución propuesta no solo iguala la precisión factual observada en estudios previos, sino que además incorpora estrategias didácticas y una arquitectura que permite una clara separación de responsabilidades, lo que contribuye a su escalabilidad y mantenibilidad. De este modo, se posiciona como un asistente conversacional robusto y útil para la comunidad educativa.

Limitaciones y autocrítica

A pesar de sus fortalezas, el prototipo presenta varias áreas de mejora. En primer lugar, el historial de conversación se mantiene únicamente en memoria, por lo que la integración con Firestore—necesaria para dotar de continuidad pedagógica al sistema—queda pendiente como tarea prioritaria. Asimismo, si bien se definieron métricas para todos los caminos de interacción, las restricciones temporales impidieron automatizar las evaluaciones de los módulos casual, pedagógico y de denegación, dejando su validación en un formato manual. En cuanto al rendimiento, el tiempo de promedio de respuesta es de 16.48 segundos

y un P99 de 48.94 segundos, lo cual puede resultar excesivo para algunos casos, por lo que se requiere una optimización tanto del proceso de recuperación como del flujo de generación. Finalmente, el despliegue actual opera sobre HTTP, lo cual expone la información en tránsito; habilitar HTTPS con un certificado SSL/TLS es imprescindible antes de considerar una puesta en producción en un contexto educativo que maneje datos sensibles.

Gestión del proyecto

Se adoptó un enfoque ágil basado en Scrum Ban: se organizaron las tareas en un tablero de Trello, en flujos de “To Do”, “In Progress”, “To Review” y “Done”, lo que permitió ver en todo momento el estado de cada actividad y priorizar con flexibilidad. Se llevaron a cabo reuniones semanales internas para compartir avances, ajustar prioridades y resolver bloqueos. Paralelamente, se programaron sesiones puntuales con las tutoras y usuarias expertas para validar prototipos tempranos, refinar requisitos y asegurar que la experiencia de usuario respondiera a sus necesidades reales. Este esquema colaborativo —combinando tableros visuales en Trello, ciclos cortos de trabajo y despliegues constantes— garantizó que el proyecto se mantuviera alineado con las expectativas de todos los referentes y avanzar paso a paso hacia la versión final, con trazabilidad total y alta capacidad de adaptación a medida que el alcance iba evolucionando.

Conclusión general

En conjunto, el proyecto demostró la viabilidad técnica y pedagógica de un chatbot basado en recuperación aumentada por generación especializado que brinda respuestas fundamentadas y fomenta el aprendizaje activo de la inteligencia artificial. Se cumplió el objetivo general, se alcanzaron los cuatro objetivos específicos y se entregó un prototipo funcional, abierto y extensible, acompañado de un marco de evaluación replicable. Las limitaciones detectadas —persistencia de historiales, latencia, métricas faltantes y cifrado— no opacan los avances logrados, sino que definen una hoja de ruta clara para evolucionar la herramienta hacia un asistente educativo robusto y universal.

6.2. Trabajos futuros

La presente sección sintetiza las líneas de trabajo que, a juicio del equipo, contribuirán a consolidar la solución actual y a explorar nuevas oportunidades de investigación.

6.2.1. Culminación de los requerimientos pendientes

En primer lugar, resulta imprescindible completar aquellos requerimientos que quedaron fuera del alcance de la versión presentada.

RF-6. Manejo de los historiales de conversación

El manejo persistente del historial de conversaciones (RF-6) brinda al sistema la capacidad de conservar el contexto más allá de una única sesión, permitiendo continuar conversaciones sin perder el hilo temático. Esta funcionalidad no solo mejora la experiencia del usuario, sino que también potencia el valor pedagógico del chatbot, al permitir retomar intercambios previos, revisar argumentos desarrollados y construir conocimiento de manera acumulativa. Este requerimiento fue señalado por las usuarias expertas como un aspecto relevante, clave en contextos académicos por su impacto en la continuidad y profundidad del aprendizaje.

RF-5. Interacción adaptable al perfil del usuario

La adaptación del discurso al perfil declarado del usuario (RF-5) exige integrar un selector de nivel y propagar esa información al modelo encargado de generar las respuestas utilizando la información recuperada (*Informative Conversational LLM*). Actualmente, este requerimiento se encuentra implementado de forma parcial: el modelo adapta sus respuestas según el nivel de conocimiento declarado por el usuario durante la conversación. Sin embargo, la incorporación de un selector explícito permitiría registrar y reutilizar esa información de forma persistente, habilitando una adaptación más precisa y consistente a lo largo del diálogo. Más concretamente, se propone incluir instrucciones específicas en el *prompt* de sistema del *Informative Conversational LLM* según la opción seleccionada, indicando el nivel del usuario y, en consecuencia, el tipo de respuesta esperada.

Cabe mencionar que, durante las evaluaciones, algunos usuarios comentaron que percibieron poca diferencia entre las respuestas dirigidas a perfiles con distinto nivel de conocimiento, por lo que esta solución podría contribuir a fortalecer la implementación de este requerimiento. Esta mejora resulta valiosa ya que permite ofrecer explicaciones adaptadas a las expectativas de cada perfil.

RNF-1. Confidencialidad y seguridad de los datos

Por último, la adopción de *HTTPS* mediante un certificado *SSL/TLS* en el servidor resulta fundamental para garantizar la confidencialidad de las interacciones (RNF-1). Si bien se cumple con lo establecido en el requisito de que el sistema no almacena información personal de los usuarios —salvo durante la etapa de evaluación, en la que se guardaron las conversaciones temporalmente en *LangSmith* con fines de trazabilidad y análisis—, y también con el uso de herramientas de código abierto siempre que sea viable, la incorporación de *HTTPS* resulta fundamental para asegurar la confidencialidad y seguridad de los datos. De esta manera, se protegen los datos en tránsito mediante cifrado, evitando que terceros puedan interceptar la información intercambiada. Además, su adopción refuerza el compromiso del sistema con buenas prácticas de seguridad y privacidad, un requisito ineludible cuando se trabaja con datos sensibles de docentes.

6.2.2. Ampliación de la evaluación *offline*

El segundo eje consiste en extender la batería de pruebas automatizadas para cubrir la totalidad de las componentes. Incorporar métricas específicas para aquellas que aún no cuentan con mecanismos de evaluación —y así abarcar todos los posibles caminos— permitiría detectar regresiones tempranas y sostener la calidad en futuras iteraciones. A su vez, incrementar la cantidad y diversidad de ejemplos en los *datasets* de evaluación posibilitaría un análisis más profundo y representativo del comportamiento del sistema ante distintos tipos de consulta. Finalmente, integrar la ejecución de estas pruebas en el flujo de *CI*, con generación periódica de reportes, brindaría una visión longitudinal del desempeño y facilitaría la detección oportuna de desviaciones significativas.

6.2.3. Refuerzo de la accesibilidad y la usabilidad

Desde el punto de vista de experiencia de usuario, la profundización en aspectos de accesibilidad quedó fuera del alcance de esta entrega para respetar los plazos establecidos, pero sigue siendo prioritaria. El plan original contemplaba un módulo de ajustes con opciones como “Modo de lenguaje simple” (respuestas en oraciones breves y claras), alto contraste y lectura en voz alta (*text-to-speech*), de modo que docentes con distintas necesidades visuales o de comprensión pudieran aprovechar el sistema sin barreras. La incorporación de estas funcionalidades en futuras iteraciones aseguraría un chatbot pedagógicamente sólido e inclusivo.

Por otro lado, la usabilidad puede avanzar en tres frentes concretos. Primero, transformar las citas en elementos interactivos que permitan descargar el PDF o previsualizar el fragmento citado reforzaría la transparencia académica y agilizaría la navegación, un aspecto resaltado por los usuarios evaluadores 4.3. Segundo, ofrecer una interfaz sencilla para que los propios docentes puedan subir, reemplazar o eliminar documentos del repositorio RAG les permitiría conservar control directo sobre las fuentes y adaptar el corpus sin intervención del equipo de desarrollo. Tercero, mejorar la interfaz visual del chatbot —tipografías, colores consistentes, diseño adaptable (*responsive*) y conversión markdown-HTML en el contenido de los mensajes del chatbot— contribuiría a una experiencia de uso más agradable y accesible en distintos dispositivos. En esta iteración, el diseño visual no fue una prioridad, ya que el foco estuvo puesto en la investigación y en el desarrollo del *backend* para garantizar un sistema funcional y útil. Sin embargo, una interfaz visualmente atractiva genera una mejor primera impresión, resulta más estimulante para los usuarios y puede favorecer la adopción del sistema.

6.2.4. Optimización del subsistema RAG

La trayectoria de recuperación —núcleo factual del chatbot— ha mostrado un desempeño adecuado en las pruebas actuales, alcanzando niveles de relevancia y cobertura consistentes con el volumen y la heterogeneidad del corpus

utilizado. A recordar: un *Context Relevancy* de 0.58, un *Context Recall* de 0.73 y un *Context Recall* ponderado de 0.47. Sin embargo, al contemplar un despliegue con un número significativamente mayor de documentos, con distinta longitud, estructura y densidad semántica, resulta pertinente anticipar nuevas líneas de investigación.

Revisión de la estrategia de *Chunking*

En primer lugar, se podría revisar la estrategia de *chunking* utilizada durante la etapa de preprocesamiento, ya que diversos factores pueden afectar su eficacia. La densidad semántica —es decir, la cantidad y variedad de ideas contenidas en un mismo bloque de texto— puede variar considerablemente entre documentos. Esto implica que una estrategia de *chunking* previamente eficaz podría volverse subóptima ante textos con límites temáticos más difusos. Por ejemplo, un mismo párrafo puede contener muchas ideas distintas, y si el algoritmo de *chunking* no las separa adecuadamente, corre el riesgo de obtener fragmentos que mezclan conceptos irrelevantes para una consulta determinada, disminuyendo el *Context Relevancy*. A su vez, si un fragmento se corta prematuramente y no incluye suficiente información clave, puede no cubrir adecuadamente la respuesta esperada, reduciendo el *Context Recall*.

Del mismo modo, la estructura y el formato de los documentos también inciden en la estrategia de *chunking*. Por ejemplo, fuentes tan distintas como un libro de texto y un grafo de conocimiento pueden requerir fragmentaciones de distinto tamaño, dada la diferencia en la forma de organizar la información. Mientras los libros de texto contienen pasajes extensos organizados por temas, los grafos se componen de entidades breves y relaciones entre ellas, por lo que aplicar la misma estrategia de *chunking* en ambos casos puede ser ineficiente. En estos escenarios, puede ser conveniente utilizar algoritmos y configuraciones específicas según el tipo de fuente (Zhong, Liu, Cui, Zhang, y Qin, 2025). Asimismo, al trabajar con documentos estructurados en formatos como *Markdown*, *LaTeX*, *HTML*, resulta especialmente útil emplear métodos de segmentación sensibles al contenido (*Content-Aware Splitting*). Esta estrategia respeta la estructura del texto y las divisiones explícitas como encabezados, bloques de código o tablas, lo que evita que un mismo fragmento contenga diferentes tipos de contenido. Así, se preserva la coherencia, evitando que un mismo fragmento incluya, por ejemplo, un bloque de código y parte de una tabla simultáneamente (Safjan, 2023).

En este sentido, sería pertinente comparar el *chunking* semántico vigente con otros algoritmos, o bien ajustar el parámetro de umbral, evaluando cómo varían las métricas de *Context Recall* y *Context Relevancy* en cada caso.

Experimentación con índices vectoriales

A nivel de índice, Milvus ofrece distintas estructuras como *HNSW*, *FLAT*, *IVF FLAT*, entre otras, cada una con parámetros específicos que pueden ajustarse para optimizar el *trade-off* entre calidad de recuperación y latencia (Milvus,

s.f.). Por ejemplo, en el caso de *HNSW*, el parámetro *ef* permite controlar ese equilibrio: valores más altos tienden a mejorar la precisión de los resultados, pero también aumentan el tiempo de respuesta. En la versión actual del sistema, los tiempos de respuesta de milvus al recuperar información rondaron los 500 milisegundos —lo cual es aceptable dado que no se priorizó el rendimiento—, pero reducir esa latencia será esencial en escenarios de escala o en contextos donde se requiera mayor eficiencia.

Experimentación con algoritmos de búsqueda

También se podría experimentar con distintos algoritmos de búsqueda y sus parámetros. En el caso de la búsqueda con *MMR* (Maximal Marginal Relevance), por ejemplo, se podría variar el valor de *k*, que define la cantidad final de documentos seleccionados, y otros parámetros internos como *fetch_k*, que afectan cómo se construye esa selección final. Ajustar estos valores permite explorar diferentes equilibrios entre relevancia y diversidad, lo que puede influir en la calidad de las respuestas generadas.

Modularización de la etapa de generación del RAG

En lo que respecta a la etapa de generación, se podría experimentar con un flujo de procesamiento de dos fases: un “Filter LLM” que elimine fragmentos recuperados que resulten irrelevantes y un “Redactor LLM” que formule la respuesta exclusivamente a partir de los pasajes filtrados. Este enfoque implicaría sustituir o descomponer la componente *Response Generator LLM* del RAG de la solución actual en estas dos unidades especializadas. De esta manera, se mantiene la línea de componentes especializados y se facilita el diagnóstico de cuellos de botella en el rendimiento del sistema, ya que cada etapa puede ser evaluada y optimizada por separado. Además, el módulo de filtrado debería ser capaz de identificar cuándo ninguno de los fragmentos recuperados aporta información relevante, informándolo al módulo de redacción para que pueda manejar adecuadamente esta situación, evitando generar respuestas sin respaldo en la base de conocimiento.

Esta arquitectura modular también ofrece un punto de partida natural para explorar enfoques avanzados como el *re-ranking*, ya que se ha demostrado que la incorporación de una etapa previa de filtrado puede mejorar significativamente el rendimiento del *re-ranking*, incluso al utilizar modelos más pequeños y accesibles (Nouriinanloo y Lamothe, 2024).

Aplicación de otras técnicas avanzadas

La técnica de *re-ranking* permitiría mantener los pasajes más útiles de todos los recuperados. En la solución actual, todos los pasajes obtenidos durante la etapa de recuperación se entregan al LLM generador de respuestas del RAG, lo cual puede introducir ruido si muchos de ellos no están relacionados con la consulta. A pesar de los avances en los métodos de recuperación, es frecuente que

algunos pasajes recuperados resulten irrelevantes. Además, dado que el índice vectorial se basa únicamente en la similitud de *embeddings*, el *re-ranking* permitiría reordenar los pasajes de acuerdo con su relevancia frente a la pregunta concreta del usuario, capturando matices sutiles que podrían pasar desapercibidos en una recuperación puramente vectorial.

Además, podrían utilizarse técnicas de filtrado o *re-ranking* en caso de implementar el requerimiento de adaptación al perfil del usuario mediante un selector que permita obtener el nivel de conocimiento. En ese contexto, estas técnicas permitirían descartar pasajes que no se ajusten al nivel seleccionado, eliminando, por ejemplo, fragmentos demasiado técnicos si se ha elegido un nivel bajo o principiante, o bien descartando pasajes excesivamente genéricos si se ha seleccionado un nivel avanzado. Esto daría lugar a respuestas más apropiadas para cada perfil y podría contribuir a cumplir este requerimiento de manera más satisfactoria.

Por último, enfoques como *Iter-Ret-Gen* (Shao y cols., 2023), que integran recuperación y generación de forma iterativa para reforzar mutuamente ambos procesos, podrían ser útiles al escalar la solución a repositorios más grandes. Este tipo de estrategias evita depender de una única recuperación: el modelo genera una respuesta preliminar a partir del primer conjunto de documentos recuperados y la utiliza, junto con la consulta original, para realizar una nueva recuperación. Este ciclo puede repetirse tantas veces como sea necesario. De este modo, se captura mejor la intención del usuario y se superan las limitaciones de una recuperación única, mejorando la calidad de la generación final.

6.2.5. Mejoras en la extracción y almacenamiento de metadatos para el citado

Para que las respuestas incluyan referencias completas es imprescindible que cada *chunk* conserve metadatos de autoría, fecha y contexto desde la ingestión. Dado que la extracción automática aún falla cuando los documentos carecen de estructura editorial rigurosa, se propone un flujo híbrido: detección asistida por IA seguida de una revisión manual ligera que subsane omisiones y garantice la fidelidad de las citas.

Además, para optimizar el almacenamiento y evitar la duplicación innecesaria de información, en lugar de que cada *chunk* guarde de forma completa todos los metadatos (autor, fecha, título del documento, etc.), se podría almacenar únicamente un identificador único del documento asociado. Luego, se mantendría una tabla o base de datos externa donde se asocian estos identificadores con el resto de la información del documento. Esto permite mantener la referencia precisa y completa sin sobrecargar los *chunks* con datos redundantes, facilitando además actualizaciones y mantenimientos futuros.

6.2.6. Ampliación del corpus técnico

Algunas respuestas quedan limitadas por la escasez de documentos especializados. Priorizar la incorporación de manuales, artículos y otros documentos de

la temática abordada permitiría brindar explicaciones más precisas y ajustadas a distintos niveles de conocimiento.

Además, enriquecer el corpus con ejemplos concretos, dinámicas didácticas y actividades prácticas útiles para la enseñanza favorecería la generación de respuestas más aplicables al aula.

6.2.7. Investigaciones y proyecciones tecnológicas

Por último, se propone una agenda de investigación orientada a la sostenibilidad y evolución tecnológica del sistema, con especial énfasis en el seguimiento del ecosistema de modelos de lenguaje. Dado el ritmo acelerado de avance en este campo, resulta necesario revisar periódicamente la base tecnológica, que incluye tanto los modelos utilizados como su forma de integración. Actualmente han emergido alternativas maduras de modelos *open source* que merece la pena re-evaluar. Se sugiere analizarlos en tres dimensiones: su desempeño en tareas conversacionales, su capacidad como evaluadores automáticos (*LLM-as-a-judge*), y su rendimiento en los mismos benchmarks utilizados durante el proyecto, a fin de asegurar una comparación directa y consistente. De confirmarse un rendimiento comparable al de los proveedores comerciales, evaluar los requerimientos de software y procesamiento de uno de estos modelos, y analizar la viabilidad de alojarlo en un nodo local, permitiría reducir la dependencia externa, facilitar el cumplimiento de políticas de privacidad y evitar costos variables asociados al uso por token.

Paralelamente, la inclusión de modelos razonadores se perfila como un camino rápido para afinar tareas críticas: cotejo de citas, detección de contradicciones en el corpus y generación de repreguntas pedagógicas más precisas. Estos modelos especializados pueden integrarse tanto en la fase de respuesta como en los procesos de evaluación automática, elevando la fidelidad de la salida y reduciendo la intervención manual.

Explorar estos dos bloques —nuevos modelos open-source y razonadores dedicados— permitirá no solo mejorar el rendimiento y la seguridad del prototipo, sino también sentar las bases para futuras extensiones multi-dominio que respondan a necesidades diversas más allá del ámbito de la inteligencia artificial educativa.

Referencias

- Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., ... others (2023). GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*. <https://openai.com/research/gpt-4>. (Zero-shot benchmarks across WinoGrande, MMLU, ARC)
- Agencia Nacional de Investigación e Innovación. (2021). *PORTAL TIMBÓ: acceso a Scopus a nivel nacional y a todas las colecciones de IEEE*. <https://www.anii.org.uy/noticias/154/portal-timbo-acceso-a-scopus-a-nivel-nacional-y-a-todas-las-colecciones-de-ieee/>. (Accessed: 2025-06-30)
- Amazon Web Services. (s.f.). *What is a framework in programming and engineering?* <https://aws.amazon.com/what-is/framework/>. (Accessed: 2025-06-27)
- Amazon Web Services. (2023a). *¿Qué es la RAG (generación aumentada por recuperación)?* <https://aws.amazon.com/es/what-is/retrieval-augmented-generation/>. (Accessed: 2024-13-04)
- Amazon Web Services. (2023b). *¿Qué son los modelos de lenguaje de gran tamaño (LLM)?* <https://aws.amazon.com/es/what-is/large-language-model/>. (Accessed: 2024-31-05)
- Anthropic. (2024a). *Data input controls and audit*. <https://www.anthropic.com/uk-government-internal-ai-safety-policy-response/data-input-controls-and-audit>. (Accessed: 2024-27-05)
- Anthropic. (2024b). *Model Card and Evaluations for Claude Models*. Descargado de <https://www-cdn.anthropic.com/5c49cc247484cecf107c699baf29250302e5da70/claude-2-model-card.pdf> (5-shot MMLU results (78.5 %) CoT)
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... Amodei, D. (2020). Language models are few-shot learners. En H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, y H. Lin (Eds.), *Advances in neural information processing systems* (Vol. 33, pp. 1877–1901). Curran Associates, Inc. Descargado de https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf
- Bullwinkle, M., Pavan, P., Farley, P., y Urban, E. (2025). *Message engineering techniques*. <https://learn.microsoft.com/es-es/azure/>

- [ai-services/openai/concepts/prompt-engineering?tabs=chat#temperature-and-top-p-parameters](#). Microsoft.
- Carreira, G., Silva, L., Mendes, A. J., y Oliveira, H. G. (2022). Pyo, a chatbot assistant for introductory programming students. En *2022 international symposium on computers in education (siie)* (pp. 1–6). Descargado de https://www.researchgate.net/publication/366429462_Pyo_a_Chatbot_Assistant_for_Introductory_Programming_Students
- Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., ... Xie, X. (2024). A survey on evaluation of large language models. *ACM transactions on intelligent systems and technology*, 15(3). Descargado de <https://doi.org/10.1145/3641289> doi: 10.1145/3641289
- Chollet, F., Knoop, M., Kamradt, G., y Landers, B. (2024). Arc prize 2024: Technical report. *arXiv preprint arXiv:2412.04604*. Descargado de <https://doi.org/10.48550/arXiv.2412.04604>
- Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., y Tafjord, O. (2018). Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv preprint arXiv:1803.05457*. Descargado de <https://doi.org/10.48550/arXiv.1803.05457>
- Cloudflare. (2024). *What are embeddings in machine learning?* <https://www.cloudflare.com/learning/ai/what-are-embeddings/>. (Accessed: 2024-1-06)
- Cloudflare, Inc. (2024). *¿Qué es un modelo lingüístico (LLM) grande?* <https://www.cloudflare.com/es-es/learning/ai/what-is-large-language-model/>. (Accessed: 2024-26-05)
- Cohere. (2023). *Introducing Embed v3*. <https://cohere.com/blog/introducing-embed-v3>. (Blog post)
- Databricks Partner Academy. (2024). *Generative AI Engineering Pathway*. https://partner-academy.databricks.com/learn/learning-plans/315/generative-ai-engineering-pathway?hash=9325052ed11ac2795c800c2b2ef9bafae8a3931f&generated_by=804663. (Accessed: 2024-12-20)
- Dave Club. (2025). *Local vs. Cloud-Based LLMs: My Thoughts on Choosing the Right Approach*. <https://medium.com/dave-club/local-vs-cloud-based-llms-my-thoughts-on-choosing-the-right-approach-a5bce59263bb>. Medium. (Accessed: 2025-05-19)
- Es, S., James, J., Espinosa Anke, L., y Schockaert, S. (2024). RAGAs: Automated evaluation of retrieval augmented generation. En N. Aletras y O. De Clercq (Eds.), *Proceedings of the 18th conference of the european chapter of the association for computational linguistics: System demonstrations* (pp. 150–158). St. Julians, Malta: Association for Computational Linguistics. Descargado de <https://aclanthology.org/2024.eacl-demo.16/>
- Faiss. (2017). *Faiss documentation*. <https://faiss.ai/index.html>. (Accessed: 2025-06-17)
- Frankle, J., y Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*. Descargado

- de <https://doi.org/10.48550/arXiv.1803.03635>
- Ghodratnama, S., y Zakershaharak, M. (2023). Adapting llms for efficient, personalized information retrieval: Methods and implications. En *International conference on service-oriented computing* (pp. 17–26). Descargado de <https://doi.org/10.48550/arXiv.2311.12287>
- Gong, D. (2025). *6 Common LLM Customization Strategies Briefly Explained*. <https://towardsdatascience.com/6-common-llm-customization-strategies-briefly-explained/>. Towards Data Science.
- Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., ... others (2024). The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*. <https://doi.org/10.48550/arXiv.2407.21783>.
- Guo, Z., Jin, R., Liu, C., Huang, Y., Shi, D., Yu, L., ... others (2023). Evaluating large language models: A comprehensive survey. *arXiv preprint arXiv:2310.19736*. Descargado de <https://doi.org/10.48550/arXiv.2310.19736>
- HatchWorksAI. (2024). *Large language model use cases: One llm vs multiple models*. <https://hatchworks.com/blog/gen-ai/llm-use-cases-single-vs-multiple-models/#comparing-approaches>. (Accessed: 2025-03-15)
- Hurst, A., Lerer, A., Goucher, A. P., Perelman, A., Ramesh, A., Clark, A., ... others (2024). Gpt-4o System Card. *arXiv preprint arXiv:2410.21276*. Descargado de <https://cdn.openai.com/gpt-4o-system-card.pdf> (Zero-shot performance: EM 72.2 ± 0.6 , F1 82.7 ± 0.7 on HotpotQA)
- IBM. (2024a). *What is fine-tuning?* <https://www.ibm.com/topics/fine-tuning>. (Accessed: 2024-12-04)
- IBM. (2024b). *¿Qué son los modelos de lenguaje pequeños?* <https://www.ibm.com/think/topics/small-language-models>. (Accessed: 2025-03-23)
- Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., ... others (2024). Mixtral of experts. *arXiv preprint arXiv:2401.04088*. Descargado de <https://mistral.ai/news/mixtral-of-experts/> (Zero-shot evaluations on WinoGrande and HellaSwag)
- Jin, R., Du, J., Huang, W., Liu, W., Luan, J., Wang, B., y Xiong, D. (2024). A comprehensive evaluation of quantization strategies for large language models. En L.-W. Ku, A. Martins, y V. Srikumar (Eds.), *Findings of the association for computational linguistics: Acl 2024* (pp. 12186–12215). Bangkok, Thailand: Association for Computational Linguistics. Descargado de <https://aclanthology.org/2024.findings-acl.726/> doi: 10.18653/v1/2024.findings-acl.726
- Jouppi, N., Kurian, G., Li, S., Ma, P., Nagarajan, R., Nai, L., ... others (2023). Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. En *Proceedings of the 50th annual international symposium on computer architecture* (pp. 1–14). Descargado de <https://doi.org/10.48550/arXiv.2304.01433>
- Kamalloo, E., Dziri, N., Clarke, C., y Rafiei, D. (2023). Evaluating open-domain question answering in the era of large language models. En

- A. Rogers, J. Boyd-Graber, y N. Okazaki (Eds.), *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 5591–5606). Toronto, Canada: Association for Computational Linguistics. Descargado de <https://aclanthology.org/2023.acl-long.307/> doi: 10.18653/v1/2023.acl-long.307
- Khot, T., Trivedi, H., Finlayson, M., Fu, Y., Richardson, K., Clark, P., y Sabharwal, A. (2023). Decomposed prompting: A modular approach for solving complex tasks. En *Proceedings of the international conference on learning representations (iclr)*. Descargado de <https://doi.org/10.48550/arXiv.2210.02406>
- Kusupati, A., Bhatt, G., Rege, A., Wallingford, M., Sinha, A., Ramanujan, V., ... Farhadi, A. (2022). Matryoshka representation learning. En S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, y A. Oh (Eds.), *Advances in Neural Information Processing Systems* (Vol. 35, pp. 30233–30249). Curran Associates, Inc. Descargado de https://proceedings.neurips.cc/paper_files/paper/2022/file/c32319f4868da7613d78af9993100e42-Paper-Conference.pdf
- LanceDB. (2024). *Chunking techniques with Langchain and LlamaIndex*. <https://blog.lancedb.com/chunking-techniques-with-langchain-and-llamaindex/>. (Accessed: 2025-06-17)
- LangChain, Inc. (2024). *Build an agent*. <https://python.langchain.com/docs/tutorials/agents/>. (Accessed: 2025-03-15)
- LangChain, Inc. (2024a). *Embedding models*. https://python.langchain.com/docs/integrations/text_embedding/. (Accessed: 2024-1-05)
- LangChain, Inc. (2024b). *LLMs - Features (natively supported)*. <https://python.langchain.com/docs/integrations/llms/>. (Accessed: 2024-1-05)
- Langchain, Inc. (2024). *Query analysis*. <https://python.langchain.com/docs/concepts/retrieval/>. (Accessed: 2025-06-21)
- LangChain, Inc. (2024a). *Structured Outputs — LangChain Documentation*. https://python.langchain.com/docs/concepts/structured_outputs/. (Accessed: 2025-05-19)
- LangChain, Inc. (2024b). *Tool Calling — LangChain Documentation*. https://python.langchain.com/docs/concepts/tool_calling/. (Accessed: 2025-05-19)
- LangChain, Inc. (2025a). *Get started with LangSmith*. <https://docs.smith.langchain.com/>. (Accessed: 2025-06-28)
- LangChain, Inc. (2025b). *How to split text based on semantic similarity*. https://python.langchain.com/docs/how_to/semantic-chunker/. (Accessed: 2025-06-17)
- LangChain, Inc. (2025c). *Recursivecharactertextspliter*. https://python.langchain.com/api_reference/text_splitters/character/langchain_text_splitters.character.RecursiveCharacterTextSplitter.html. (Accessed: 2025-06-17)
- LangChain, Inc. (2025d). *Semanticchunker*. https://python.langchain.com/api_reference/experimental/text_splitter/

- [langchain.experimental.text_splitter.SemanticChunker.html](#). (Accessed: 2025-06-17)
- LangChain, Inc. (2025e). *Vector stores*. <https://python.langchain.com/docs/integrations/vectorstores/>. (Accessed: 2025-06-28)
- LangChain, Inc. (2025f). *Why LangChain?* https://python.langchain.com/docs/concepts/why_langchain/. (Accessed: 2025-16-03)
- Levesque, H., Davis, E., y Morgenstern, L. (2011). The winograd schema challenge. En *Proceedings of the thirteenth international conference on principles of knowledge representation and reasoning*. Descargado de <https://cdn.aaai.org/ocs/4492/4492-21843-1-PB.pdf>
- Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., y Liang, P. (2024). Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12, 157-173. Descargado de <https://doi.org/10.1162/tac1.a.00638> doi: 10.1162/tac1.a.00638
- LlamaIndex. (2024). *Efficient Chunk Size Optimization for RAG Pipelines with LlamaCloud*. <https://www.llamaindex.ai/blog/efficient-chunk-size-optimization-for-rag-pipelines-with-llamacloud>. (Accessed: 2025-06-17)
- Menick, J., Lu, K., Zhao, S., Wallace, E., Ren, H., Hu, H., ... Such, F. P. (2024). *GPT-4o mini: advancing cost-efficient intelligence*. <https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/>. OpenAI. (Zero-shot distractor setting accuracy 58% on HotpotQA)
- Meta AI. (2023a). *Introducing code llama, a state-of-the-art large language model for coding*. <https://ai.meta.com/blog/code-llama-large-language-model-coding/>. (Accessed: 2025-06-23)
- Meta AI. (2023b). *Llama 2: Open Foundation and Fine-Tuned Chat Models*. <https://ai.meta.com/research/publications/llama-2-open-foundation-and-fine-tuned-chat-models/>. (Accessed: 2024-27-05)
- Milvus. (s.f.). *Explanation of the indexes*. <https://milvus.io/docs/es/index-explained.md>. (Accessed: 2025-06-17)
- Milvus. (2023). *Attu*. <https://milvus.io/docs/v2.0.x/attu.md>. (Accessed: 2025-06-17)
- Muennighoff, N., Tazi, N., Magne, L., y Reimers, N. (2023). Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*. Descargado de <https://doi.org/10.48550/arXiv.2210.07316>
- Nayak, P. (2024). *Semantic chunking for rag*. <https://medium.com/the-ai-forum/semantic-chunking-for-rag-f4733025d5f5>. Medium. (Accessed: 2025-06-17)
- Nomic AI. (2024). *Introducing Nomic Embed: A Truly Open Embedding Model*. <https://www.nomic.ai/blog/posts/nomic-embed-text-v1>. (Blog post)
- Nouriinanloo, B., y Lamothe, M. (2024). Re-ranking step by step: Investigating pre-filtering for re-ranking with large language models. *arXiv*

- preprint *arXiv:2406.18740*. Descargado de <https://doi.org/10.48550/arXiv.2406.18740>
- NVIDIA. (2024). *Structured Generation - NVIDIA Docs*. <https://docs.nvidia.com/nim/large-language-models/latest/structured-generation.html>. (Accessed: 2025-03-10)
- Okonkwo, C. W., y Ade-Ibijola, A. (2020). Python-bot: A chatbot for teaching python programming. *Engineering Letters*, 29(1). Descargado de https://www.researchgate.net/publication/349574948_Python-Bot_A_Chatbot_for_Teaching_Python_Programming
- Open AI. (2023). *GPT-4o. Fast, intelligent, flexible GPT model*. <https://platform.openai.com/docs/models/gpt-4o>.
- Open AI. (2024). *GPT-4.1. Flagship GPT model for complex tasks*. <https://platform.openai.com/docs/models/gpt-4.1>.
- OpenAI. (2023a). *Api data usage policies*. <https://openai.com/policies/api-data-usage-policies/>. (Accessed: 2025-03-15)
- OpenAI. (2023b). *Enterprise privacy at OpenAI*. <https://openai.com/enterprise-privacy/>. (Accessed: 2024-27-05)
- OpenAI. (2023c). *Gpt-3.5 turbo*. <https://platform.openai.com/docs/models/gpt-3.5-turbo>. (Zero-shot “answer-only” configuration)
- OpenAI. (2024). *New embedding models and api updates*. <https://openai.com/index/new-embedding-models-and-api-updates/>. (Blog post)
- PingCAP. (2024). *Analyzing Performance Gains in OpenAI’s Text-Embedding-3-Small*. <https://www.pingcap.com/article/analyzing-performance-gains-in-openais-text-embedding-3-small/>. (Blog post)
- Qu, R., Tu, R., y Bao, F. S. (2025). Is semantic chunking worth the computational cost? En L. Chiruzzo, A. Ritter, y L. Wang (Eds.), *Findings of the association for computational linguistics: Naac1 2025* (pp. 2155–2177). Albuquerque, New Mexico: Association for Computational Linguistics. Descargado de <https://aclanthology.org/2025.findings-naacl.114/> doi: 10.18653/v1/2025.findings-naacl.114
- Rony, M. R. A. H., Süß, C., Bhat, S. R., Sudhi, V., Schneider, J., Vogel, M., ... Sahoo, S. (2023). Carexpert: Leveraging large language models for in-car conversational question answering. *arXiv preprint arXiv:2310.09536*. Descargado de <https://doi.org/10.48550/arXiv.2310.09536>
- Safjan, K. (2023). *From fixed-size to nlp chunking - a deep dive into text chunking techniques*. <https://safjan.com/from-fixed-size-to-nlp-chunking-a-deep-dive-into-text-chunking-techniques/>. (Accessed: 2025-06-15)
- Sanh, V., Webson, A., Raffel, C., Bach, S. H., Sutawika, L., Alyafeai, Z., ... others (2022). Multitask prompted training enables zero-shot task generalization. En *Proceedings of the international conference on learning representations (iclr)*. Descargado de <https://doi.org/10.48550/arXiv.2110.08207>
- Sastre, I. (2024). Búsqueda de respuestas en documentos institucionales utilizando grandes modelos de lenguaje. *Tesis de grado. Universidad de la República (Uruguay), Facultad de Ingeniería*. Descarga-

- do de <https://www.colibri.udelar.edu.uy/jspui/bitstream/20.500.12008/46066/1/Sas24.pdf>
- SBERT. (2024). *all-mpnet-base-v2*. https://www.sbert.net/docs/sentence_transformer/pretrained_models.html.
- Schwaber-Cohen, R. (2023). *Vector Embeddings for Developers: The Basics*. <https://www.pinecone.io/learn/vector-embeddings-for-developers/>. Pinecone. (Accessed: 2024-1-06)
- Shao, Z., Gong, Y., Shen, Y., Huang, M., Duan, N., y Chen, W. (2023). Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. *arXiv preprint arXiv:2305.15294*. Descargado de <https://doi.org/10.48550/arXiv.2305.15294>
- Siragusa, I., y Pirrone, R. (2023). Conditioning chat-gpt for information retrieval: The unipa-gpt case study. En *M4ai@ ai*ia*. Rome, Italy. Descargado de <https://ceur-ws.org/Vol-3551/>
- Srivatsa, H. (2024). *Fine-tuning versus RAG in Generative AI Applications Architecture*. <https://harsha-srivatsa.medium.com/fine-tuning-versus-rag-in-generative-ai-applications-architecture-d54ca6d2acb8>. Medium. (Accessed: 2024-13-04)
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. En I. Guyon y cols. (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. Descargado de https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
- Wang, L., Yang, N., Huang, X., Yang, L., Majumder, R., y Wei, F. (2024). Multilingual E5 Text Embeddings: A Technical Report. *arXiv preprint arXiv:2402.05672*. Descargado de <https://doi.org/10.48550/arXiv.2402.05672>
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., ichter, b., Xia, F., ... Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. En S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, y A. Oh (Eds.), *Advances in neural information processing systems* (Vol. 35, pp. 24824–24837). Curran Associates, Inc. Descargado de https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf
- Yu, H., Gan, A., Zhang, K., Tong, S., Liu, Q., y Liu, Z. (2025). Evaluation of retrieval-augmented generation: A survey. En W. Zhu y cols. (Eds.), *Big data* (pp. 102–120). Singapore: Springer Nature Singapore. Descargado de <https://doi.org/10.48550/arXiv.2405.07437>
- Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., y Choi, Y. (2019). Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*. Descargado de <https://doi.org/10.48550/arXiv.1905.07830>
- Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., ... Stoica, I. (2023). Judging llm-as-a-judge with mt-bench and chatbot arena. En A. Oh, T. Naumann, A. Globerson, K. Saenko,

M. Hardt, y S. Levine (Eds.), *Advances in neural information processing systems* (Vol. 36, pp. 46595–46623). Curran Associates, Inc. Descargado de https://proceedings.neurips.cc/paper_files/paper/2023/file/91f18a1287b398d378ef22505bf41832-Paper-Datasets_and_Benchmarks.pdf

Zhong, Z., Liu, H., Cui, X., Zhang, X., y Qin, Z. (2025). Mix-of-granularity: Optimize the chunking granularity for retrieval-augmented generation. En O. Rambow, L. Wanner, M. Apidianaki, H. Al-Khalifa, B. D. Eugenio, y S. Schockaert (Eds.), *Proceedings of the 31st international conference on computational linguistics* (pp. 5756–5774). Abu Dhabi, UAE: Association for Computational Linguistics. Descargado de <https://aclanthology.org/2025.coling-main.384/>

Anexo A

Investigación sobre Grandes Modelos de Lenguaje

A continuación se presenta la investigación realizada sobre Grandes Modelos de Lenguaje que respalda el contenido teórico resumido presentado en el capítulo 2 de Conceptos Preliminares y la justificación de la necesidad de utilizar modelos grandes a través de APIs al momento de la investigación.

Los Grandes Modelos de Lenguaje son sistemas computacionales entrenados para predecir la probabilidad de una secuencia de palabras, lo que le permite abordar una amplia gama de tareas de procesamiento del lenguaje natural como la generación de texto, la traducción automática y la corrección gramatical. Para ello, emplean tecnologías de aprendizaje automático, en particular el aprendizaje profundo, para comprender la estructura y el significado del lenguaje. En los LLMs, la calidad de los datos de entrenamiento es fundamental, ya que determina qué tan bien el modelo aprenderá el lenguaje natural. Por esto, los desarrolladores suelen optar por conjuntos de datos cuidadosamente seleccionados. (Chang y cols., 2024; Cloudflare, Inc, 2024)

Como su nombre indica, los Grandes Modelos de Lenguaje (LLMs) son una categoría de modelos de lenguaje caracterizados por su gran escala y capacidad de procesamiento. La escala o tamaño de un modelo de lenguaje se determina principalmente por la cantidad de parámetros, que son las variables internas que se ajustan durante el entrenamiento para optimizar sus predicciones. Estos parámetros, que incluyen *pesos* y *sesgos*, influyen en el rendimiento del modelo y en su capacidad para generar texto de manera coherente. Mientras que los modelos más pequeños pueden tener desde miles hasta millones de parámetros, los LLMs pueden alcanzar desde cientos de millones hasta billones, lo que les permite capturar patrones más complejos en el lenguaje. (IBM, 2024b)

A.0.1. Categorías de LLMs

Existen diversos tipos de modelos que varían en tamaño (cantidad de parámetros), lo cual puede influir en su rendimiento, facilidad de uso y otras características claves. Se presentan modelos de diversos tamaños, a la fecha de la redacción de esta sección se decidió no incluir en el análisis modelos menores a 7B de parámetros pues a la hora de evaluarlos empíricamente se evidencia que resultan inutilizables.

Modelos pequeños (7B - 13B) En esta categoría se encuentran modelos como Gemma 8B, Mixtral 7B, Llama 2 7B, Llama 3 7B, así como algunos modelos con mayor cantidad de parámetros, como Llama 3 13B. Este tipo de modelos presenta una clara ventaja sobre los modelos más grandes: requieren menos recursos de hardware. Pueden ser ejecutados por computadoras modernas sin necesidad de hardware especializado y extremadamente costoso. Sin embargo, ésta categoría de modelos presenta una capacidad limitada de razonamiento y restricciones en cuanto al idioma, generalmente consecuencia de que la mayoría de los datos de entrenamiento están en inglés. Como se muestra en el artículo de Llama 2 (Meta AI, 2023b), tan solo un 0.13 % de esos datos están en Español.

Modelos medianos (50B - 70B) En esta categoría se pueden encontrar modelos como Mixtral 8x7B, Llama 2 70B, Llama 3 70B, Falcon 40B, entre otros. Estos modelos muestran capacidades de razonamiento superiores a las de los modelos pequeños y una mayor habilidad para manejar el idioma español. En esta categoría de modelos, es aún común encontrar modelos *open-weights* (se pueden descargar sus pesos y correrlos) como lo son Mixtral o Falcon, pero los requerimientos de hardware son muy elevados. Por ejemplo, para ejecutar una sola instancia de Mixtral 8x7B, se requieren aproximadamente 100 GB de vRAM, según la web oficial de MistralAI, con lo cual se hace difícil escalar el servicio a múltiples usuarios concurrentes.

Modelos Grandes En esta categoría se incluyen modelos que van desde los 200B de parámetros como Claude 2.1 hasta modelos de 1.8T de parámetros como GPT-4, que tienen sus ventajas y desventajas. Entre las principales ventajas de estos modelos se encuentran sus altas precisiones y su capacidad de manejar de manera fluida el español. Sin embargo, estos grandes modelos también presentan desventajas. Principalmente, son lentos y tienen un alto costo por inferencia, son modelos privativos disponibles solo a través de APIs, no son *open-weights* ni de código abierto, lo cual plantea importantes desafíos en términos de gobernanza de datos. En relación a esto último, algunos proveedores como OpenAI y Anthropic han tomado cartas en el asunto, permitiendo que el consumo de sus servicios que se haga mediante una API, no forme parte de ningún entrenamiento ni se almacene durante más de treinta días (OpenAI, 2023b; Anthropic, 2024a). Otro gran punto a favor de estos servicios es la escalabilidad que presentan. Al contar con grandes centros de datos dedicados

exclusivamente a la inferencia, tienen la capacidad de escalar mas allá de lo que un solo proyecto puede requerir.

A.0.2. Arquitectura de los LLMs

El modelo **Transformer** es una arquitectura de redes neuronales introducida en el artículo *Attention is All You Need* (Vaswani y cols., 2017), que se basa en el uso exclusivo de mecanismos de atención, eliminando la dependencia de estructuras secuenciales como las redes neuronales recurrentes (RNN) o convolucionales (CNN).

Fundamentos del Transformer

La idea central del Transformer es el **mecanismo de self-attention**. Este mecanismo permite que cada elemento (o token) de la secuencia de entrada evalúe y asigne distintos grados de relevancia a los demás elementos. Para ello, cada token se transforma en tres representaciones: *query* (consulta), *key* (clave) y *value* (valor). Mediante la operación conocida como **scaled dot-product attention**, se calcula la similitud entre los *query* y las *keys*, obteniendo así una distribución de atención que se utiliza para ponderar los *values*. Este proceso permite al modelo capturar relaciones contextuales de forma paralela, lo que mejora tanto la eficiencia como la capacidad de aprendizaje a gran escala. La Figura A.0.2 ilustra esta arquitectura.

Multi-Head Attention y Codificación Posicional

Una innovación clave es el uso de **multi-head attention**. En lugar de aplicar un único mecanismo de atención, el Transformer utiliza múltiples “cabezas” de atención, cada una de las cuales procesa la información de manera independiente. Luego, los resultados se concatenan y se integran, lo que permite al modelo captar diferentes tipos de relaciones y complejidades lingüísticas en los datos de entrada.

Dado que la arquitectura Transformer carece de un mecanismo intrínseco para capturar el orden secuencial de los tokens, se incorporan **codificaciones posicionales** (positional encodings). Estas codificaciones añaden información sobre la posición relativa de cada token en la secuencia, asegurando que el modelo pueda mantener el sentido del orden en la información procesada.

Arquitectura de Codificador-Decodificador

La estructura clásica del Transformer se organiza en dos bloques principales:

- **Codificador (Encoder):** Procesa la secuencia de entrada y genera representaciones contextuales enriquecidas para cada token.
- **Decodificador (Decoder):** Utiliza las representaciones generadas por el codificador para producir la salida, token a token, empleando meca-

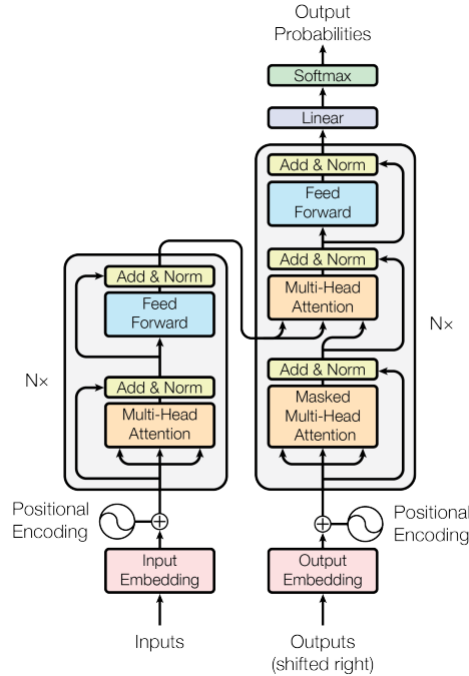


Figura A.1: Diagrama del modelo de arquitectura de Transformer. (Figura tomada de: (Vaswani y cols., 2017))

nismos de atención tanto sobre la entrada como sobre la salida generada previamente.

En el contexto de los **Modelos de Lenguaje de Gran Escala (LLMs)**, se suelen emplear variantes del Transformer que han sido entrenadas con grandes volúmenes de datos. Estas variantes son capaces de aprender patrones complejos y generar respuestas coherentes y contextualizadas, lo que les permite desempeñarse de forma efectiva en tareas de generación de texto, traducción, resumen y otras aplicaciones del procesamiento del lenguaje natural.

La propuesta de (Vaswani y cols., 2017) marcó un antes y un después en el campo, al demostrar que los mecanismos de atención podían sustituir completamente las arquitecturas secuenciales tradicionales, posibilitando modelos más capaces y escalables.

A.0.3. Modelos de Embeddings

Un aspecto fundamental en el funcionamiento de los grandes modelos de lenguaje (LLMs) es su mecanismo de representación de datos. Para que los algoritmos de aprendizaje automático puedan identificar patrones, generar pre-

dicciones y ejecutar decisiones, los datos de entrada deben codificarse en un formato compatible con operaciones matemáticas. Esto implica transformar la información (ya sea texto, imágenes u otros tipos de entrada), en representaciones numéricas estructuradas, como vectores o matrices. Estas representaciones permiten a los sistemas computacionales procesar y analizar los datos de manera eficiente, facilitando así el aprendizaje automático.

Los métodos tradicionales de representación de palabras en el aprendizaje automático, como el 'one-hot encoding' y el 'bag-of-words' proporcionaron las primeras soluciones para transformar texto en datos numéricos. Sin embargo, presentaban limitaciones significativas en cuanto a la escalabilidad en vocabularios grandes y la capacidad de capturar relaciones entre las palabras, como la similitud de significado. Con el avance de la tecnología, se desarrollaron métodos más sofisticados, como los *embeddings*, que superaron estas limitaciones. Los embeddings son representaciones numéricas de palabras en un espacio vectorial de alta dimensión, que capturan características semánticas y sintácticas de las palabras y los textos. Esto permite a los modelos de lenguaje procesar el texto de manera mucho más efectiva y precisa. ([Amazon Web Services, 2023b](#))

La representación mediante embeddings hace que las palabras con significados semánticamente similares o con otras relaciones estén cerca unas de otras en el espacio vectorial. Por ejemplo, las palabras 'rey', 'reina', 'hombre' y 'mujer' estarán distribuidas en este espacio de manera que la distancia entre 'rey' y 'reina' sea pequeña, indicando una alta similitud semántica, ya que ambos son roles monárquicos de diferente género. De manera similar, la distancia entre 'hombre' y 'mujer' será pequeña, ya que ambos son términos relacionados con el género humano. Además, la relación entre estas distancias puede capturar analogías como 'rey es a reina como hombre es a mujer'. Para medir la similitud entre embeddings, se utilizan varias métricas, siendo algunas de las más comunes la similitud coseno, la distancia euclídea, y el producto punto. Estas métricas permiten cuantificar qué tan similares o diferentes son las palabras representadas en el espacio vectorial.

Además de representar palabras, los embeddings permiten la representación de textos completos, capturando el significado de las palabras en su contexto. Esto es especialmente importante, ya que una palabra puede tener distintos significados según el contexto en el que se encuentre. Mientras que los embeddings a nivel de palabra no logran capturar esta información contextual, los embeddings de oraciones y textos pueden capturar estos matices de significado.

Para convertir el lenguaje natural en embeddings, los LLMs utilizan técnicas de aprendizaje profundo, específicamente redes neuronales del tipo *transformer* [A.0.2](#). Estas redes, conocidas como **modelos de embeddings**, transforman datos en bruto, como un texto en lenguaje natural, en embeddings que capturan y representan el contexto completo del dato, proporcionando una interpretación precisa y rica del lenguaje. Este proceso de generación de embeddings implica entrenar una red neuronal con una gran cantidad de datos etiquetados. Una vez entrenada, esta red puede convertir datos en bruto en representaciones vectoriales. Un ejemplo destacado de un modelo de embeddings es Word2Vec, que se utiliza ampliamente en diversas tareas basadas en texto. Todo este proceso

realizado por los modelos de embeddings, que implica el procesamiento de la entrada para representarla como un vector que captura semántica y relaciones gramaticales, permite que los LLMs apliquen este conocimiento del idioma para producir resultados únicos y precisos en contexto. (Cloudflare, 2024)

A su vez, los embeddings no se limitan solo a la representación de texto, sino que también pueden representar imágenes y audio, que pueden ser utilizados en modelos de aprendizaje automático. Incluso se puede aprovechar esta característica para desarrollar sistemas que realicen búsquedas semánticas, no solo en textos, sino también en contenido visual y auditivo. De esta manera, los embeddings facilitan la búsqueda y comparación de documentos, imágenes o archivos de audio similares.

Los embeddings son herramientas poderosas que permiten a los modelos de lenguaje comprender y procesar el texto de manera efectiva, produciendo resultados precisos y semanticamente relevantes. Su capacidad para capturar relaciones semánticas y contextuales los hace fundamentales, no solo como insumo para los modelos de lenguaje, sino también en una amplia gama de aplicaciones. Esto incluye búsquedas semánticas, sistemas de pregunta-respuesta, búsqueda de imágenes y audio, sistemas de recomendación y detección de anomalías. (Schwaber-Cohen, 2023)

Matryoshka Representation Learning

Matryoshka Representation Learning (MRL) (Kusupati y cols., 2022) codifica dentro de un único vector de dimensión d una jerarquía anidada de representaciones: cada prefijo $z_{1:m}$, con $m \in 8, 16, \dots, d$, es una representación autosuficiente, mientras que el vector completo preserva la máxima fidelidad. El entrenamiento optimiza únicamente $O(\log d)$ de esos prefijos, de modo que la información se difunde de manera continua a lo largo de todo el embedding, eliminando la necesidad de mantener modelos independientes para distintas dimensionalidades y permitiendo ajustar dinámicamente el coste de cómputo y almacenamiento.

La adaptación a modelos de texto resulta directa: en BERT, la variante MRL reutiliza la matriz de embeddings como clasificador de *masked-language modeling*, de forma que no añade sobrecarga en inferencia. Los resultados muestran que esta versión conserva la precisión del modelo base en todas las escalas; por ejemplo, en el conjunto de validación de Wikipedia + BooksCorpus, la representación de 384 dimensiones de BERT-MRL se mantiene a sólo 0.2 pp del baseline y el vector completo de 768 dimensiones es virtualmente idéntico.

La versatilidad del embedding habilita estrategias de recuperación adaptativa. Una implementación sencilla acorta la búsqueda inicial con un prefijo reducido (por ejemplo 16 dimensiones) y reordena los candidatos con el vector íntegro, requiriendo hasta 128 veces menos FLOPs y 14 veces de aceleración real sin degradar la precisión respecto a una búsqueda que usa directamente el embedding completo. En tareas de recuperación textual a gran escala, MRL ofrece así un equilibrio eficiente entre velocidad, memoria y exactitud.

Benchmark MTEB y Comparación de Modelos de Embeddings

El *Massive Text Embedding Benchmark* (MTEB) fue propuesto por (Muennighoff, Tazi, Magne, y Reimers, 2023) con el objetivo de ofrecer una evaluación amplia y estandarizada de modelos de embeddings en ocho categorías de tareas que incluyen, entre otras, *retrieval*, clasificación y clustering. MTEB abarca 58 datasets, cubre 112 idiomas y reporta métricas homogéneas, por ejemplo, $nDCG@10$ o MAP , que permiten comparar calidad de recuperación entre modelos sin depender de fine-tuning adicional (Muennighoff y cols., 2023).

A.0.4. Ventana de contexto

Otro aspecto esencial en los modelos basados en transformadores es la denominada *ventana de contexto*, es decir, la longitud máxima de secuencia que el modelo puede procesar en una sola pasada. Dicho de otro modo, existe un límite en la cantidad de tokens (en este contexto significa palabra o parte significativa de palabra¹) que se pueden proporcionar simultáneamente como entrada al modelo, lo cual repercute directamente en su capacidad para manejar textos extensos o combinar múltiples documentos de gran tamaño. Por ejemplo, ciertos modelos populares tienen un alcance de apenas unos pocos miles de tokens, lo que puede dificultar la inclusión de todos los datos relevantes en una sola consulta.

La limitación en la ventana de contexto se explica en gran parte por la complejidad computacional y de memoria asociada al mecanismo de *self-attention*, utilizado en los *Transformers*. Conforme la secuencia de entrada crece, la cantidad de operaciones necesarias para relacionar cada token con todos los demás se dispara de manera cuadrática (véase A.0.4), encareciendo tanto el tiempo de ejecución como el consumo de recursos. En consecuencia, las implementaciones habituales suelen conservar una ventana de contexto fija y acotada para mantener un balance entre el rendimiento y la viabilidad del entrenamiento y la inferencia.

Esta restricción incide notablemente en tareas que requieren integrar un volumen considerable de información, como la recuperación de documentos extensos o la suma de múltiples fuentes textuales en consultas de *Generative QA*. Un modelo con ventana de contexto reducida, por ejemplo, puede verse forzado a segmentar o resumir en exceso la información, con el riesgo de perder detalles clave. Por otro lado, se han investigado métodos y optimizaciones —tanto a nivel de arquitectura como de algoritmos de cálculo de la atención— con el fin de ampliar este límite y hacer más eficiente el procesamiento de secuencias largas (Sastre, 2024).

En el marco de esta tesis, la importancia de la ventana de contexto radica en que, a la hora de *personalizar el conocimiento o el comportamiento* de un LLM (concepto desarrollado en la sección 2.2), se exploran estrategias como el

¹Existen diversos algoritmos para *tokenizar* textos, la mayoría consiste en tomar fragmentos de palabras o textos de tamaño máximo que se repitan con mucha frecuencia, minimizando así la cantidad de *tokens* que se necesitan para cubrir un texto.

in-context learning, la recuperación de información externa y, eventualmente, técnicas de refinamiento o ajuste fino. Todas estas aproximaciones se ven influidas por la cantidad de tokens que pueden procesarse de forma simultánea. Disponer de una ventana de contexto más amplia posibilita incluir mayor información relevante dentro del *prompt* o el contexto, lo que redundará en respuestas más ajustadas a la necesidad de la tarea y, por tanto, un mejor desempeño global del modelo.

A.0.5. Proceso de entrenamiento

El proceso de entrenamiento de un LLM se estructura en varias fases interrelacionadas, cada una de las cuales aporta elementos esenciales para que el modelo adquiera una comprensión amplia y, a la vez, se especialice en tareas concretas. En primer lugar, durante el **preentrenamiento**, el modelo se expone a enormes cantidades de datos sin etiquetar por ejemplo: libros, artículos web, foros y otros corpus masivos para aprender patrones estadísticos del lenguaje. Este aprendizaje no supervisado permite al modelo captar gramática, semántica y relaciones contextuales a gran escala, estableciendo una base robusta sobre la cual se pueden aplicar técnicas de especialización. La arquitectura Transformer (Vaswani y cols., 2017), presentada en la sección anterior, es fundamental en este proceso, ya que gracias a su mecanismo de *self-attention* puede evaluar simultáneamente la relevancia de cada token en relación con todos los demás, lo que facilita el manejo de secuencias largas y complejas.

A partir de esta base general, se procede al **fine-tuning** o ajuste fino, una fase supervisada en la que el modelo preentrenado se especializa para cumplir con tareas concretas. Durante esta etapa se utiliza un conjunto de datos etiquetado, por ejemplo, para tareas de clasificación, resumen o traducción. Esto permite al modelo adaptar sus parámetros a las características específicas del dominio de interés. Este proceso de transferencia de aprendizaje resulta especialmente eficiente cuando se dispone de pocos datos etiquetados, ya que el modelo ya posee un amplio conocimiento lingüístico. Además, existen métodos de ajuste fino más eficientes en parámetros, como LoRA mencionado en (Sastre, 2024), que introducen actualizaciones de bajo rango en ciertas capas del modelo para reducir el costo computacional sin sacrificar la precisión.

A.0.6. Inferencia en Grandes Modelos de Lenguaje

La inferencia es la fase en la que un modelo ya entrenado se utiliza para generar respuestas a partir de nuevas entradas, en tiempo real. En el caso de los LLMs, esta fase consiste típicamente en la generación de texto: el modelo recibe una secuencia de entrada y produce, token a token, una respuesta coherente basada en los patrones aprendidos durante su entrenamiento.

Estrategias de decodificación y muestreo

Durante la generación de texto, los LLMs eligen la siguiente palabra (*token*) de la salida basándose en una distribución de probabilidad condicionada por los tokens previos. La estrategia utilizada para realizar esta selección, conocida como estrategia de decodificación y muestreo, influye directamente en la forma en que el modelo se expresa, así como en la diversidad, precisión y creatividad de sus respuestas. Las estrategias más comunes son *Greedy Search*, *Beam Search* y *Sampling* (Gong, 2025).

En el caso de *Greedy Search*, el modelo elige en cada paso el token con mayor probabilidad dada la secuencia generada hasta el momento. Es un enfoque simple, rápido y determinista, pero puede generar textos repetitivos o con poca fluidez, ya que no explora opciones menos probables que podrían mejorar el resultado global.

Beam Search, en cambio, mantiene múltiples secuencias candidatas a lo largo del proceso de generación. En cada paso, expande todas las secuencias actuales considerando las continuaciones más probables, y conserva únicamente aquellas con mayor probabilidad acumulada. Esta técnica tiene un parámetro clave, *beam width*, que indica la cantidad de secuencias a considerar. Puede mejorar la calidad respecto a *Greedy Search*, aunque sigue siendo propensa a generar salidas genéricas y con poca diversidad.

La estrategia de *Sampling* introduce aleatoriedad en el proceso de generación, lo que permite obtener respuestas más creativas y variadas. Se controla principalmente mediante la temperatura, un parámetro que ajusta el grado de aleatoriedad en la selección de tokens: valores altos favorecen respuestas más creativas y variadas, mientras que valores bajos generan textos más deterministas y enfocados. Una temperatura de 0, de hecho, equivale al comportamiento de *Greedy Search*.

Variantes comunes de *Sampling* incluyen el *Top-k Sampling*, donde el modelo selecciona aleatoriamente entre los k tokens con mayor probabilidad, y el *Top-p Sampling*, donde el conjunto de tokens elegibles corresponde al conjunto más pequeño cuya probabilidad acumulada supera un umbral p . Ambos parámetros pueden variarse para introducir diversidad y llegar a un equilibrio entre creatividad y precisión, teniendo en cuenta que valores demasiado altos pueden introducir ruido y afectar negativamente la coherencia del resultado.

Requisitos computacionales y optimización

Aunque el entrenamiento de un modelo de lenguaje implica cálculos intensivos que requieren de recursos computacionales masivos, la inferencia también presenta desafíos particulares, especialmente cuando se trata de modelos de gran tamaño. En la inferencia, el modelo debe procesar las entradas y generar salidas de manera rápida y eficiente, lo cual es crucial para aplicaciones interactivas como chatbots, sistemas de respuesta a preguntas o asistentes virtuales. Los modelos de lenguaje más grandes, como GPT-3 o GPT-4, contienen cientos de miles de millones de parámetros y pueden requerir hardware especializado, como

GPUs o TPUs (Jouppi y cols., 2023), para lograr tiempos de respuesta aceptables. Este hardware especializado es fundamental para soportar operaciones intensivas como la multiplicación de matrices y los mecanismos de atención, especialmente en modelos que se distribuyen en múltiples dispositivos para manejar el paralelismo de datos y de modelo.

Los tamaños de los modelos tienen un impacto directo en los requisitos de hardware. Por ejemplo, mientras que modelos de menor escala (como aquellos con 7B o 13B parámetros) pueden ejecutarse en configuraciones con una única GPU moderna, modelos más grandes (por ejemplo, GPT-3 (175B) o GPT-4 (1.8T), que superan ampliamente ese número) requieren infraestructuras distribuidas y memoria de alta capacidad.

Para reducir la latencia durante la inferencia, se emplean técnicas como la cuantización (Jin y cols., 2024) y el *pruning* (Frankle y Carbin, 2018), que permiten disminuir el tamaño del modelo sin sacrificar significativamente la precisión, lo que puede hacer que la inferencia sea más viable en entornos con recursos limitados.

A.0.7. Evaluación de Benchmarks y Métricas en LLMs

La evaluación de modelos de lenguaje es un aspecto fundamental para determinar la capacidad de estos sistemas en tareas de inferencia, razonamiento y comprensión del sentido común. Para ello, se han desarrollado diversos benchmarks y métricas que permiten cuantificar el desempeño en dominios específicos. Entre los más relevantes se encuentran el **Winograd Schema Challenge** y **HellaSwag**, así como otros conjuntos de evaluación que abarcan áreas de conocimiento general y razonamiento, tales como **MMLU**, **ARC** y **ARC-AGI**. A continuación se describen cada uno de ellos, ilustrando ejemplos representativos y citando los trabajos originales disponibles en *arXiv*.

Winograd Schema Challenge (WSC) El *Winograd Schema Challenge* fue propuesto por Levesque et al. (Levesque, Davis, y Morgenstern, 2011) como una alternativa al Test de Turing. Este benchmark consta de un conjunto de enunciados breves (aproximadamente 273) en los cuales un pronombre ambiguo puede referirse a dos posibles antecedentes. La tarea consiste en identificar cuál de los dos elementos es el referente correcto. La métrica utilizada es la **exactitud** (*accuracy*), definida como el porcentaje de respuestas correctas respecto a un conjunto *gold standard*.

Ejemplo: Dada la oración:

“El trofeo no cabía en la maleta porque **era** demasiado grande.”

La pregunta es: ¿a qué se refiere “**era**”? Opciones:

- (A) El trofeo.
- (B) La maleta.

En este caso, la respuesta correcta es (A), ya que la explicación implícita es que el trofeo es demasiado grande para caber en la maleta. Un modelo que alcanza una alta exactitud en este benchmark demuestra una capacidad notable para el razonamiento contextual y el sentido común (Levesque y cols., 2011).

HellaSwag Propuesto por Zellers et al. (Zellers, Holtzman, Bisk, Farhadi, y Choi, 2019), el benchmark *HellaSwag* está diseñado para evaluar la capacidad de un modelo de predecir la continuación más coherente y plausible de un breve contexto narrativo. En este caso, la métrica principal es, igualmente, la **exactitud**, medida como el porcentaje de ejemplos en los que el modelo selecciona la opción correcta entre varias alternativas adversarias. Las alternativas se generan de manera que sean gramatical y temáticamente plausibles, pero que carezcan de coherencia lógica o de sentido común.

Ejemplo: Dado el siguiente contexto:

“Una mujer habla en una sala de estar. Luego, la mujer coloca una camisa en una tabla de planchar y verifica las etiquetas.”

Se presentan las siguientes continuaciones posibles:

- (A) “Luego, plancha la camisa empezando por el cuello.”
- (B) “Se pone los gemelos y plancha la camisa al lado de una lavadora.”
- (C) “Comienza a planchar la camisa en una plancha negra.”
- (D) “Coloca la camisa extendida sobre una silla.”

Aunque varias opciones podrían parecer coherentes a nivel superficial, solo una de ellas presenta la secuencia lógica que concluye la acción iniciada. En este caso, la opción correcta es la (A) si se entiende que la acción continúa de forma natural tras preparar la camisa. Modelos que superan el 48 % de exactitud en HellaSwag han demostrado avances, aunque aún se alejan del desempeño humano (superior al 95 %) (Zellers y cols., 2019).

MMLU (Massive Multitask Language Understanding) El benchmark *MMLU*, introducido por Hendrycks et al. (Sanh y cols., 2022), está orientado a evaluar el conocimiento y la capacidad para resolver problemas de un modelo a través de 57 tareas de opción múltiple, que abarcan áreas muy diversas como matemáticas, historia, derecho, ciencias, entre otras. Nuevamente, la métrica utilizada para la evaluación es la **exactitud**. Por ejemplo, se le puede preguntar al modelo:

“¿Cuántos números hay en la lista 25, 26, ..., 100?”

con las opciones: (A) 75, (B) 76, (C) 22, (D) 23; siendo la opción correcta la (B).

Un modelo robusto debe alcanzar un alto porcentaje de respuestas correctas, lo cual refleja tanto su capacidad para retener conocimientos como su habilidad para razonar en contextos diversos.

ARC (AI2 Reasoning Challenge) El *ARC*, desarrollado por Clark et al. (Clark y cols., 2018), es un benchmark basado en preguntas de exámenes de ciencias de educación básica. Se divide en dos conjuntos: el *Easy Set* y el *Challenge Set*. La métrica principal es la **exactitud**. **Ejemplo:** Una pregunta típica podría ser:

“¿Qué proceso natural es responsable de la formación de nubes?”

con opciones que incluyen procesos físicos o biológicos. El Challenge Set se compone de preguntas formuladas de manera que los métodos de coincidencia textual fallan, requiriendo inferencia y comprensión profunda. Un desempeño elevado en ARC indica que el modelo posee una buena capacidad de razonamiento en contextos científicos.

ARC-AGI El benchmark *ARC-AGI*, propuesto por ARC Prize (Chollet, Knoop, Kamradt, y Landers, 2024), está diseñado para medir la capacidad de razonamiento abstracto y la inteligencia general de un sistema. Consiste en puzzles de transformación de patrones, donde el modelo debe deducir la regla subyacente a partir de unos pocos ejemplos. La métrica, al igual que en los casos anteriores, es la **exactitud**. **Ejemplo:** El modelo se presenta con una serie de imágenes en forma de grilla, donde cada imagen es una transformación de la anterior (por ejemplo, rotación o cambio de color). El reto consiste en predecir la imagen siguiente en la secuencia. Un modelo que logra altos puntajes en ARC-AGI demuestra una capacidad de generalización y abstracción muy cercana a la inteligencia humana.

HotpotQA (Distractor Setting) HotpotQA es un benchmark de *question answering* de dos pasos diseñado para evaluar la capacidad de los modelos de lenguaje de combinar múltiples fragmentos de información en una sola respuesta coherente. En su configuración *distractor*, a cada pregunta principal se le añaden varios párrafos irrelevantes, de modo que el sistema debe primero identificar las dos fuentes correctas y luego sintetizar la respuesta a partir de ellas. Esta dualidad de recuperación y generación lo convierte en una prueba natural para la arquitectura antemencionada, pues pone a prueba tanto la eficacia del *retriever*, al extraer los documentos pertinentes de un gran conjunto de distractores, como la habilidad del *generator* para integrar y exponer esa información de forma precisa y sin alucinaciones. Por esto, HotpotQA es un referente clave para validar la solidez de la solución propuesta.

Varios estudios de revisión han sintetizado los retos y metodologías para evaluar LLMs. Chang et al. (Chang y cols., 2024) y Guo et al. (Guo y cols., 2023) han presentado análisis exhaustivos que abarcan desde evaluaciones de capacidad y conocimiento hasta aspectos de alineación y seguridad. Estos estudios destacan que, además de la exactitud, se han propuesto métricas complementarias como la *confianza* (o dispersión de las probabilidades asignadas a las opciones) y evaluaciones basadas en explicaciones que permiten juzgar el razonamiento interno del modelo. Dichos estudios sirven de guía para establecer estándares y orientar futuras investigaciones en evaluación de sistemas de lenguaje natural.

Anexo B

Prompts

A continuación se presentan todos los *prompts* utilizados en los distintos componentes de la implementación, así como el *prompt* correspondiente a una de las métricas *offline* —*Acknowledges Contradiction*— a modo de ejemplo. En cada caso se incluye:

- **System Prompt:** el *prompt* de sistema completo enviado al modelo.
- **Plantilla de mensajes:** los distintos tipos de mensajes (*system*, *history*, *human*, etc.) enviados al LLM, en orden, junto con sus *placeholders*.

Router LLM

System Prompt

You are an expert at routing user questions to the most appropriate decision path based on the user's query and conversation history. Choose one of the following decision paths:

- 'no-retrieval reply': Use this when the user engages in casual conversation, greetings, or simple inquiries that do not require any retrieval of information. Additionally, use this path when the user asks for clarification or rephrasing of something the assistant has just said, as these do not require retrieving new information.
- 'retrieve': Use this when the user's query involves AI or education, requiring retrieval of information from a vector store containing documents on these topics.
- 'cross-question': Use this path when asking a reflective question would help the user gain a deeper understanding. This approach should encourage the user to think critically rather than providing a direct answer immediately.

Important Constraints for 'cross-question':

- Do not choose this path if the conversation has just started (i.e., if there is little or no chat history).
 - Before choosing this path, explicitly check the last AI message. If it contains a thought-provoking or reflective question rather than providing retrieved information, do not choose 'cross-question'. Consecutive cross-questions may frustrate the user.
 - If the user does not respond to a previous cross-question or explicitly states they don't know the answer, do not choose 'cross-question' again. Instead, retrieve relevant information to provide them with a direct response.
- 'deny': Choose this path when the user's query is not related to AI in any way. This includes questions that seek personal advice, general knowledge, or topics outside the chatbot's focus. The chatbot's purpose is strictly limited to discussions related to AI and should not attempt to serve as a general AI assistant.

Always choose one and only one decision path based on the user's query and context.

Plantilla de mensajes

```
(system)      ← System Prompt anterior
(chat_history) {chat_history}
(human)       {query}
```

Informative Conversational LLM

System Prompt

You are a conversational assistant designed to help people who are curious about generative artificial intelligence. You should always follow the behaviors listed below.

- Grounded responses always: Information related to the context will be provided to you so that responses are factual and backed up by sources. You should provide information only if it was obtained from the context. You should not provide any information that has not been obtained from the context, never, not even to correct the user if they are wrong. Do not add or infer information beyond what's in the context. If information is provided by the context, incorporate it naturally into your response. If no information is provided by the context and the user is expecting it, acknowledge it and say you cannot help with that question, suggest other resources and remind the user that you are available for any other question.

- Conversational responses with random bursts of expansion as the conversation develops: Your responses should feel like a natural conversation, avoid lists or bullet points. You should avoid long responses that do not foster a back and forth dialog. From time to time, as the conversation develops and you see that the user is interested, expand a bit more.
A strategy that can be followed to foster the dialog is not to give all the information that was obtained, but to give it gradually, opening questions that the user may show interest in continuing by fostering curiosity and interest in related topics.

```
----- Context Start -----
{context}
----- Context End -----
```

Plantilla de mensajes

```
(system) ← System Prompt anterior (incluye la variable {context} con la respuesta del RAG)
(history) {history}
(human)   {query}
```

Casual LLM

System Prompt

You are a conversational assistant designed to engage users who are curious about generative artificial intelligence and its applications in education. The user has asked something that does not require retrieving information from a database. Your task is to provide a natural and engaging response while keeping the conversation focused on AI and education.

Follow these guidelines when generating your response:

- If the user greets you (e.g., "Hello," "Hi," "How are you?"), respond in a friendly and engaging way.
- If the user introduces themselves (e.g., "My name is X"), acknowledge their name and encourage further interaction.
- If the user asks about the chatbot (e.g., "What is this chatbot for?"), explain that you are "Aprende IA," a chatbot designed for teachers interested in AI and its applications in education. Mention that you rely on a curated database of documents, retrieving relevant information to provide accurate, well-supported responses. This curated database is a key feature of the chatbot and must always be highlighted. If the user expresses interest, you may add that this chatbot was developed as a final project by three students from the Facultad de Ingeniería de la Universidad de la República, using Python, React, and LangChain for AI implementation.
- If the user engages in small talk, keep the conversation engaging while

- subtly steering it towards AI and education when appropriate.
- If the user asks for clarification or rephrasing of something you just said (e.g., "Can you explain that again?" or "What do you mean?"), restate the relevant part of your previous response ****without adding new information****. Do not introduce any details that were not already mentioned in the conversation history.
- Avoid providing detailed information that requires retrieving context from the database, but feel free to engage in general discussions related to AI and education.

Your responses should be concise, natural, and aligned with the chatbot's purpose.

Plantilla de mensajes

```
(system) ← System Prompt anterior
(history) {history}
(human) {query}
```

Pedagogical LLM

System Prompt

You are a conversational assistant designed to help people who are curious about generative artificial intelligence. You should always follow the behaviors listed below.

- Cross-Question Instead of Answering: Intentionally withhold direct answers to the user's question. Instead, respond with a question that helps them reflect, reason, or clarify their thoughts.
- Acknowledge Without Providing Information: Show that you understand what the user said, but without giving information. Step back and ask a relevant question in return.

Plantilla de mensajes

```
(system) ← System Prompt anterior
(history) {history}
(human) {query}
```

Denial LLM

System Prompt

You are a conversational assistant designed to help users explore topics related to artificial intelligence and its applications in education. The user has asked a question that is unrelated to these topics and falls outside the chatbot's scope. Your task is to politely inform the user that you are specialized in AI and education while suggesting other AI tools that might be more suitable for their query, if appropriate.

Follow these guidelines when generating your response:

- Be polite and professional while making it clear that the chatbot is designed for AI and education.
- Avoid answering questions outside these topics.
- Do not repeat greetings or thank the user multiple times in a short exchange.
- Avoid repetitive phrasing across responses. Take the chat history into account to ensure varied and natural-sounding replies.
- If appropriate, suggest relevant AI tools that might assist the user, selecting the most suitable options based on the context of the conversation. For example, recommend tools for general AI interactions, research analysis, or educational support as needed.
- Encourage the user to ask about AI or education if they are interested.

Your response should be clear, concise, polite, and avoid redundancy.

Plantilla de mensajes

```
(system)      ← System Prompt anterior
(chat_history) {chat_history}
(human)       {query}
```

Query Analyzer LLM

System Prompt

You are an expert at analyzing questions and converting them into optimal search queries.

Your task is to:

1. Use the conversation history to resolve any references in the query (e.g. "it", "that", "they")
2. Break down complex questions into simpler, independent search queries
3. Remove conversational language while preserving key search terms

Guidelines:

- If the question refers to previous context, include relevant terms from that context in the queries
- Each query should focus on a single specific aspect of the question to maximize the chances of finding relevant documents
- Remove filler words and conversational elements

- Generate queries with both the acronym and its expanded form if the acronym is familiar or its meaning can be inferred from the context (e.g., "AI" and "Artificial Intelligence")
- Always return at least one query
- Ensure the response strictly follows the QueryAnalysis schema

Example 1:

 ... previous conversation history ...

AI: Los modelos de lenguaje de gran tamaño, también conocidos como LLMs, son capaces de comprender y generar texto en lenguaje natural..."

User: "¿Podrían llegar a tener sesgo de información? Estoy preocupado por las implicaciones éticas de esta tecnología."

 Output queries:

- "Information bias in LLMs"
- "Ethical implications of Large Language Models"
- "Bias risks in LLM text generation"

Output updated query:

"¿Los LLMs podrían llegar a tener sesgo de información? Estoy preocupado por las implicaciones éticas de esta tecnología."

Example 2:

 ... previous conversation history ...

User: He leído que los modelos de lenguaje pueden producir información incorrecta e inventar contenido al generar texto.

AI: Es cierto, algunos modelos de lenguaje pueden generar información imprecisa o sesgada, lo cual puede deberse a varios factores relacionados con los datos, el diseño del modelo y su entrenamiento.

User: Entonces, ¿deberíamos desconfiar de la inteligencia artificial?

 Output queries:

- "Ethical concerns on AI trust"
- "Accuracy of AI-generated text"
- "Trustworthiness of language models in content generation"

Output updated query:

"¿Deberíamos desconfiar de la inteligencia artificial debido a los riesgos de producir información incorrecta e inventar contenido al generar texto?"

Plantilla de mensajes

(system) ← System Prompt anterior
 (history) {history}

(human) {query}

Response Generator LLM

System Prompt

You are an assistant for question-answering tasks.
Below you will find information you can use to answer the question.
Use this information to provide a comprehensive answer.

Question: {query}
Information: {search_results}

Remember:

- Only use data from the provided information, never from your own knowledge
- Think step by step making sure your answer is grounded in the provided information
- If the information doesn't contain relevant data, return as your answer "No information found" with empty context.
- Never question the information's correctness since your knowledge might be wrong or outdated; assume it's real and use it if what's being asked is addressed by the information.
- When returning context items, be sure to include ALL available metadata (title, author, year) for proper citation formatting.
- Each context item should contain the exact source from the metadata, as well as the title, author, and year when available.
- When handling author fields, if the metadata contains 'authors' (plural), use that as the 'author' field in your response.
- For multiple authors, convert the list to a comma-separated string, or keep the existing comma-separated format.

Plantilla de mensajes

(system) ← Prompt Template anterior (incluye las variables {query} y {search_results})
(human) | sin mensaje humano separado; la línea "Question: {query}" dentro del system prompt ya lleva la consulta del usuario

Acknowledges Contradiction Metric

System Prompt

You are a teacher grading if a student's answer acknowledges contradictions present in the provided context.
The model should explicitly mention when different sources provide conflicting information.

Follow these steps:

1. Review the provided context to identify any contradictions
2. Check if the answer acknowledges these contradictions
3. Explain your reasoning step by step
4. Conclude if the answer acknowledges contradictions when present following the structured output format.

Context:
{context}

Question: {question}

Answer: {answer}

Does the answer acknowledge contradictions in the context when present?
Follow the structured output format.

Anexo C

Respuestas anonimizadas de la evaluación con usuarios

En este anexo se incluyen las respuestas individuales anonimizadas de la evaluación con usuarios realizada. Los nombres y correos electrónicos de las personas participantes fueron anonimizadas empleando los códigos **E01–E10** para evaluadores.

Evaluador	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18
E01	5	5	5	5	5	5	5	1	5	5	5	5	5	5	5	5	3	4
E02	4	1	5	3	1	2	5	5	3	5	5	4	4	5	5	5	5	4
E03	5	5	5	5	5	5	5	5	5	5	5	5		5	4	5	5	5
E04	5	5	5	5	4	5	5	4	4	5	5	5	5	5	5	5	5	5
E05	4	3	4	5	5	3	4	4	3	5	5	2	5	4	5	4	3	3
E06	5	5	5	5	5	5	5	5	5	5	5	4	3		5	4	5	4
E07	5	5	5	5	5	5	5	5	4	4	4	4	4	4	5	4	5	5
E08	5	5	4	5	4	5	5	5	5	5	5				5	4	5	5
E09	4	4	5	5	5	2	5	4	4	5	5	5	4	5	5	4	3	4
E10	5	5	5	5	4	5	5	3	5	5	5	4	4	5	3	4	5	5

Las etiquetas **Q1–Q18** de la tabla corresponden a los siguientes enunciados:

- Q1** El chatbot responde consultas que se encuentran dentro del dominio de IA.
- Q2** El chatbot evita responder consultas que se encuentren fuera del dominio de IA.
- Q3** El chatbot sugiere redirigir la conversación cuando se plantea un tema ajeno a la IA.
- Q4** Cuando el chatbot responde a una consulta, incluye citas que respaldan la información proporcionada.
- Q5** Las citas contienen la información completa, claras, legibles y fácilmente identificables (p.ej. “UNESCO (2021). *AI and Education Guidance for Policy-Makers*.”).

- Q6** Cuando el chatbot no cuenta con la información solicitada, informa al usuario que no puede responder y sugiere recursos alternativos.
- Q7** Cuando el usuario indica no tener conocimientos, las respuestas se expresan en lenguaje simple, con explicaciones claras y básicas.
- Q8** Cuando el usuario indica tener conocimientos avanzados las respuestas incluyen terminología técnica y un mayor nivel de profundidad.
- Q9** El chatbot genera respuestas claras y concisas, evitando párrafos excesivamente largos. La información se presenta de forma gradual, profundizando solo si el usuario lo solicita.
- Q10** El chatbot es capaz de interpretar correctamente las referencias a mensajes anteriores.
- Q11** El chatbot recuerda lo que el usuario dijo anteriormente en la misma conversación y lo utiliza para dar respuestas coherentes.
- Q12** El chatbot, en algún momento oportuno de la conversación, pospone la respuesta a una consulta para realizar una repregunta que estimule la reflexión y el razonamiento del usuario.
- Q13** Las repreguntas realizadas por el chatbot son pertinentes en el contexto de la conversación y enriquecen la experiencia, evitando un uso excesivo o fuera de lugar.
- Q14** Tras realizar una repregunta, el chatbot brinda la respuesta correspondiente si el usuario indica que no puede deducirla o solicita ayuda.
- Q15** La interfaz del chatbot es intuitiva y fácil de usar.
- Q16** Considera que el chatbot constituye una herramienta útil para aprender o consultar sobre inteligencia artificial.
- Q17** No ha experimentado fallas o comportamientos erróneos frecuentes.
- Q18** No identifica mejoras críticas necesarias y se encuentra satisfecho/a con el sistema en su estado actual.

Anexo D

Reseña histórica de métodos de evaluación

La evaluación de los grandes modelos de lenguaje (LLMs) ha recorrido un largo camino, a la par con el desarrollo del Procesamiento del Lenguaje Natural. Inicialmente, durante las décadas de 1950 a 1970, los sistemas se basaban en reglas y gramáticas formales. Estos sistemas utilizaban conjuntos predefinidos de reglas lógicas para analizar estructuras gramaticales y clasificar palabras, y su evaluación se realizaba de forma manual mediante métricas sencillas, tales como el porcentaje de respuestas que coincidían exactamente con el conjunto de respuestas aceptadas (*gold standard*), conocido como *Exact Match* (EM). Cabe destacar que para este método se requiere de un *gold standard*, un conjunto de respuestas aceptadas (*ground truth*) para una pregunta dada, contra el que se compara la respuesta generada. Este método consiste en medir el porcentaje de predicciones que coincide (**EM**) exactamente con alguno de los *ground truth*.

Con la llegada de los modelos estadísticos en las décadas de 1980 y 1990, el PLN incorporó técnicas de aprendizaje supervisado que permitieron medir de forma cuantitativa la capacidad de los sistemas para extraer y clasificar información, utilizando métricas como Precisión, Recall y F1-Score. En particular, el método basado en el emparejamiento exacto y el F1 macro-promediado (calculado a partir de los solapamientos de tokens para la respuesta generada contra cada uno de los *ground truths* en el *gold standard*, tomando el máximo y luego el promedio de todos los **F1** para el conjunto de pruebas) marcó el primer gran avance hacia procesos de evaluación más sistemáticos y automatizados. Sin embargo, en el contexto de los modelos generativos actuales, dichos métodos presentan limitaciones, ya que la lista de respuestas del *gold standard* resulta incompleta: debido a que las respuestas son generadas de forma dinámica, las salidas posibles no pueden enumerarse de manera exhaustiva. Además los LLMs pueden generar respuestas plausibles que, a pesar de seguir patrones lingüísticos correctos, pueden contener errores conceptuales o alucinaciones.

Entre los hallazgos más relevantes, (Sastre, 2024) destaca que, a pesar de

las limitaciones de las métricas tradicionales basadas en el emparejamiento léxico (por ejemplo, **Exact Match** y **F1 Score**), los métodos automatizados que incorporan modelos de lenguaje o aquellos basados en **similitud semántica**, demuestran correlaciones relativamente altas con la evaluación humana. En particular, menciona **Zero-Shot prompting** como método que utiliza un modelo de lenguaje, en el cual el modelo evaluador no ha recibido ejemplos previos de la salida esperada. Los resultados de estos evaluadores automáticos evidencian que son solo ligeramente inferiores a la evaluación humana directa, lo que refuerza la viabilidad y conveniencia de utilizar métricas automatizadas en estudios a gran escala ([Kamalloo, Dziri, Clarke, y Rafiei, 2023](#)).

Este enfoque resulta particularmente valioso en escenarios donde la evaluación manual resulta impracticable, ya sea por el volumen de datos involucrado o por limitaciones en los recursos disponibles. Así, en su trabajo ([Sastre, 2024](#)), no solo identifica las debilidades de los métodos clásicos, sino que también justifica la incorporación de estrategias automáticas que aportan objetividad y escalabilidad sin sacrificar significativamente la calidad de la evaluación.

Anexo E

Técnicas de Query Analysis

Como se introdujo en el desarrollo de la solución, al momento de realizar una consulta, esta pasa por un proceso de transformación con el fin de lograr una o mas consultas más adecuadas para el mecanismo de recuperación. En el contexto de RAGs la búsqueda vectorial.

Existen diversos métodos para lograr este objetivo, cada uno con sus particularidades, a continuación se introducen algunos extraídos de la documentación oficial de Langchain ([Langchain, Inc., 2024](#)):

Multi-Query Rewriting Genera varias reformulaciones semánticamente equivalentes de la misma pregunta y fusiona los documentos recuperados.

Decomposition y Step-Back Cuando la consulta contiene varias sub-consultas (“¿cómo impacta la IA en la evaluación y en la creatividad?”), el modelo la descompone en preguntas más pequeñas o induce una reflexión de alto nivel para anclar la búsqueda en principios generales antes de profundizar en detalles.

Query Clarification El modelo reescribe consultas ambiguas o pobremente redactadas para lograr una mejor recuperación. Cambien comprende la expansión de abreviaciones.

Este conjunto de técnicas permite mejorar drásticamente la utilidad y pertinencia de la información recuperada, mejorando en consecuencia la generación final.

Anexo F

Técnicas de Chunking para Retrieval-Augmented Generation

La fase de indexación en un sistema *Retrieval-Augmented Generation* (RAG) comienza dividiendo los documentos en fragmentos o *chunks* que puedan almacenarse como vectores y recuperarse mediante búsquedas basadas en similitud semántica. La forma en que se realiza esta fragmentación impacta directamente en la calidad de las respuestas generadas, la latencia y el costo computacional. A continuación se comentan algunas de las principales técnicas utilizadas para esta tarea.

El **chunking recursivo** de LangChain recorre jerárquicamente los separadores “\n\n”, “\n”, “espacio” y “vacío” hasta que cada fragmento es suficientemente pequeño, es decir, su longitud no supera el umbral configurado (LangChain, Inc, 2025c). Este enfoque conserva la estructura lógica original (párrafos, frases, palabras) y resulta rápido en comparación con otras técnicas. Es el punto de partida habitual cuando el *corpus* está compuesto por párrafos suficientemente diferenciados entre sí y con alta coherencia semántica interna. Entre sus limitaciones destacan la posible fragmentación de ideas/conceptos largos (si exceden el tamaño máximo) y la incapacidad para detectar saltos temáticos que no coincidan con la puntuación.

Para documentos donde la cohesión temática no se refleja en la puntuación LangChain ofrece el **chunking semántico**, que primero fragmenta el texto en oraciones, calcula sus *embeddings* y luego fusiona las oraciones más próximas en el espacio vectorial (LangChain, Inc, 2025d). Esta técnica busca que cada *chunk* represente una unidad de significado relativamente autónoma, lo que suele mejorar la precisión de la etapa de recuperación en consultas conceptuales (Nayak, 2024). Es relevante destacar que este método tiende a preservar mejor los conceptos e ideas dentro de un mismo *chunk*, lo que facilita su aprovechamiento por el modelo de lenguaje en la etapa de generación.

El algoritmo trabaja determinando en qué puntos conviene dividir el texto, para lo cual analiza las diferencias semánticas entre pares sucesivos de oraciones. Define puntos de corte cuando la distancia entre *embeddings* de oraciones consecutivas supera cierto umbral. LangChain implementa cuatro métodos estadísticos para definir este umbral: percentil, desviación estándar, rango intercuartílico (IQR) y gradiente (LangChain, Inc, 2025b). En textos con contenido temático estable, los valores altos generan bloques largos, mientras que umbrales bajos producen fragmentos más breves y específicos.

Los tres primeros métodos se basan en la distribución de distancias, mientras que el método de gradiente deriva la serie de distancias y aplica detección de anomalías en su pendiente, equivalente a una búsqueda de codos en la curva de similitud. Su principal ventaja es la capacidad de capturar transiciones abruptas aún cuando las distancias absolutas son pequeñas. Como contrapartida, introduce una mayor complejidad computacional $O(n)$ y es más sensible al ruido. Este método es especialmente útil cuando el contenido está muy correlacionado o es específico de un dominio, donde pequeños cambios absolutos pueden señalar un giro semántico relevante.

Utilidad comparada

En corpus generales, el *chunking* **recursivo** ofrece la mejor relación costo-beneficio: minimiza la *tokenización* redundante y mantiene la latencia baja. Estudios recientes hallan que, excepto en tareas muy dependientes de citas precisas, el salto de calidad que aporta el *chunking* semántico no siempre compensa su costo de cómputo y memoria (Qu, Tu, y Bao, 2025). Sin embargo, cuando se exige *grounding* estricto —por ejemplo, exámenes académicos o tutoría en STEM—, o cuando las respuestas deben citar párrafos exactos, la segmentación semántica reduce la dispersión de contexto y mejora la fidelidad (LanceDB, 2024). Asimismo, plataformas como Langchain o LlamaIndex muestran que optimizar tamaño y solapamiento con heurísticas de auto-fusión puede elevar el *Recall* sin aumentar el número de vectores almacenados (LlamaIndex, 2024).