Universidad de las Ciencias Informáticas
FACULTAD DE TECNOLOGÍAS INTERACTIVAS

Desarrollo de una herramienta basada en LLM y RAG para asistir el análisis de artículos científicos

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Joaquin Enrique Rivas Sánchez

Tutores: Msc. Angel Alberto Vazquez Sánchez
Msc. Lisset Salazar Gómez

La Habana, 2025

Dedicatoria

Le dedico esta tesis a mi Cleo, la perrita mas unica que jamas existirá, Aunque ya no estés, siempre pensaré en ti.

Agradecimientos

Le doy gracias a mis padres sin los cuales no estaría vivo en primer lugar, por siempre estar ahi para mi cuando los necesite, por siempre apoyarme sin pedir nada a cambio, por impulsarme a conseguir lo que quiero, y por siempre creer en mi.

Agradezco sinceramente al profesor Ángel y a la profesora Lisset, quienes me acompañaron como tutores a lo largo de este camino. Su orientación, paciencia y disposición para ayudarme a superar cada obstáculo han sido esenciales para la culminación de este trabajo de diploma.

Gracias a todos los miembros de los tribunales por ayudarme a mejorar esta tesis.

Finalmente gracias al verdadero héroe, mi computadora que frente a todos los problemas y el estrés al que la sometí nunca me defraudo y me a acompañado siempre en mis peores momentos.

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales sobre esta, con carácter exclusivo..

Para que así conste firmamos la presente a los 18 dias del mes de junio del año 2025.

Joaquin Enrique Rivas Sánchez
Autor

MSc. Angel Alberto Vazquez Sánchez
Tutor

MSc. Lisset Salazar Gómez
Tutora

Resumen

La presente tesis de grado propone una herramienta de código abierto, basada en modelos de lenguaje de gran tamaño (LLM) y en generación aumentada por recuperación (RAG), para facilitar el análisis semiautomático de artículos científicos, adaptada al contexto tecnológico y lingüístico de Cuba. Enfrentando limitaciones como baja conectividad, restricciones geopolíticas y falta de recursos, la solución utiliza Python, vLLM y Gradio, con modelos como `DeepSeek-R1-Distill-Qwen-1.5B` y `BAAI/bge-m3`. Desarrollada bajo la metodología *Extreme Programming* (XP), la herramienta incluye módulos de reformulación de consultas, recuperación y generación de respuestas. Evaluaciones automatizadas utilizando las métricas definidas por RAGAS revelaron resultados alentadores, validando la viabilidad técnica del sistema y sentando las bases para futuras mejoras.

Palabras clave: Aprendizaje Profundo, Contexto Cubano, Código Abierto, Generación Aumentada por Recuperación, Procesamiento de Lenguaje Natural

Índice general

Índice de figuras

Índice de tablas

Introducción

En Cuba, la investigación científica enfrenta desafíos únicos derivados de las limitaciones tecnológicas y el acceso restringido a recursos digitales. Los investigadores trabajan frecuentemente con grandes volúmenes de artículos científicos, los cuales deben analizarse manualmente debido a la falta de herramientas automatizadas adaptadas al contexto local. Este proceso consume un tiempo valioso que podría destinarse a la generación de nuevo conocimiento, especialmente en áreas críticas como la medicina, la agricultura o la energía, sectores prioritarios para el desarrollo nacional. Además, la conectividad a internet es intermitente y costosa, lo que dificulta el uso de soluciones basadas en la nube o modelos de inteligencia artificial (IA) de alto consumo computacional, y a los que en muchas ocasiones se encuentran bloqueadas para nuestro país.

Las herramientas existentes para el análisis automatizado de textos, como los sistemas comerciales (p. ej., ChatGPT, Elicit, etc.), presentan barreras significativas en este contexto. Por un lado, su dependencia de conexión estable a internet y hardware avanzado los hace inaccesibles para muchos investigadores cubanos, ademas están optimizados para el inglés y no priorizan el español, lo que limita su precisión en documentos científicos locales.

A partir de la problemática anteriormente descrita se plantea como **problema de investigación**: ¿Cómo desarrollar una herramienta de inteligencia artificial accesible, eficiente y localizable que permita a investigadores cubanos analizar documentos científicos de forma automatizada, aprovechando los Modelos de Lenguaje de Gran Tamaño y Generación Aumentada por Recuperación, sin depender de infraestructura costosa o conexión a internet? Se tiene como **objeto de estudio**: Modelo de Lenguaje de Gran Tamaño. Como **campo de acción**: Extensión de Modelo de Lenguaje de Gran Tamaño basado en Generación-Aumentada por Recuperación.

Se propone como **objetivo general**: Desarrollar una herramienta de código abierto, basada en LLM y RAG, para el análisis semiautomático de artículos científicos en formato `pdf`, adaptada al contexto tecnológico y lingüístico de Cuba.

Para dar cumplimiento al objetivo propuesto se proponen un conjunto de **tareas de investigación**:

1. Elaboración del marco teórico referente a modelos del lenguaje de gran tamaño y la extension utilizando generación aumentada por recuperación.
2. Análisis de sistemas homólogos para la identificación de los requisitos de software.
3. Implementación del sistema a partir de los requisitos de software.
4. Validación de la propuesta de solución.

Métodos de investigación

Para alcanzar el conocimiento necesario que permita cumplir con el objetivo establecido, se realiza una investigación basada en diversos métodos científicos, tanto teóricos como empíricos. A continuación, se detallan los métodos utilizados en este estudio.

Métodos teóricos:

- **Modelación**: facilitó la creación de diagramas, representado de manera gráfica parte del contenido de la investigación.
- **Analítico-sintético**: aplicado en la revisión de documentos y bibliografía, permitió identificar y extraer las ideas clave necesarias para sustentar teóricamente tanto la investigación como la propuesta presentada.

Estructura de la investigación por capítulos

El presente trabajo está dividido en los siguientes tres capítulos que recogen todo lo abordado en la investigación.

Capítulo 1 - Fundamentación Teórica: Se centra en la fundamentación teórica y conceptos fundamentales relacionados con el objeto de estudio y campo de acción. Se realiza un análisis de las soluciones similares y se selecciona la metodología de desarrollo de software y las herramientas y tecnologías propuestas para la propuesta de solución.

Capítulo 2 - Propuesta de solución: Se realiza la fundamentación de la propuesta de solución, los requisitos funcionales y no funcionales, se realiza el diseño ingenieril donde se describen las buenas prácticas así como los patrones de diseño.

Capítulo 3 - Implementación y realización de pruebas: En este capítulo se abordan los procesos de implementación de la solución propuesta, detallando los estándares utilizados. Además, se describen las pruebas realizadas para evaluar su funcionamiento, fiabilidad y desempeño, garantizando que cumpla con los objetivos planteados en la investigación.

Finalmente son presentadas las Conclusiones, Recomendaciones, Referencias Bibliográficas.

Fundamentación Teórica

En el presente capítulo se describen los conceptos relevantes que conforman el marco teórico relacionado con la investigación. También se detallan las herramientas, metodología y tecnologías que serán utilizadas para el proceso de desarrollo del software.

1.1. Modelos de Lenguaje de Gran Tamaño (LLM)

Un **modelo de aprendizaje automático** es un sistema computacional diseñado para aprender patrones y relaciones a partir de datos sin una programación explícita para cada tarea. Utiliza algoritmos estadísticos y técnicas de optimización para ajustar sus parámetros, permitiendo realizar predicciones o tomar decisiones basadas en la experiencia adquirida. Entre las categorías más comunes se encuentran el aprendizaje supervisado, el no supervisado y el aprendizaje por refuerzo. En el aprendizaje supervisado, el modelo es entrenado con datos etiquetados, es decir, cada entrada se encuentra asociada a una salida esperada. Este enfoque es ampliamente utilizado en tareas como la clasificación de texto, el reconocimiento de imágenes y la predicción de series temporales, ya que permite a los modelos aprender patrones a partir de ejemplos concretos. Por otro lado, en el aprendizaje no supervisado, el modelo trabaja con datos no etiquetados y debe identificar patrones o estructuras ocultas en los datos por sí mismo. Se emplea en problemas como la agrupación (clustering), la reducción de dimensionalidad y la detección de anomalías. Finalmente, el aprendizaje por refuerzo se basa en un sistema de recompensas en el que un agente interactúa con un entorno y aprende a tomar decisiones óptimas mediante prueba y error. Este enfoque ha demostrado ser particularmente efectivo en aplicaciones como el control de robots, la optimización de procesos y los juegos, donde el modelo mejora progresivamente su desempeño en función de la retroalimentación recibida (Goodfellow et al., 2016).

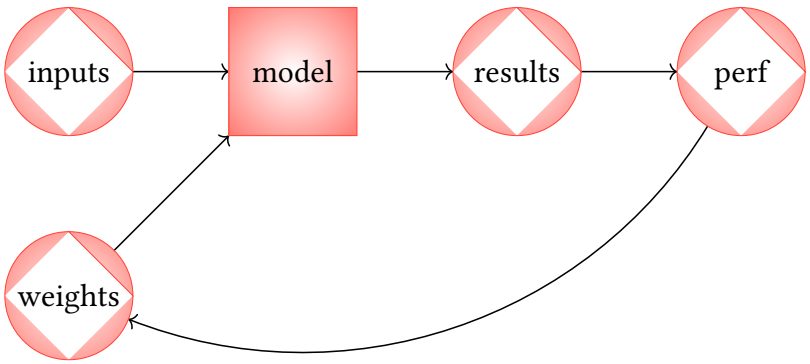


Figura 1: Esquema general del proceso de aprendizaje profundo (Gugger & Howard, 2020)

En el contexto del aprendizaje automático, Figura 1 ilustra de manera esquemática el flujo de trabajo de un modelo de aprendizaje profundo, se observa que el proceso comienza con la entrada de datos (inputs), que entrena al modelo (model). El modelo genera predicciones o resultados (results), los cuales son evaluados mediante una métrica de desempeño (perf). Adicionalmente, el modelo ajusta sus parámetros a través de un mecanismo de pesos (weights), permitiendo la optimización y mejora progresiva del rendimiento (Gugger & Howard, 2020).

Un **modelo de lenguaje** es una representación estadística basada en redes neuronales que asigna probabilidades a secuencias de palabras, permitiendo predecir la siguiente palabra en un texto. Inicialmente, se utilizaron modelos de n-gramas, pero el advenimiento de las técnicas de aprendizaje profundo ha permitido construir modelos mucho más sofisticados que comprenden contextos largos y capturan matices semánticos y sintácticos complejos. Los modelos del lenguaje son la base para tareas de generación y comprensión de texto, siendo esenciales en aplicaciones como sistemas de traducción automática, resumen de textos y respuestas a preguntas (Goodfellow et al., 2016).

El desarrollo reciente en el campo de los modelos del lenguaje se debe en gran medida a la introducción de la arquitectura Transformer en 2017, presentada en el artículo «Attention is All You Need».

Esta arquitectura revolucionó el procesamiento del lenguaje natural al:

- Eliminar la dependencia secuencial: A diferencia de las redes neuronales recurrentes (RNN) y las LSTM, los Transformers procesan todas las palabras en paralelo, lo que mejora la eficiencia en el entrenamiento.
- Introducir el mecanismo de self-attention: Este mecanismo permite al modelo ponderar la importancia de cada palabra en una oración, capturando relaciones de largo alcance de manera efectiva.
- Escalar a grandes volúmenes de datos: Debido a su arquitectura, se han desarrollado modelos de gran tamaño (como GPT, BERT y sus variantes) que utilizan billones de parámetros, lo que ha impulsado su capacidad para comprender y generar lenguaje de manera sorprendente (Vaswani et al., 2023).

La arquitectura Transformer ha permitido que los modelos de lenguaje han alcanzado un nivel sin precedentes de comprensión y generación de texto, superando enfoques tradicionales en términos de contextualización y manejo de dependencias a largo plazo (Vaswani et al., 2023). Esta evolución ha permitido aplicar estos modelos a una amplia variedad de tareas dentro del procesamiento del lenguaje natural, desde la generación automática de texto hasta sistemas avanzados de preguntas y respuestas, entre otras aplicaciones. A continuación, se detallan algunas de las principales tareas que estos modelos han revolucionado y los retos que aún persisten en su implementación.

Los sistemas de preguntas y respuestas han sido profundamente transformados por los LLM, ya que permiten interpretar preguntas en lenguaje natural y proporcionar respuestas precisas, facilitando la interacción entre humanos y máquinas en asistentes virtuales y sistemas de búsqueda avanzada (Gómez-Rodríguez, 2025). Del mismo modo, en la generación de texto y resumen automático, estos modelos han optimizado la creación de resúmenes concisos y coherentes a partir de grandes volúmenes de información, lo que es especialmente útil en entornos científicos (Han et al., 2025). Por otra parte, en la traducción automática, modelos como GPT-4 han logrado capturar la semántica y gramática de múltiples idiomas, mejorando la calidad de las traducciones (OpenAI et al., 2024).

Sin embargo, el uso de estos modelos en estas tareas no está exento de desafíos. Entre los principales retos se encuentran:

- **Alucinaciones y generación de información errónea**: A pesar de sus capacidades avanzadas, estos modelos pueden generar respuestas incorrectas o inconsistentes, lo que plantea riesgos en aplicaciones críticas.

- **Falta de interpretabilidad**: La naturaleza de los modelos basados en Transformers dificulta comprender cómo llegan a ciertas respuestas, lo que es un desafío en ámbitos como el derecho o la medicina, donde la transparencia es crucial.
- **Sesgos en los datos de entrenamiento**: Los LLM pueden heredar y amplificar sesgos presentes en sus datos de entrenamiento, lo que puede afectar la equidad y la objetividad de sus respuestas.
- **Limitaciones en el acceso a información actualizada**: Aunque los modelos pueden ser preentrenados con grandes volúmenes de datos, carecen de mecanismos nativos para acceder a información en tiempo real sin estrategias complementarias.

1.2. Memoria paramétrica y no paramétrica

La **memoria paramétrica** se refiere al conocimiento almacenado directamente en los parámetros de un modelo de inteligencia artificial, como las redes neuronales profundas. Este tipo de memoria permite a los modelos generalizar información a partir de los datos con los que fueron entrenados, pero presenta desafíos como la dificultad para actualizar información sin un nuevo entrenamiento y la incapacidad de recordar datos específicos de manera precisa. Además, estos modelos pueden olvidar o distorsionar información si no han sido entrenados con suficientes ejemplos representativos.

Por otro lado, la **memoria no paramétrica** almacena información fuera de los parámetros del modelo, permitiendo acceder a datos específicos cuando es necesario. Esto resuelve algunos problemas de la memoria paramétrica, como la dificultad de actualizar información, ya que los datos pueden modificarse sin necesidad de reentrenar el modelo. También permite acceder a conocimientos más precisos y actualizados, mejorando la interpretabilidad y adaptabilidad del sistema a nuevas situaciones sin comprometer su capacidad de generalización (Lewis et al., 2021).

1.3. Técnicas para extender la memoria de un modelo

Dado que los modelos de lenguaje de gran tamaño (LLM) tienen una capacidad limitada para almacenar y actualizar información de manera eficiente, han surgido diversas técnicas para extender su memoria y mejorar su capacidad de adaptación. Estas técnicas permiten que los modelos accedan a información externa, se actualicen sin necesidad de un nuevo entrenamiento y retengan conocimientos previos sin sufrir degradación (Lewis et al., 2021).

1.3.1. Transferencia del Aprendizaje

La transferencia del aprendizaje es una técnica clave para extender la memoria de un modelo, permitiendo que un modelo preentrenado en una tarea general se adapte a una tarea específica con una cantidad reducida de datos adicionales. En lugar de entrenar un modelo desde cero, se reutilizan los pesos y representaciones aprendidas en un dominio más amplio para mejorar el desempeño en un nuevo contexto.

Este enfoque ha demostrado ser particularmente útil en modelos de lenguaje, donde un modelo preentrenado en grandes corpus de texto puede especializarse en tareas más específicas como análisis de sentimientos, traducción de idiomas o recuperación de información. Sin embargo, uno de los desafíos principales es evitar el *catastrophic forgetting*, donde el modelo puede perder información valiosa del preentrenamiento al ajustarse a nuevos datos (Ven et al., 2024).

Una de las principales técnicas para extender la memoria de un modelo es el **fine-tuning**, que consiste en ajustar un modelo de lenguaje preentrenado con un conjunto de datos específico

para adaptar su conocimiento a una nueva tarea. En este proceso, se reutilizan los parámetros aprendidos durante el entrenamiento previo y se afinan en función de nuevos ejemplos, permitiendo que el modelo retenga su conocimiento general y lo especialice en un contexto particular. A pesar de su efectividad, el fine-tuning implica un costo computacional elevado y la necesidad de datos etiquetados relevantes, lo que puede limitar su aplicabilidad en entornos con información en constante cambio (Gao et al., 2024).

1.3.2. Generación Aumentada por Recuperación (RAG)

La RAG es un enfoque híbrido que combina la capacidad de generación de los modelos de lenguaje con técnicas de recuperación de información externa. Con RAG, el modelo no se limita únicamente a su conocimiento almacenado en sus parámetros (memoria paramétrica), sino que además consulta fuentes externas (memoria no paramétrica) para complementar y actualizar la información que utiliza en la generación de respuestas.

Este proceso se lleva a cabo en dos fases principales:

1. **Recuperación de Información**: Se emplean técnicas de búsqueda para identificar y extraer fragmentos o documentos relevantes de una base de datos o repositorio. Esto permite acceder a datos específicos y actualizados, superando la limitación inherente de los modelos preentrenados que no pueden incorporar nuevos conocimientos sin reentrenamiento.
2. **Generación de Texto**: La información recuperada se integra en el proceso de generación del modelo, enriqueciendo el contexto y permitiendo la elaboración de respuestas más precisas y especializadas. Esta fusión mejora la capacidad del modelo para manejar temas complejos o de nicho, lo que resulta especialmente valioso en el análisis semiautomático de artículos científicos (Lewis et al., 2021).

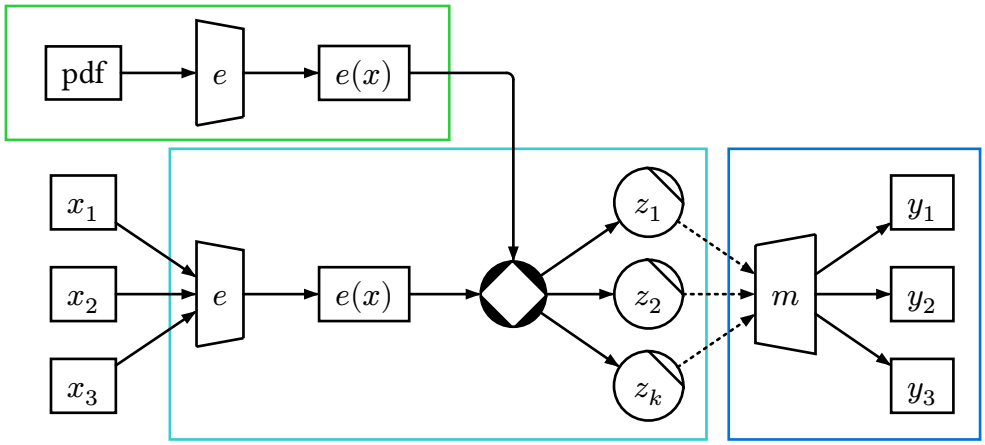


Figura 2: Descripción general de RAG (Elaboración propia)

La Figura 2 ilustra el flujo general de un sistema de Generación Aumentada por Recuperación (RAG) dividido en tres etapas, cada una señalada con un color distinto:

- **Indexado** (recuadro verde): Se muestran los documentos en PDF que, tras un proceso de extracción de información, se convierten en representaciones vectoriales (o *word embeddings*, denotadas como $e(x)$). Este paso permite crear una base de conocimiento indexada a partir de los textos originales.
- **Recuperación** (recuadro celeste): Aquí se representan las consultas del usuario (x_1, x_2, x_3), que también son transformadas en sus correspondientes embeddings $e(x)$. El círculo negro indica el cálculo de la similitud coseno (*cosine similarity*) entre los embeddings de la consulta

y los de la base de conocimiento, con el fin de recuperar los top_k documentos más relevantes para cada pregunta.

- **Generación** (recuadro azul): Finalmente, el modelo utiliza los documentos recuperados ($z_1, z_2, ..., z_k$) y las consultas para producir las respuestas (y_1, y_2, y_3). Este paso corresponde a la fase de generación, donde se integran los resultados de la recuperación para brindar información coherente y contextualizada al usuario.

Word Embedding: Es una representación vectorial de palabras en un espacio numérico continuo. Cada palabra se asigna a un vector de números, de manera que palabras con significados o contextos similares tienen vectores cercanos entre sí. Estos vectores se obtienen mediante técnicas de aprendizaje automático, lo que permite que los modelos capten relaciones semánticas y sintácticas en los textos (Mikolov et al., 2013).

Cosine Similarity: Es una medida de similitud entre dos vectores que calcula el coseno del ángulo entre ellos. En el contexto de los word embeddings, se utiliza para determinar cuán similares son dos palabras o documentos. Un valor cercano a 1 indica que los vectores son muy similares (es decir, las palabras tienen contextos o significados parecidos), mientras que valores cercanos a 0 o negativos indican poca o ninguna similitud (Mikolov et al., 2013).

$$\text{cosine similarity} = S_c(A, B) := \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Reranking: Es un paso de post-procesamiento en sistemas de recuperación por etapas que reordena el listado inicial de candidatos para elevar los más relevantes al tope. Tras una primera recuperación (ej., BM25 y/o búsqueda semántica), un modelo especializado (como un *cross-encoder*) evalúa de nuevo cada par consulta-documento, ajustando el ranking según criterios más finos de relevancia. En pipelines RAG, el reranking maximiza el recall efectivo al reducir el ruido ingresado al LLM, mejorando la calidad de las respuestas generadas. Una estrategia habitual consiste en aplicar un segundo modelo entrenado con datos anotados para discriminar con mayor precisión los fragmentos más útiles antes de la generación de texto (Pinecone, 2025).

1.3.3. Comparativa

La Tabla 1 compara los dos enfoques anteriormente mencionados: Fine-Tuning y RAG. Se analizan y contrastan estos métodos según diferentes características clave, incluyendo la adaptación a nuevos conocimientos, los requerimientos computacionales, la latencia, sus aplicaciones ideales y sus limitaciones.

Tabla 1: Comparativa entre Fine-Tuning y RAG (Gao et al., 2024)

Características	Fine-Tuning	RAG
Adaptación	Ajusta un modelo preentrenado sin necesidad de reentrenar completamente para cada nueva actualización de conocimiento	No requiere reentrenamiento o reajuste; permite agregar conocimiento externo en tiempo real
Requerimientos Computacionales	Normalmente requiere altos recursos computacionales para entrenar	Menores en comparación, ya que se enfoca en la adaptación, integración y recuperación de la información
Latencia	Baja, ya que el modelo responde directamente con el conocimiento integrado	Mayor, debido al proceso de recuperación y generación de respuestas

Características	Fine-Tuning	RAG
Aplicaciones	Ideal para tareas que requieren replicar estructuras o estilos específicos	Perfecto para tareas de recuperación de información y generación basada en fuentes externas
Limitaciones	No adecuado para incorporar rápidamente nuevos conocimientos, posibles preocupaciones éticas sobre la recuperación de datos	Mayor complejidad técnica y, debido a tener más partes interdependientes, se incrementa el riesgo de un único punto de falla

A partir de la Tabla 1, se pueden extraer varias conclusiones clave sobre las ventajas y desventajas de cada enfoque. El Fine-Tuning es una técnica poderosa cuando se busca un rendimiento altamente especializado en tareas específicas, especialmente aquellas que requieren una estructura o estilo constante en las respuestas. Sin embargo, su principal desventaja radica en los altos costos computacionales y en la dificultad de incorporar nuevo conocimiento de manera ágil, lo cual lo hace menos flexible ante entornos dinámicos donde la información cambia frecuentemente.

Por otro lado, el enfoque RAG (Retrieval-Augmented Generation) demuestra ser más adecuado para escenarios donde se necesita acceder a información actualizada o contextual, ya que permite integrar fuentes externas en tiempo real sin requerir reentrenamiento del modelo base. Aunque presenta una mayor latencia debido al proceso de recuperación de información y conlleva mayor complejidad técnica, su flexibilidad y menor costo computacional lo hacen más escalable y adaptable a cambios rápidos.

Considerando las necesidades del proyecto —especialmente la incorporación dinámica de conocimiento y la eficiencia en recursos—, la arquitectura RAG se presenta como la opción más conveniente, al equilibrar rendimiento con adaptabilidad y escalabilidad.

1.4. Estudio de sistemas a nivel internacional

A partir del estudio de los sistemas internacionales detallados en la Tabla 2 (Elicit (*Elicit: the AI Research Assistant*, 2025), ChatGPT (*ChatGPT*, 2025), ChatPDF (*ChatPDF AI | Chat with any PDF | Free*, 2025), Humata (*Humata*, 2025) y Scholarcy (*Scholarcy*, 2025)), se identificaron limitaciones que afectan su usabilidad en el contexto nacional. Estos servicios, desarrollados bajo modelos de software propietario y dependientes de servidores remotos, enfrentan barreras geopolíticas (como bloqueos en Cuba) y restricciones de uso (límites mensuales de artículos, páginas o mensajes), lo que limita su escalabilidad y adaptabilidad a entornos con necesidades intensivas, como la investigación académica o la implementación en regiones con censura. Estas restricciones contradicen los principios de accesibilidad universal y soberanía tecnológica, fundamentales para proyectos que buscan autonomía en el manejo de datos y herramientas.

Tabla 2: Comparativa entre homólogos

Servicio	Bloqueada	Uso Limitado	Localización	Código Abierto	Privacidad
Elicit	No	20 Artículos / mes	Inglés mayormente	No	Servidor remoto

Servicio	Bloqueada	Uso Limitado	Localización	Código Abierto	Privacidad
ChatGPT	Si	No permite registrarse desde Cuba	Todos	No	Servidor remoto
ChatPDF	Si	2 pdf y 20 mensajes/dia	Todos	No	Servidor remoto
Humata	No	60 páginas/mes	Inglés mayormente	No	Servidor remoto
Scholarcy	Si	10 resúmenes	Inglés mayormente	No	Servidor remoto

Si bien las soluciones analizadas no resuelven integralmente todos los desafíos identificados, ofrecen referencias valiosas para el diseño de un sistema alternativo. Por ejemplo:

- Estandarización de funcionalidades: La capacidad de ChatGPT para operar en múltiples idiomas (aunque su accesibilidad sea limitada) destaca la importancia del soporte multilingüe en plataformas globales (*ChatGPT*, 2025).
- Gestión de recursos: Las restricciones de uso, como las 60 páginas/mes de Humata, aunque insuficientes, establecen un marco base para implementar modelos de suscripción escalables.
- Seguridad básica: La dependencia de servidores remotos, si bien plantea riesgos, subraya la necesidad de protocolos de privacidad robustos en futuros desarrollos.
- Gestión de referencias bibliográficas: ChatPDF se distingue por su eficacia en el manejo de referencias bibliográficas, ya que permite identificar y vincular citas con el texto original, facilitando su visualización y verificación (*ChatPDF AI | Chat with any PDF | Free*, 2025).

1.5. Metodología de desarrollo de software

Una metodología es un conjunto organizado de procedimientos, técnicas, herramientas y documentos auxiliares que guían a los desarrolladores en la implementación de sistemas de información. Se estructura en fases y subfases que facilitan la elección de las técnicas más adecuadas en cada etapa del proyecto, permitiendo una planificación, gestión, control y evaluación efectiva, y asegurando la calidad y coherencia del producto final (Sommerville & Galipienso, 2005).

Dado que el proyecto es pequeño, el tamaño del equipo consta de un solo integrante, el tiempo es limitado, la metodología *Extreme Programming* (XP) es ideal porque permite recibir retroalimentación frecuente y realizar ajustes rápidos en el desarrollo. Su enfoque en ciclos iterativos cortos y pruebas constantes ayuda a garantizar que cada avance sea funcional y alineado con los objetivos del proyecto, optimizando el uso del tiempo y asegurando mejoras continuas sin desviaciones significativas.

Metodología de desarrollo de software Programación Extrema (XP)

La metodología XP se fundamenta en iteraciones frecuentes y breves ciclos de lanzamiento, donde los requisitos se definen mediante “historias de usuario” que se transforman en tareas

de ingeniería. Entre sus prácticas clave se encuentran la programación en pareja, el desarrollo guiado por pruebas (TDD) y la integración continua, lo que garantiza la calidad y la adaptabilidad del software a lo largo del proceso. Además, XP incorpora valores fundamentales y técnicas como el diseño simple, pruebas unitarias, soluciones *spike*, prototipos y refactorización constante, lo que la hace especialmente efectiva para desarrollar sistemas complejos con equipos pequeños (Sommerville & Galipienso, 2005). Esta se desarrolla a través de varias fases fundamentales iterativas y flexibles: Planificación, Diseño, Codificación y Pruebas.

En la **fase de planificación**, se recopilan y definen las historias de usuario que expresan de forma práctica los requisitos y objetivos del sistema. Aquí, el equipo estima el esfuerzo y los recursos necesarios para cada historia, lo que permite establecer prioridades basadas en el valor que aportan al proyecto y en las restricciones de tiempo. Este enfoque iterativo posibilita la adaptación de la planificación a medida que se van clarificando los requerimientos y surgen nuevos desafíos.

En la **fase de diseño**, se busca mantener la solución lo más sencilla posible, evitando complejidades innecesarias. El diseño es emergente, es decir, se va refinando conforme se avanza en el desarrollo, permitiendo incorporar cambios y mejoras de forma gradual. La práctica de la refactorización es fundamental en esta etapa, ya que implica reorganizar y mejorar el código sin modificar su comportamiento externo, lo que ayuda a mantener una estructura clara y eficiente.

La **fase de codificación** se centra en la implementación efectiva de las historias de usuario. Aquí, la programación en parejas fomenta la colaboración y la revisión constante del código, reduciendo errores y mejorando la calidad del software. Además, se integra el Desarrollo Basado en Pruebas (TDD), donde se escriben primero las pruebas unitarias que definen el comportamiento esperado, y luego se desarrolla el código que cumpla con estas pruebas, asegurando así que cada parte del sistema funcione correctamente desde el inicio.

Por último, en la **fase de pruebas**, se verifica de manera continua la funcionalidad y calidad del software. Se realizan pruebas unitarias para cada módulo y pruebas de aceptación basadas en las historias de usuario, lo que permite detectar y corregir errores de forma temprana. La integración continua facilita que los cambios se incorporen de inmediato al proyecto, garantizando que el sistema evolucione de manera estable y acorde a los requisitos establecidos.

1.6. Tecnologías a utilizar

A continuación se detallan las tecnologías y herramientas utilizadas así como la Metodología del desarrollo.

1.6.1. Lenguaje de Programación

Python v3.12: es un lenguaje de programación multiparadigma que prioriza la legibilidad del código y la productividad. Aunque soporta programación orientada a objetos (OOP), también integra enfoques funcionales y procedurales, lo que lo hace versátil para abordar problemas complejos.

Ventajas de Python:

- Sencillez: Su diseño enfocado en la legibilidad reduce la curva de aprendizaje y facilita la colaboración en equipos.

- Ecosistema robusto y especializado: Cuenta con frameworks maduros para la IA como Pandas, TensorFlow, PyTorch, NumPy.
- Gestión automática de memoria: Utiliza un recolector de basura para liberar recursos no utilizados, simplificando la administración de memoria.
- Comunidad activa y código abierto: Miles de librerías de terceros (PyPI) y documentación exhaustiva permiten resolver desafíos técnicos con rapidez, reduciendo costos de desarrollo y mantenimiento.
- Integración con lenguajes compilados: Mediante módulos como Cython o interoperabilidad con C/C++, optimiza segmentos críticos de rendimiento sin sacrificar la productividad (Castro et al., 2023).

Desventajas de Python:

- Rendimiento en tiempo real limitado: Al ser interpretado y tener un manejo de memoria automático, su ejecución es más lenta que lenguajes compilados como C/C++ o Rust. Esto puede ser un cuello de botella en aplicaciones de alta concurrencia o procesamiento masivo.
- Global Interpreter Lock (GIL): Restringe la ejecución paralela de hilos, limitando el aprovechamiento de multiples núcleos de la CPU.
- Tipado dinámico: Aunque en ciertos casos puede favorecer la flexibilidad, puede generar errores en tiempo de ejecución difíciles de depurar.. (Castro et al., 2023).

1.6.2. Modelos LLM

DeepSeek-R1-Distill-Qwen-1.5B-Q8_0: es una versión cuantizada[1] y destilada[2] del modelo de lenguaje DeepSeek-R1, diseñada para ser más eficiente y compacta mientras mantiene un rendimiento sólido, especialmente en tareas de razonamiento.

Ventajas

- **Rendimiento en razonamiento**: Hereda capacidades de razonamiento del modelo más grande DeepSeek-R1, mostrando un rendimiento impresionante en tareas de razonamiento lógico.
- **Versatilidad**: Puede ser utilizado en una variedad de aplicaciones, desde generación de texto hasta tareas de comprensión del lenguaje.
- **Licencia permisiva**: Está disponible bajo la licencia Apache 2.0, que permite su uso comercial sin restricciones.
- **Flexibilidad de implementación**: Disponible en varios formatos y niveles de cuantización, lo que permite su uso en diferentes tipos de dispositivos y aplicaciones (DeepSeek-AI, 2025).

BAAI/bge-m3: es un modelo avanzado de embeddings. Destaca por su versatilidad en tres dimensiones clave.

[1]Técnica de compresión que reduce la precisión numérica de los pesos del modelo. Esto disminuye significativamente el tamaño del modelo y sus requisitos computacionales, lo que lo hace más eficiente en términos de memoria y energía

[2]Técnica que permite transferir el conocimiento de un modelo grande y complejo, denominado «modelo profesor», a un modelo más pequeño y eficiente, conocido como «modelo alumno». Este proceso busca reducir el tamaño y la complejidad del modelo original sin comprometer significativamente su rendimiento

- Multifuncionalidad: Combina tres métodos de recuperación en un solo modelo:
  1. Embeddings Densos: Vectores compactos para búsqueda semántica.
  2. Recuperación léxica (sparse): Ponderación de tokens al estilo BM25.
  3. Interacción multi-vector (Col-BERT): Múltiples vectores por texto para mayor precisión.
- Multilingüístico: Soporta más de 100 idiomas con un rendimiento optimizado para lenguas como inglés, español, chino, francés y árabe.
- Multigranularidad: Procesa textos desde frases cortas hasta documentos largos (hasta 8,192 tokens) esto lo hace ideal para RAG (Retrieval-Augmented Generation) con textos extensos (Chen et al., 2024).

1.6.3. Bibliotecas

vLLM v0.8.5: es una biblioteca de código abierto diseñada para ofrecer inferencia de modelos de lenguaje grandes (LLM) de manera eficiente, especialmente en entornos con GPU, optimizando el uso de memoria y recursos de hardware. Implementada principalmente en Python y C++, vLLM se ha convertido en un motor de inferencia de alto rendimiento ampliamente adoptado en la industria (Kwon et al., 2023).

Ventajas clave:

- Procesamiento por lotes continuo: Permite procesar múltiples solicitudes simultáneamente, reutilizando cálculos comunes mediante una caché de KV (key-value), lo que mejora significativamente la eficiencia y reduce la latencia.
- Gestión eficiente de memoria GPU: Gracias a mecanismos como PagedAttention y técnicas como PyTorch Compile/CUDA Graph, vLLM maximiza el uso de la memoria disponible, permitiendo servir más usuarios y contextos largos con menos hardware.
- Compatibilidad y flexibilidad: Soporta múltiples arquitecturas y modelos populares de código abierto (como Llama, Mistral, Mixtral, Qwen), integra APIs compatibles con OpenAI y facilita el despliegue tanto en la nube como en servidores propios.
- Escalabilidad y despliegue: Admite inferencia distribuida, paralelismo de tensores, despliegue con Docker, NVIDIA Triton, Ray y SkyPilot, y se adapta fácilmente a diferentes tamaños de modelos y cargas de trabajo.
- Innovación y comunidad activa: Proyecto de código abierto con una comunidad activa que impulsa mejoras continuas, transparencia y personalización (Kwon et al., 2023).

Gradio v5.20.1: es una biblioteca de Python de código abierto diseñada para crear interfaces web interactivas para aplicaciones de IA. Permite a los desarrolladores construir y desplegar interfaces gráficas de manera rápida y sencilla, sin necesidad de conocimientos avanzados en desarrollo web

Ventajas clave:

- Fácil de usar: Con solo unas pocas líneas de código, puedes crear una interfaz web funcional para probar o demostrar modelos de IA.
- Soporte para múltiples componentes: Incluye una amplia variedad de elementos interactivos, como cuadros de texto, botones, deslizadores, subida de archivos, imágenes, audio y más.

- Interactividad en tiempo real: Soporta streaming y actualizaciones dinámicas, ideal para aplicaciones como chatbot o generadores de texto.
- Personalización: Permite modificar el diseño, añadir temas personalizados y combinar múltiples componentes para crear interfaces complejas (*Gradio*, 2025).

1.7. Conclusiones parciales

Se dieron cumplimiento a los primeros objetivos específicos, por lo que se llegó a las siguientes conclusiones:

- El análisis de los conceptos asociados estableció una base teórica sólida que facilitó la comprensión profunda de la problemática y los mecanismos subyacentes, permitiendo identificar las herramientas y técnicas más adecuadas para el proyecto.
- La elección de tecnologías como Python y vLLM definió las bases para un diseño optimizado y escalable, especialmente en entornos con recursos limitados, lo que garantiza una mayor eficiencia en el procesamiento de los modelos.
- Las características particulares del proyecto evidenciaron la necesidad de adoptar la metodología XP, que favoreció una gestión iterativa y flexible. Esta metodología permitió realizar ajustes continuos durante el desarrollo, asegurando la calidad del sistema y adaptándose rápidamente a los cambios emergentes.

Propuesta de solución

Este capítulo se enfoca en la formulación y desarrollo de la solución propuesta, derivada de un riguroso proceso de investigación. Bajo el marco metodológico adoptado, se identifican y establecen los requisitos funcionales y no funcionales que definirán las características esenciales del sistema. Paralelamente, se examina la arquitectura de base que sustentará la implementación, generando los artefactos correspondientes a la fase de análisis y diseño.

2.1. Descripción de la propuesta de solución

Luego de llevar a cabo un análisis exhaustivo de los conceptos y herramientas clave que se emplearán en esta investigación, se presenta la propuesta de solución en la Figura 3, la cual se describe de la siguiente manera.

El sistema permite al usuario indexar documentos PDF que son segmentados en fragmentos más pequeños (chunks). A cada fragmento se le calculan representaciones vectoriales (embeddings), las cuales se almacenan en una base de datos vectorial en memoria. Cuando el usuario realiza una consulta, esta se reformula en preguntas más específicas, que también se convierten en embeddings. Luego, el sistema ejecuta una búsqueda híbrida (semántica y léxica) para recuperar los k fragmentos más relevantes desde la base vectorial. Estos se reordenan según su pertinencia y se combinan con la consulta original para formar un prompt que es procesado por el modelo de lenguaje (LLM), el cual genera una respuesta. Si la respuesta no resuelve completamente la consulta, el sistema puede formular nuevas preguntas basadas en la información aún no cubierta, repitiendo el ciclo hasta alcanzar una respuesta satisfactoria o agotar un número máximo de iteraciones. Este enfoque permite enriquecer dinámicamente el contexto ofrecido al modelo, mejorando la precisión y pertinencia de las respuestas.

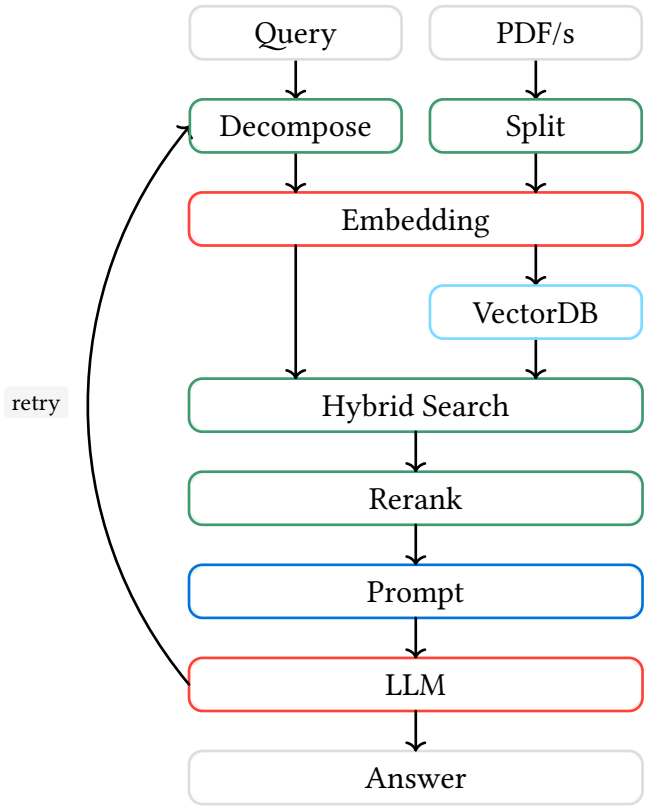


Figura 3: Propuesta de Solución (Elaboración Propia)

2.2. Requisitos de software

2.2.1. Requisitos funcionales

- **HU1**: Enviar consultas
- **HU2**: Enviar archivos PDF
- **HU3**: Generar respuestas
- **HU4**: Procesar archivos
- **HU5**: Buscar documentos relevantes
- **HU6**: Mostrar documentos recuperados y sus puntuaciones
- **HU7**: Regenerar respuesta
- **HU8**: Dar retroalimentación de una respuesta
- **HU9**: Editar una consulta previa
- **HU10**: Ajustar los parámetros del sistema
- **HU11**: Crear múltiples conversaciones
- **HU12**: Limpiar el chat

2.2.2. Requisitos no funcionales

Usabilidad

- **RnF1**: Se podrá interactuar de forma fácil por cualquier usuario.
- **RnF2**: Se podrá emplear el idioma Inglés y Español.

Rendimiento

- **RnF4**: El sistema permitirá que múltiples usuarios lo empleen a la vez.

Legales

- **RnF5**: Las herramientas seleccionadas para el desarrollo de la propuesta de solución estarán respaldadas por licencias libres.

Hardware

- **RnF7**: El sistema debe ejecutarse en una computadora con memoria RAM mínima de 8 gigabytes.
- **RnF8**: El sistema debe ejecutarse en una computadora con GPU Nvidia con memoria VRAM mínima de 8 gigabytes.

2.3. Descripción de las historias de usuario

En este capítulo se definen los requisitos funcionales y no funcionales del sistema, los cuales describen, respectivamente, las funcionalidades que debe ofrecer la aplicación y las restricciones o cualidades que debe cumplir (como rendimiento, usabilidad y seguridad). En la metodología XP, estos requisitos se especifican mediante historias de usuario, redactadas en un lenguaje claro y accesible por el cliente, permitiendo encapsular de forma concisa lo que el sistema debe realizar. Este enfoque facilita la priorización, estimación y validación de cada historia, garantizando una comunicación efectiva entre desarrolladores y usuarios, y permitiendo iteraciones de desarrollo ágiles y adaptables a las necesidades reales (Sommerville & Galipienso, 2005).

Tabla 3: Historia de usuario 1

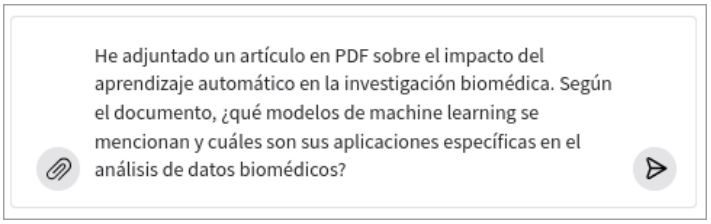
Historia de Usuario	
Número: 1	**Nombre**: Enviar consultas
Usuario: Investigador	
Prioridad de negocio: Alta	**Riesgo en desarrollo**: Bajo
Tiempo estimado: 1	**Iteración asignada**: 1
Programador Responsable: Joaquin Enrique Rivas Sánchez	
Descripción: El usuario debe poder escribir y enviar consultas al sistema.	
Prototipo Interfaz: 	

Tabla 4: Historia de usuario 2

Historia de Usuario	
Número: 2	**Nombre**: Enviar archivos PDF
Usuario: Investigador	
Prioridad de negocio: Alta	**Riesgo en desarrollo**: Bajo
Tiempo estimado: 1	**Iteración asignada**: 1
Programador Responsable: Joaquin Enrique Rivas Sánchez	
Descripción: El usuario debe poder enviar archivos PDF al sistema para su procesamiento de hasta 300 megabytes.	
Observaciones: En caso de intentar enviar otros formatos de archivo, se mostrara un mensaje de advertencia.	
Prototipo Interfaz:	

Historia de Usuario


Tabla 5: Historia de usuario 3

Historia de Usuario	
Número: 3	**Nombre**: Generar respuestas
Prioridad de negocio: Alta	**Riesgo en desarrollo**: Medio
Tiempo estimado: 2	**Iteración asignada**: 1
Programador Responsable: Joaquin Enrique Rivas Sánchez	
Descripción: El sistema debe generar respuestas basadas en las consultas realizadas por el usuario.	
Observaciones: En caso de fallar se muestra un mensaje de error.	

Tabla 6: Historia de usuario 4

Historia de Usuario	
Número: 4	**Nombre**: Procesar archivos
Usuario: Investigador	
Prioridad de negocio: Alta	**Riesgo en desarrollo**: Alto
Tiempo estimado: 3	**Iteración asignada**: 1
Programador Responsable: Joaquin Enrique Rivas Sánchez	
Descripción: El sistema debe procesar los archivos enviados y almacenar la información extraída.	

Tabla 7: Historia de usuario 5

Historia de Usuario	
Número: 5	**Nombre**: Buscar documentos relevantes
Prioridad de negocio: Alta	**Riesgo en desarrollo**: Alto
Tiempo estimado: 2	**Iteración asignada**: 1
Programador Responsable: Joaquin Enrique Rivas Sánchez	
Descripción: El sistema debe identificar y recuperar documentos relevantes a la consulta realizada.	
Observaciones: En caso de no encontrar documentos relevantes no devuelve nada.	

Tabla 8: Historia de usuario 6

Historia de Usuario	
Número: 6	**Nombre**: Mostrar documentos recuperados
Usuario: Investigador	
Prioridad de negocio: Media	**Riesgo en desarrollo**: Bajo
Tiempo estimado: 1	**Iteración asignada**: 2

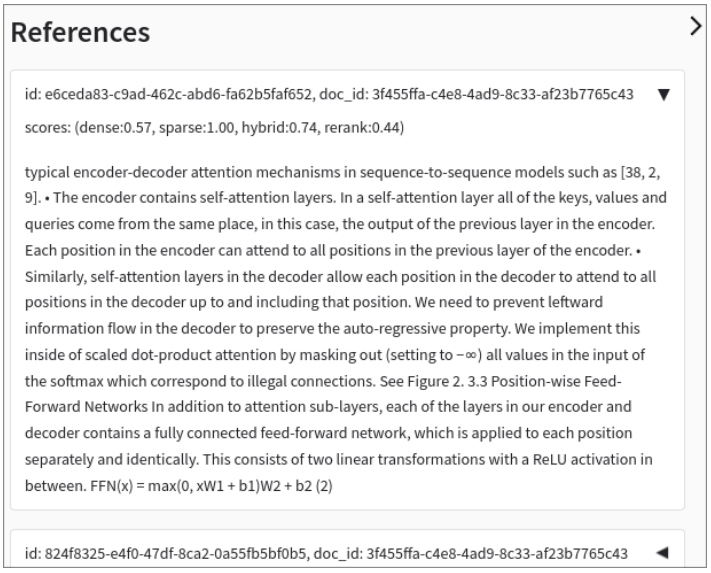
Historia de Usuario
Programador Responsable: Joaquin Enrique Rivas Sánchez
Descripción: El sistema debe mostrar una lista de documentos con sus respectivas puntuaciones de recuperación.
Observaciones: En caso de no recibir documentos recuperados no se mostrara nada.
Prototipo Interfaz: 

Tabla 9: Historia de usuario 7

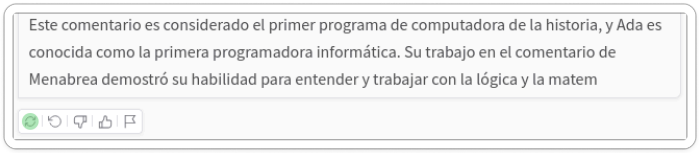
Historia de Usuario	
Número: 7	**Nombre**: Regenerar respuesta
Usuario: Investigador	
Prioridad de negocio: Media	**Riesgo en desarrollo**: Bajo
Tiempo estimado: 1	**Iteración asignada**: 2
Programador Responsable: Joaquin Enrique Rivas Sánchez	
Descripción: El usuario debe poder solicitar al sistema que regenere una respuesta si la inicial no satisface sus necesidades.	
Prototipo Interfaz: 	

Tabla 10: Historia de usuario 8

Historia de Usuario	
Número: 8	**Nombre**: Dar name: retroalimentación de una respuesta
Usuario: Investigador	
Prioridad de negocio: Media	**Riesgo en desarrollo**: Bajo
Tiempo estimado: 1	**Iteración asignada**: 2

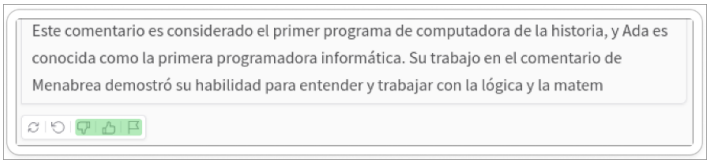
Historia de Usuario
Programador Responsable: Joaquin Enrique Rivas Sánchez
Descripción: El usuario debe poder proporcionar retroalimentación sobre las respuestas del sistema. Los criterios para retroalimentar son los siguientes: Me gusta, No me gusta, Alucinación, Inapropiado, Dañino.
Prototipo Interfaz: 

Tabla 11: Historia de usuario 9

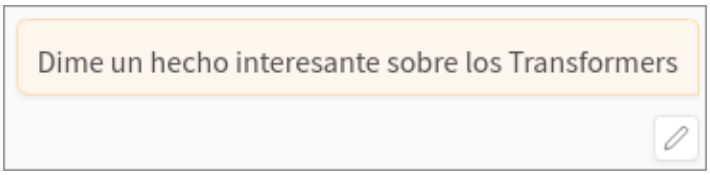
Historia de Usuario	
Número: 9	**Nombre**: Editar una consulta previa
Usuario: Investigador	
Prioridad de negocio: Media	**Riesgo en desarrollo**: Bajo
Tiempo estimado: 1	**Iteración asignada**: 2
Programador Responsable: Joaquin Enrique Rivas Sánchez	
Descripción: El sistema debe permitir la edición de consultas y generar nuevas respuestas basadas en los cambios.	
Prototipo Interfaz: 	

Tabla 12: Historia de usuario 10

Historia de Usuario	
Número: 10	**Nombre**: Ajustar los parámetros del sistema
Usuario: Investigador	
Prioridad de negocio: Baja	**Riesgo en desarrollo**: Medio
Tiempo estimado: 2	**Iteración asignada**: 2
Programador Responsable: Joaquin Enrique Rivas Sánchez	
Descripción: El usuario debe poder acceder a opciones de configuración para modificar parámetros específicos.	
Observaciones: Los parámetros incluyen: Temperatura del modelo, Máximo de tokens de salida, Penalización de frecuencia y de presencia	
Prototipo Interfaz:	

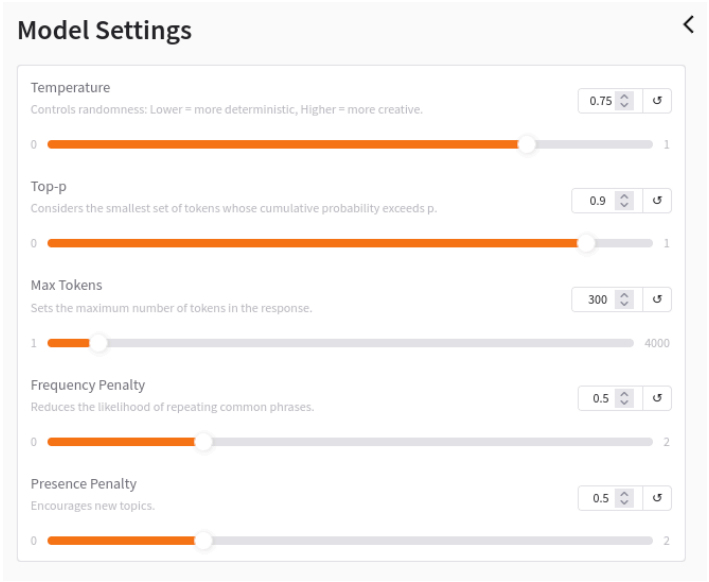
Historia de Usuario


Tabla 13: Historia de usuario 11

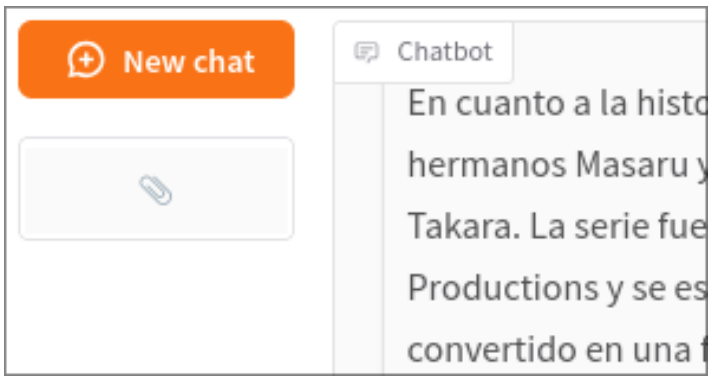
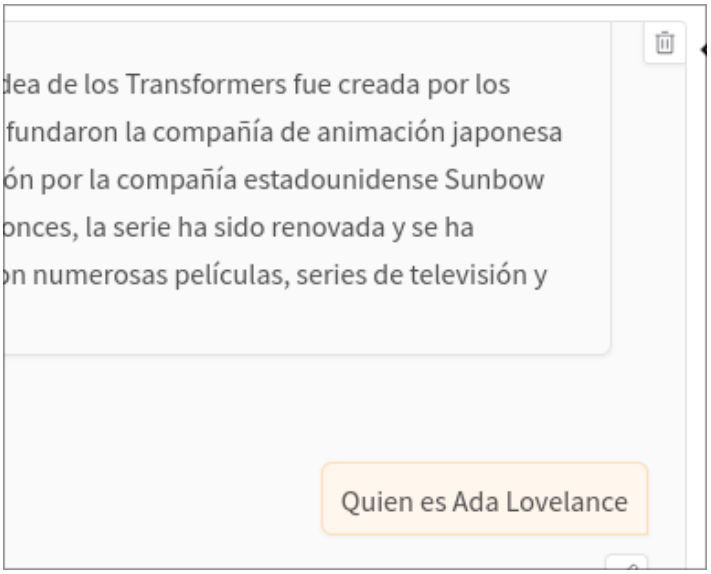
Historia de Usuario	
Número: 11	**Nombre**: Crear múltiples conversaciones
Usuario: Investigador	
Prioridad de negocio: Media	**Riesgo en desarrollo**: Bajo
Tiempo estimado: 1	**Iteración asignada**: 2
Programador Responsable: Joaquin Enrique Rivas Sánchez	
Descripción: El usuario debe poder iniciar múltiples conversaciones independientes para gestionar diferentes temas o consultas.	
Observaciones: Cada conversación maneja su propio historial y contexto.	
Prototipo Interfaz: 	

Tabla 14: Historia de usuario 12

Historia de Usuario	
Número: 12	**Nombre**: Limpiar el chat
Usuario: Investigador	
Prioridad de negocio: Baja	**Riesgo en desarrollo**: Bajo

Historia de Usuario	
Tiempo estimado: 1	**Iteración asignada**: 2
Programador Responsable: Joaquin Enrique Rivas Sánchez	
Descripción: El usuario debe poder limpiar el historial del chat para comenzar una nueva conversación.	
Prototipo Interfaz: 	

2.4. Plan de iteraciones

Habiendo identificado previamente las historias de usuario se debe crear el plan de iteraciones donde cada HU se convierte en tareas específicas de desarrollo y para cada uno se establecen pruebas. En cada ciclo se analizan las pruebas fallidas para prevenir que no vuelvan a ocurrir y ser corregidas.

Se acordaron 2 iteraciones que a continuación serán descritas:

- **Iteración 1**: Se desarrollan las HU 1,2,3,4,5 las cuales corresponden al envío de consultas y archivos PDF; la generación de respuestas; procesamiento de archivos; la búsqueda de documentos relevantes. Al finalizar la iteración se realizarán las pruebas.
- **Iteración 2**: Se desarrollan las HU 6,7,8,9,10,11,12 las cuales corresponden a la capacidad de mostrar documentos recuperados; regenerar, dar retroalimentación de las respuestas; editar una consulta previa; ajustar los parámetros del sistema; crear múltiples conversaciones; limpiar el chat y por último incluir citas en las respuestas. Al finalizar la iteración se realizarán las pruebas faltantes y la entrega final de la propuesta de solución.

En la Tabla 15 se muestra el plan de iteraciones y se incluye el tiempo estimado por iteración, así como las HU a desarrollar. Se tomó como unidad de medida 1 semana y cada una contaba de 5 días laborales, de los cuales se trabajarán 8 horas cada día. Se estima un total de duración de 4 meses y medio.

Tabla 15: Estimación de esfuerzo por historia de usuario

Iteración	Historias de usuario		Duración (semanas)
1	1	Enviar consultas	1
	2	Enviar archivos PDF	1

Iteración	Historias de usuario		Duración (semanas)
	3	Generar respuestas	2
	4	Procesar archivos	3
	5	Buscar documentos relevantes	2
2	6	Mostrar documentos recuperados	1
	7	Regenerar respuesta	1
	8	Dar retroalimentación de una respuesta	1
	9	Editar una consulta previa	1
	10	Ajustar los parámetros del sistema	2
	11	Crear múltiples conversaciones	1
	12	Limpiar el chat	1
Total			17

2.5. Arquitectura de software

La arquitectura de software se refiere a la estructura organizativa de un sistema de software, que incluye sus componentes principales y las relaciones entre ellos. Es el diseño de alto nivel que guía la evolución y el desarrollo del sistema, asegurando que cumpla con sus objetivos funcionales y no funcionales. Un patrón arquitectónico es una solución recurrente a problemas comunes en la construcción de software, proporcionando una estructura predefinida que se puede aplicar en diferentes contextos, ayudando a resolver desafíos de diseño. Finalmente, un estilo arquitectónico es un conjunto de reglas y restricciones que define la organización de los componentes del sistema y las interacciones entre ellos, reflejando un enfoque particular para abordar problemas de diseño en una categoría de sistemas. Un estilo puede incluir varios patrones arquitectónicos que estructuran el sistema de acuerdo con principios y prácticas específicas (Richards & Ford, 2020). En la propuesta de solución se hizo uso de la arquitectura por capas

Arquitectura por Capas

La arquitectura en capas es un patrón estructural que organiza un sistema en niveles jerárquicos, donde cada capa solo se comunica con la capa inmediatamente superior o inferior. Esta regla refuerza la separación de responsabilidades, facilita el mantenimiento y permite una evolución más controlada del sistema. Un aspecto clave de esta arquitectura es la distinción entre capas abiertas y cerradas: en una **capa cerrada**, el flujo debe pasar secuencialmente por cada capa, sin poder ser saltada; en cambio, una **capa abierta** permite que otras capas superiores accedan directamente a ella, lo que flexibiliza el diseño pero puede debilitar el aislamiento entre niveles (Richards & Ford, 2020).

Para la propuesta de solución se definieron 4 capas (Figura 4):

1. **Capa de Presentación (UI)**: Implementada con Gradio, esta capa gestiona la interacción con el usuario, permitiéndole introducir consultas y visualizar las respuestas del sistema.
2. **Capa de Lógica**: Orquesta el flujo general de la aplicación, conectando los distintos componentes.

3. **Capa de Inferencia (AI/ML)**: Aquí se realiza la generación de respuestas mediante modelos de lenguaje. Esta capa encapsula la lógica relacionada con los clientes LLM y su ejecución.
4. **Capa de Persistencia**: Representada por una "base del conocimiento" (Vector DB) almacenada en memoria.

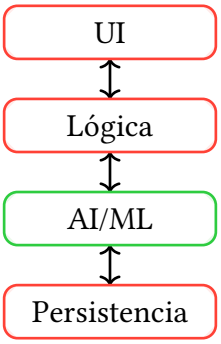


Figura 4: Capas del Sistema (Elaboración propia)

2.6. Patrones de diseño

Un patrón de diseño es una solución reutilizable y probada para problemas comunes de diseño en el desarrollo de software. A diferencia de los patrones arquitectónicos, que se enfocan en la estructura general del sistema, los patrones de diseño abordan problemas específicos en la implementación y organización del código a nivel de componentes y clases. Estos patrones ayudan a mejorar el rendimiento, mantenibilidad, la escalabilidad y la flexibilidad del software, proporcionando estructuras estandarizadas que facilitan la comunicación entre desarrolladores (Johnson et al., 2005).

2.6.1. Patrón Generador

En este proyecto, el patrón Generador (Generator) se emplea para implementar iteradores ligeros que evitan la creación de colecciones completas en memoria. Gracias a él, la interfaz gráfica puede recibir actualizaciones parciales—por ejemplo, fragmentos de respuesta o estados de avance—en cuanto estén disponibles, en lugar de esperar a que todo el procesamiento de datos finalice. Esto resulta esencial para el streaming de resultados en aplicaciones interactivas, pues mejora la capacidad de respuesta y la experiencia del usuario (Rossum & Jr., 2011).

```python
def create_query_plan(
    query: str, plan: QueryPlan
) -> Generator[ChatMessageAndChunk, None, None]:
    # Mensaje indicando el inicio de la operación
    yield (
        ChatMessage(
            role="assistant",
            content=f"Creating Query Plan for {query}",
            metadata={
                "title": "✍️ Creating Query Plan",
                "status": "pending",
            },
        ),
        [],
    )
```

```
    # ... (Lógica para generar el plan con LLM) ...
    # Mensaje indicando el fin de y el resultado
    yield (
        ChatMessage(
            role="assistant",
            content=str(plan),
            metadata={ # ... metadata ...},
        ),
        [],
    )
```

2.6.2. Máquina de Estados

Una Máquina de Estados es un patrón que permite a un objeto alterar su comportamiento cuando su estado interno cambia. Se utiliza para modelar sistemas que pueden existir en un número finito de estados y transiciones entre ellos en respuesta a eventos o condiciones. Es útil para gestionar flujos de control complejos y secuencias de operaciones ordenadas. En este código, el proceso general de la función `ask` sigue una secuencia (planificar, recuperar, generar, validar, refinar) que puede iterar hasta alcanzar un estado final ("completo"). (Taylor & Taylor, 2021).

```python
state = GenerationState() # Inicializa el objeto que contendrá el estado
while not state.complete and iterations < max_iterations:
    # ... (Pasos de planificación, recuperación, generación) ...

    # El paso de validación actualiza explícitamente el estado
    yield from validate_answer(plan, state) # Puede cambiar state.complete

    # Si el estado aún no es completo, se refina el plan para la siguiente iteración
    if not state.complete:
        yield from refine_query_plan(plan, state)

    iterations += 1

# Al salir del bucle, se considera que se ha alcanzado un estado final
yield state.answer, chunks
```

2.6.3. Gestión de Configuración

Consiste en separar los datos de configuración (parámetros que pueden variar entre entornos como desarrollo o producción, rutas de los modelos, etc.) del código fuente de la aplicación. Esto mejora la flexibilidad, portabilidad y mantenibilidad, ya que permite modificar el comportamiento de la aplicación sin cambiar el código, simplemente ajustando las variables de entorno o configuración (Washizaki, 2024).

```python
class Settings(BaseModel):
    EMBEDDING_MODEL: str = os.getenv("EMBEDDING_MODEL", "BAAI/bge-m3")
    RERANKER_MODEL: str = os.getenv("RERANKER_MODEL", "BAAI/bge-reranker-v2-m3")
    LLM_MODEL: str = os.getenv("LLM_MODEL", "deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B")
```

```
    DTYPE: str = os.getenv("DTYPE", "float16")
    ...

# otro archivo.py
from settings import settings
llm_model = vLLMClient() if settings.ENVIRONMENT == "prod" else OpenAIClient()
```

2.6.4. Inyección de Dependencias

Es un patrón de diseño en el que un objeto recibe las otras instancias de objetos (sus «dependencias») que necesita, desde una fuente externa, en lugar de crearlas internamente. Esto promueve el bajo acoplamiento entre componentes y mejora la testeabilidad, ya que las dependencias pueden ser fácilmente reemplazadas por implementaciones alternativas o mocks durante las pruebas. Es una forma de implementar el principio de Inversión de Control (IoC) (Sobernig & Zdun, 2010).

Python

```python
def ask(
    message: Message,
    history: list[ChatMessage],
    db: KnowledgeBase, # <-- 'db' es una dependencia inyectada
    temperature: float,
    max_tokens: int,
    top_p: float,
    frequency_penalty: float,
    presence_penalty: float,
    max_iterations: int = 3,
) -> Generator[ChatMessageAndChunk, None, None]:
    query, files = extract_message_content(message)

    if files:
        yield from insert_files(files, db) # <- Pasa la dependencia 'db'

    use_rag = not db.is_empty
    if use_rag:
        # ...
        yield from iterative_retrieval(plan, db, chunks)
```

2.7. Conclusiones parciales

Se llegaron a las siguientes conclusiones:

- La descripción general de la propuesta de solución ayudó a sentar las bases para la comprensión del flujo de la información y el contexto de la solución.
- Se definieron las funcionalidades específicas del sistema a través de las historias de usuario, así como los requisitos no funcionales.
- Se obtuvo una planificación del tiempo de desarrollo a través del plan de iteraciones, así como el orden en el que se implementarán las tareas.
- Se implementaron patrones de diseño con el fin de garantizar una mayor extensibilidad y mantenibilidad de la solución.

Implementación y realización de pruebas

Este capítulo presenta la implementación de la solución propuesta y las pruebas realizadas para evaluar su funcionamiento. Se detallan los mecanismos empleados para la verificación del sistema, así como los resultados obtenidos. Además, se analiza la transformación alcanzada en el objeto de estudio y la factibilidad técnica de la solución.

3.1. Tareas de Ingeniería

Tabla 16: Tarea de Ingeniería 1

Tarea de Ingeniería - HU 1	
Nombre de Tarea: Enviar consultas	
Tipo de Tarea: Desarrollo	**Tiempo Estimado:** 1
Fecha de inicio: 2025-01-03	**Fecha de fin:** 2025-01-10
Descripción: Se realiza la integración de una caja de texto que contiene un botón de enviar, esta se encarga de enviar la consulta al componente que se encarga de procesarla.	

Tabla 17: Tarea de Ingeniería 2

Tarea de Ingeniería - HU 2	
Nombre de Tarea: Enviar archivos PDF	
Tipo de Tarea: Desarrollo	**Tiempo Estimado:** 1
Fecha de inicio: 2025-01-10	**Fecha de fin:** 2025-01-17
Descripción: En la caja de texto de enviar consulta se añade un botón para agregar un documento,este tiene un limite de 300mb. Cuando el usuario envía la consulta los archivos son procesados en el backend.	

Tabla 18: Tarea de Ingeniería 3

Tarea de Ingeniería - HU 3	
Nombre de Tarea: Generar respuestas	
Tipo de Tarea: Desarrollo	**Tiempo Estimado:** 2
Fecha de inicio: 2025-01-17	**Fecha de fin:** 2025-01-31
Descripción: La consulta se divide en sub consultas, estas se utilizan para obtener documentos relevantes, luego con estos documentos se genera una respuesta, si la respuesta esta completa (la pregunta fue respondida) se le manda al usuario, sino se inicia una búsqueda de los elementos faltantes, esto se puede repetir hasta que se completa la respuesta o se alcanza el máximo numero de iteraciones.	

Tabla 19: Tarea de Ingeniería 4

Tarea de Ingeniería - HU 4	
Nombre de Tarea: Procesar archivos	
Tipo de Tarea: Desarrollo	**Tiempo Estimado:** 3

Tarea de Ingeniería - HU 4	
Fecha de inicio: 2025-01-31	**Fecha de fin:** 2025-02-21
Descripción: Los archivos son divididos en pequeños pedazos que se encuentran dentro de un umbral de tokens que el modelo de embeddings puede leer, luego se generan los embeddings y se almacenan en la base de datos vectorial en memoria.	

Tabla 20: Tarea de Ingeniería 5

Tarea de Ingeniería - HU 5	
Nombre de Tarea: Buscar documentos relevantes	
Tipo de Tarea: Desarrollo	**Tiempo Estimado:** 2
Fecha de inicio: 2025-02-21	**Fecha de fin:** 2025-03-07
Descripción: Se crean los embeddings de la consulta, estos se utilizan para encontrar en la base de datos embeddings que compartan cierta similitud semántica (mediante la similitud del coseno) así como lexicográfica, luego estos se reordenan usando el modelo Reranker.	

Tabla 21: Tarea de Ingeniería 6

Tarea de Ingeniería - HU 6	
Nombre de Tarea: Mostrar documentos recuperados	
Tipo de Tarea: Desarrollo	**Tiempo Estimado:** 1
Fecha de inicio: 2025-03-07	**Fecha de fin:** 2025-03-14
Descripción: Los documentos recuperados son mostrados en una barra lateral derecha, en forma de lista donde cada elemento es colapsable.	

Tabla 22: Tarea de Ingeniería 7

Tarea de Ingeniería - HU 7	
Nombre de Tarea: Regenerar respuesta	
Tipo de Tarea: Desarrollo	**Tiempo Estimado:** 1
Fecha de inicio: 2025-03-14	**Fecha de fin:** 2025-03-21
Descripción: Se activa la opción de Gradio para regenerar la respuesta.	

Tabla 23: Tarea de Ingeniería 8

Tarea de Ingeniería - HU 8	
Nombre de Tarea: Dar retroalimentación de una respuesta	
Tipo de Tarea: Desarrollo	**Tiempo Estimado:** 1
Fecha de inicio: 2025-03-21	**Fecha de fin:** 2025-03-28
Descripción: Se activa la opción de Gradio para dar retroalimentación, la cual se guarda en un archivo.	

Tabla 24: Tarea de Ingeniería 9

Tarea de Ingeniería - HU 9	
Nombre de Tarea: Editar una consulta previa	
Tipo de Tarea: Desarrollo	**Tiempo Estimado:** 1

Tarea de Ingeniería - HU 9	
Fecha de inicio: 2025-03-28	**Fecha de fin:** 2025-04-04
Descripción: Se activa la opción de Gradio para dar editar la consulta.	

Tabla 25: Tarea de Ingeniería 10

Tarea de Ingeniería - HU 10	
Nombre de Tarea: Ajustar los parámetros del sistema	
Tipo de Tarea: Desarrollo	**Tiempo Estimado:** 2
Fecha de inicio: 2025-04-04	**Fecha de fin:** 2025-04-18
Descripción: Se implementa una barra lateral izquierda con varios campos para seleccionar los parámetros de generación, estos se almacenan en el local storage del navegador, y se le pasa a la logica del backend para instruir la generación.	

Tabla 26: Tarea de Ingeniería 11

Tarea de Ingeniería - HU 11	
Nombre de Tarea: Crear múltiples conversaciones	
Tipo de Tarea: Desarrollo	**Tiempo Estimado:** 1
Fecha de inicio: 2025-04-18	**Fecha de fin:** 2025-04-25
Descripción: Se activa la opción en Gradio para crear múltiples conversaciones.	

Tabla 27: Tarea de Ingeniería 12

Tarea de Ingeniería - HU 12	
Nombre de Tarea: Limpiar el chat	
Tipo de Tarea: Desarrollo	**Tiempo Estimado:** 1
Fecha de inicio: 2025-04-25	**Fecha de fin:** 2025-05-02
Descripción: Se activa la opción en Gradio para limpiar el chat.	

3.2. Evaluación

La evaluación de aplicaciones que integran LLMs exige protocolos más estrictos que el software convencional, pues sus salidas son inherentemente no deterministas[3]. Por ello, se recurre a métricas cuantitativas (e.g., exactitud, coherencia, robustez) y a revisiones humanas estructuradas para calibrar tanto la fidelidad de los resultados como su adecuación al contexto de uso (Es et al., 2023).

Para realizar la evaluación se creó una herramienta externa (Figura 9) cuyo objetivo principal fue automatizar el proceso de generación del conjunto de datos (dataset) de prueba. A partir de esta herramienta, se construyó un dataset específico bajo condiciones controladas. Este conjunto de datos presenta las siguientes características principales:

- Incluye un total de 80 pares pregunta-respuesta (4 pares por cada tema), cada uno acompañado por su correspondiente id de contexto recuperado.

[3]Cada invocación puede generar respuestas distintas incluso con el mismo prompt

- Las preguntas fueron generadas de forma automática (usando un LLM), y están orientadas a distintos dominios (ver Tabla 28).
- El dataset fue conformado de diferentes artículos de Wikipedia en versión inglés y español
- El dataset fue estructurado en formato `.jsonl`, lo que facilita su uso en procesos automáticos de evaluación y su integración con herramientas como RAGAS.

Tabla 28: Documentos seleccionados (Elaboración propia)

Area	Temas
Matemáticas	Números Primos, Algebra lineal, Calculo, Probabilidad
Ciencias de la Computación	Algoritmo, Estructura de datos, Inteligencia artificial, Programación de computadoras
Biología	Célula (biología), Genética, Evolución, Ecología
Física	Mecánica clásica, Electromagnetismo, Mecánica cuántica, Termodinámica
General	Batman, Perro salchicha, Teoría conspirativa, Religión

A continuación se definen las métricas de evaluación utilizadas.

3.2.1. Métricas de Generación

Las métricas de generación definidas tienen como fin valorar la calidad de las respuestas producidas una vez incorporada la información recuperada. Su objetivo es comprobar que las respuestas no solo sean coherentes y precisas, sino que atiendan de forma directa la pregunta planteada (Yu et al., 2025).

A continuación, se describen las principales métricas introducidas por RAGAS (Es et al., 2023):

- **Faithfulness**: Una respuesta es fiel cuando no introduce hechos ajenos o inexactitudes que no estén respaldadas por las fuentes consultadas. Esta métrica es especialmente útil para ayudar a detectar las alucinaciones (Es et al., 2023).

  Para implementarla se siguen los siguientes pasos:

  1. Extraer afirmaciones atómicas de la respuesta generada utilizando un modelo de lenguaje (LLM).
  2. Para cada afirmación, determinar si puede inferirse del contexto recuperado.

  $$\text{Faithfulness} = \frac{\#\text{ de afirmaciones correctas}}{\#\text{ total de afirmaciones}}$$

- **Factual Correctness**: Que tan bien la respuesta responde la pregunta del usuario (Es et al., 2023).

  Para implementarla se siguen los siguientes pasos:

  1. Usar un LLM para generar posibles preguntas a las que la respuesta podría corresponder.
  2. Calcular vectores de embedding tanto para la pregunta original como para las preguntas generadas.
  3. Calcular la similitud de coseno entre estos embeddings.
  4. Tomar el promedio de las similitudes como la puntuación de relevancia de la respuesta.

Answer Relevance = Promedio de la similitud del coseno entre la pregunta original y generada

- **Context Recall**: Determina si la información recuperada es suficiente para responder la pregunta, y no contiene contenido excesivamente irrelevante (Es et al., 2023).

Para implementarla se siguen los siguientes pasos:

1. Usar un LLM para identificar afirmaciones que son relevantes para responder la pregunta.
2. Calcular de la siguiente forma

$$\text{Context Recall} = \frac{\#\text{ de afirmaciones relevantes}}{\#\text{ total de afirmaciones}}$$

3.2.2. Resultados

En la Tabla 29 se muestran los parámetros de generación utilizados.

Tabla 29: Parámetros de Generación

Parámetro	Valor	Descripción
Context Window	8196	Límite máximo de tokens que el modelo puede considerar como entrada
Temperature	0.25	Controla la aleatoriedad en la generación; valores bajos producen respuestas más deterministas
Max Tokens	500	Número máximo de tokens que el modelo puede generar como salida
Top P	0.95	Limita la probabilidad acumulada de las opciones consideradas; afecta la diversidad de la respuesta
Frequency Penalty	0.5	Penaliza tokens que ya han aparecido frecuentemente, reduciendo repeticiones
Presence Penalty	1.2	Penaliza tokens ya mencionados, fomentando la aparición de ideas nuevas

Los resultados obtenidos (Figura 5) a través de la evaluación con RAGAS reflejan un rendimiento sólido en el componente de recuperación de contexto, pero revelan importantes deficiencias en la fase de generación de respuestas.

- El puntaje de `Context Recall` fue alto (0.9781), lo que indica que el sistema es eficaz para recuperar información relevante; en otras palabras, el modelo accede correctamente a los datos necesarios para responder la mayoría de las preguntas.
- No obstante, la métrica `Faithfulness` fue baja (0.4543), lo que evidencia un problema serio: el módulo generador frecuentemente produce respuestas que no están plenamente fundamentadas en el contexto proporcionado. Esto sugiere que el modelo recurre a conocimientos previos aprendidos durante el preentrenamiento, ignora partes clave del contexto, o interpreta información de manera imprecisa. En consecuencia, se compromete la confianza del usuario, ya que las respuestas pueden contener afirmaciones incorrectas o incluso alucinaciones.
- Esta situación también se refleja en el puntaje de `Factual Correctness` (0.6242), que indica una precisión moderada: las respuestas son parcialmente correctas, aunque inconsistentes. La generación es adecuada en algunos casos, pero no puede garantizarse una fidelidad completa con la información recuperada.

En conjunto, estos resultados conducen a tres implicaciones clave: (1) la recuperación funciona de forma confiable y robusta; (2) el modelo generador requiere ajustes para mejorar la fidelidad a la evidencia contextual; y (3) es necesario implementar medidas correctivas como el perfeccionamiento de los prompts (instrucciones explícitas para que el modelo solo utilice el contexto dado), garantizar la claridad y relevancia de los fragmentos recuperados, realizar fine-tuning con datos del dominio específico y, finalmente, aplicar filtros o mecanismos de control para reducir la generación de contenido erróneo o no verificado. En resumen, el sistema muestra un buen potencial, pero necesita mejoras en la etapa de indexado para alcanzar una mayor precisión y confiabilidad.

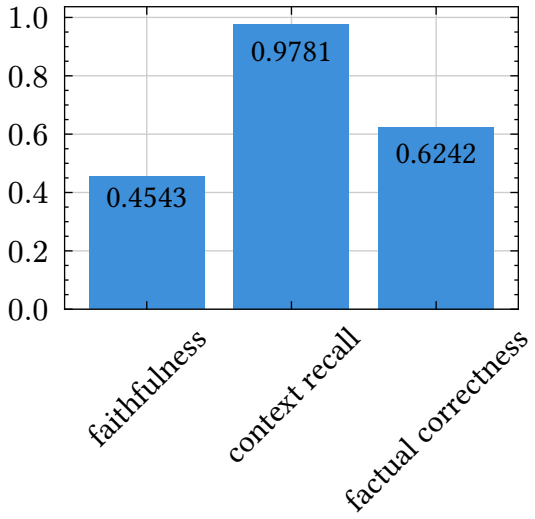


Figura 5: Resultados de RAGAS (Elaboración propia)

3.3. Pruebas de Rendimiento

En aplicaciones como los sistemas RAG, donde intervienen múltiples componentes como modelos de embeddings, y LLMs, el rendimiento puede impactar directamente en la experiencia del usuario. Estas pruebas permiten identificar cuellos de botella, tiempos de respuesta inadecuados o requerimientos excesivos de recursos, aspectos críticos especialmente en contextos donde se espera una interacción fluida y en tiempo real. Además, los resultados de estas pruebas ayudan a tomar decisiones informadas sobre optimización, escalabilidad y viabilidad del sistema en distintos entornos de implementación.

Para ello, se realizaron microbenchmarks[4] con el fin de medir el rendimiento individual de componentes clave del sistema, como el generador de embeddings, el reranker y el modelo generativo.

Las pruebas fueron ejecutadas en el siguiente hardware:

- i7 7700K
- 16GB RAM
- RTX 2070 (8GB VRAM)

3.3.1. LLM: Tiempos y Memoria

Para cuantificar de forma precisa el rendimiento del componente de generación de texto, se implementó un script que simula tres escenarios de uso mediante conjuntos de prompts de distinta longitud (Short ~ 34 tokens, Medium ~ 128 tokens y Long ~ 342 tokens). Tras realizar

[4]Es una prueba que evalúa el rendimiento de una unidad específica del sistema de forma aislada, permitiendo obtener métricas detalladas de su comportamiento.

una fase de calentamiento de tres invocaciones al modelo, el script ejecuta cinco iteraciones cronometradas con `timeit` y calcula dos métricas fundamentales: la latencia[5] y el throughput[6].

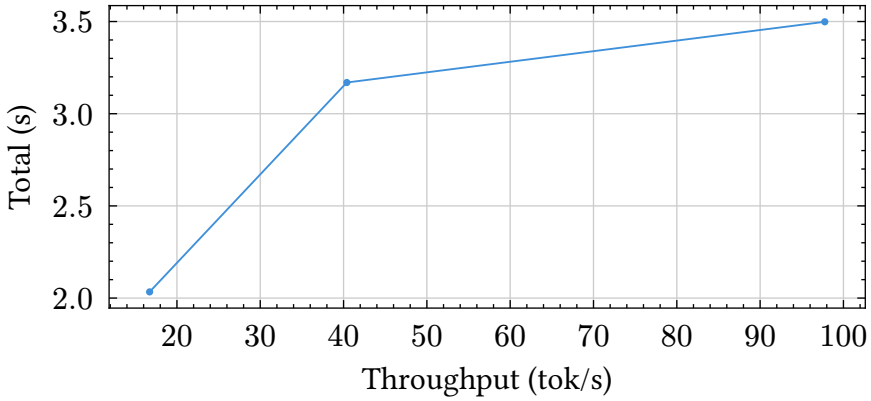


Figura 6: Benchmark LLM (Elaboración propia)

Los resultados (Figura 6) muestran que, al aumentar la longitud del prompt, la latencia crece de 2,03 s (Short) a 3,50 s (Long), mientras que el throughput pasa de 16,7 tok/s a 97,8 tok/s. Esto indica que el modelo procesa lotes más grandes de forma más eficiente —generando muchos más tokens por segundo— aunque a costa de un ligero aumento en el tiempo total de respuesta, que en todo caso se mantiene dentro de un rango aceptable para aplicaciones interactivas.

Más rendimiento puede ser obtenido si se optimizan los parámetros de vLLM y se hace uso de batching, adicionalmente el uso de formatos cuantizados `.gguf` tiene un soporte experimental y no está completamente optimizado.

Por otro lado, en cuestión de consumo de memoria, se puede utilizar la siguiente fórmula para determinar la cantidad de VRAM o RAM requerida para ejecutar el modelo. Para ejecutar un modelo de 1.5B de parámetros con una cuantización de 8 bits, el cálculo sería el siguiente.

$$\mathrm{M} = \frac{P * 4B}{32/Q} * 1.2 = \frac{1.5 * 4B}{32/8} * 1.2 = 1.80$$

Tabla 30: Memoria Requerida (GB)

Símbolo	Descripción
M	Memoria de la GPU expresada en gigabytes
P	Cantidad de parámetros en el modelo
$4B$	4 bytes, expresando los bytes usados para cada parámetro
32	Hay 32 bits en 4 bytes
Q	Cantidad de bits que se deben usar para cargar el modelo. 32, 16, 8, 4 bits.
1.2	Representa un 20% de sobrecarga por cargar elementos adicionales en la memoria de la GPU.

3.3.2. Embeddings: Tiempos y Memoria

Para evaluar el rendimiento del componente de generación de embeddings del sistema, se desarrolló un script que mide el tiempo promedio de codificación de un conjunto de oraciones

[5]Tiempo total que tarda el sistema en procesar una petición y ofrecer la respuesta completa

[6]Tasa de producción del sistema, expresada como la cantidad de unidades procesadas (en este caso, tokens por segundo

bajo diferentes configuraciones del modelo de embeddings. Las pruebas se realizaron sobre tres conjuntos de entrada: corto (2 oraciones - 17 tokens), medio (100 oraciones - 850 tokens) y largo (1000 oraciones - 8500 tokens), utilizando distintos tamaños de lote ("batch")[7].

Se consideraron cinco configuraciones del modelo:
1. solo embeddings densos
2. solo dispersos
3. solo ColBERT
4. combinación de densos + dispersos
5. combinación completa (densos + dispersos + ColBERT).

Para cada configuración, se realizó una fase de calentamiento inicial, seguida de 10 ejecuciones cronometradas, cuyo tiempo promedio fue registrado. Esta metodología permitió comparar el rendimiento relativo de cada variante del modelo bajo diferentes volúmenes de entrada.

En la Figura 7 se muestra los resultados, donde crear los embeddings para 1000 oraciones no supera los **2 segundos**.

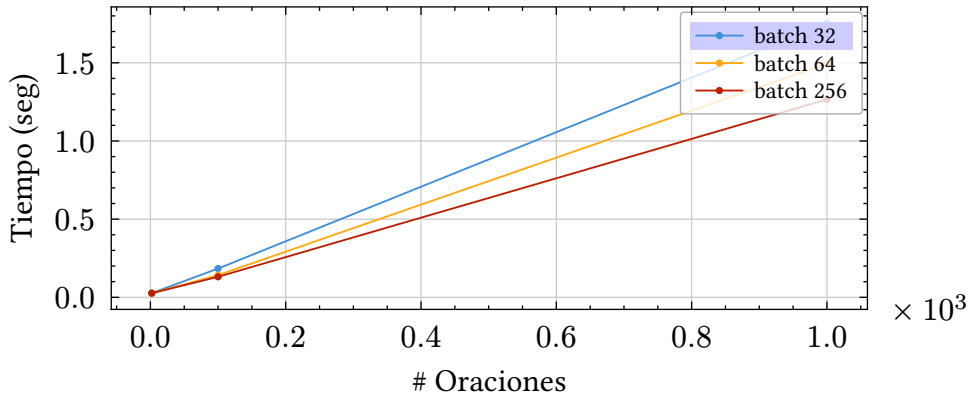


Figura 7: Benchmark Embeddings (Elaboración propia)

Se observó que el modelo de embeddings utilizado ocupa ≈ 2 GB VRAM.

3.4. Pruebas de Unidad

Las pruebas de unidad constituyen la primera línea de defensa para garantizar la calidad del sistema, validando el comportamiento correcto de funciones y componentes individuales de forma aislada. Estas pruebas se enfocan en unidades mínimas de código —como funciones, clases o métodos— con el objetivo de detectar errores lo antes posible en el ciclo de desarrollo. En este proyecto, se implementaron pruebas de unidad para cubrir los componentes clave del sistema, asegurando que cada parte funcione según lo esperado bajo diversas condiciones. Además, se utilizaron técnicas como la inyección de dependencias y el uso de mocks para aislar correctamente el entorno de ejecución y evitar efectos colaterales (Pressman & Maxim, 2019). A continuación, se describen los módulos evaluados y los criterios utilizados para definir los casos de prueba.

Los módulos evaluados incluyen:

1. Componente de compresión de contexto: se validó que la reducción de contexto conserve la información relevante y no introduzca distorsiones.

[7]El batching es una técnica que consiste en procesar múltiples entradas al mismo tiempo, optimizando así el uso de recursos y acelerando la inferencia en modelos.

2. Constructor de prompts: se verificó la correcta integración de los distintos elementos del prompt, así como la gestión de tokens.
3. Cliente LLM: se evaluó su comportamiento ante distintas configuraciones de generación y la correcta gestión de errores.
4. Sistema de recuperación: se comprobó que las consultas devuelvan resultados coherentes con los criterios esperados de relevancia.
5. Módulo de puntuación: se validó la aplicación correcta de las métricas y su influencia en el ordenamiento final.

Para cada componente, se definieron casos de prueba que cubren tanto escenarios típicos como condiciones límite o errores esperados. Se utilizaron técnicas de aislamiento mediante mocks para simular dependencias externas como llamadas a modelos, permitiendo así verificar el comportamiento interno sin efectos colaterales ni dependencias del entorno.

Algunos ejemplos se muestran a continuación:

Tabla 31: Caso de Prueba 1

Caso de Prueba 1	
Descripción: Verifica que la función `compress_history` conserve todo el historial si está dentro del límite de tokens.	
Entrada	Historial de chat con 3 mensajes cortos, límite de 100 tokens.
Salida Esperada	La función devuelve una lista con los 3 mensajes sin truncar.
Evaluación de la Prueba: Satisfactoria	

Tabla 32: Caso de Prueba 2

Caso de Prueba 2	
Descripción: Evalúa que `generate_answer` utilice correctamente el modelo para generar respuestas en flujo.	
Entrada	Consulta “What is AI?”, historial de 3 mensajes, 2 chunks de contexto y modelo simulado.
Salida Esperada	Se genera una secuencia de respuestas en orden: “Step 1”, “Step 2”, “Answer”.
Evaluación de la Prueba: Satisfactoria	

Tabla 33: Caso de Prueba 3

Caso de Prueba 3	
Descripción: Verifica que el modelo de embeddings puede codificar entradas múltiples y devuelve vectores `dense`, `sparse` y `colbert` correctamente.	
Entrada	Lista de textos: `['text1', 'text2', 'text3']`, con flags `return_dense=True`, `return_sparse=True`, `return_colbert=True`
Salida Esperada	Diccionario con claves `dense`, `sparse` y `colbert` con tres elementos cada uno.
Evaluación de la Prueba: Satisfactoria	

Tabla 34: Caso de Prueba 4

Caso de Prueba 4
Descripción: Prueba que dense_similarity devuelve una matriz de similitud con la forma esperada (consultas vs documentos).

Caso de Prueba 4	
Entrada	Vectores densos de 1 consulta y 2 documentos aleatorios
Salida Esperada	Matriz de salida de forma `(1, 2)`.
Evaluación de la Prueba: Satisfactoria	

Tabla 35: Caso de Prueba 5

Caso de Prueba 5	
Descripción: Valida el cálculo de similitud `ColBERT` tomando el promedio de la mejor coincidencia por token de la pregunta.	
Entrada	1 pregunta (2 tokens) y 1 fragmento de texto (3 tokens) con vectores explícitos y normalizados.
Salida Esperada	Similitud promedio esperada: 0.95.
Evaluación de la Prueba: Satisfactoria	

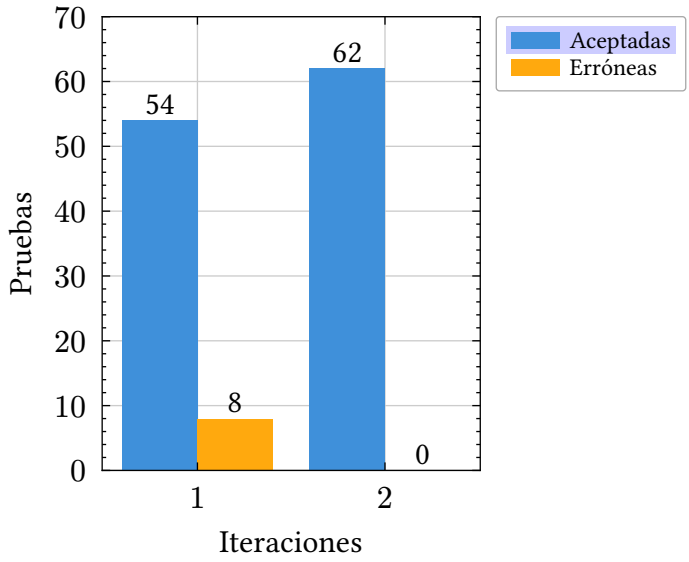


Figura 8: Resultados de las pruebas unitarias por iteración (Elaboración propia)

3.5. Conclusiones parciales

A partir del proceso de implementación y verificación de la solución propuesta, se derivan las siguientes conclusiones:

- Las pruebas de unidad y los mecanismos de evaluación aplicados demuestran que cada componente del sistema funciona conforme a lo esperado.
- Las pruebas de rendimiento revelan tiempos de respuesta adecuados, incluso con entradas de gran tamaño, así como un uso efectivo de recursos computacionales.
- Se incorporaron metodologías especializadas para la validación de sistemas con LLM, que van más allá del paradigma tradicional del software. El uso de métricas de generación aporta un marco sólido para la evaluación de sistemas RAG.

En conjunto, estos elementos validan la veracidad, fiabilidad y factibilidad técnica de la solución desarrollada, sentando las bases para su posible extensión, mejora, adaptación o implementación en contextos reales.

Conclusiones Finales

Con el desarrollo de la herramienta se puede afirmar que:

- Se logró una revisión integral de los Modelos de Lenguaje de Gran Tamaño (LLM) y su extensión mediante Generación Aumentada por Recuperación (RAG), lo cual permitió sustentar conceptualmente el diseño de la herramienta propuesta.
- En el análisis de sistemas homólogos, se identificaron limitaciones clave de soluciones existentes, lo que orientó de forma efectiva la definición de los requisitos funcionales y no funcionales del sistema.
- La metodología XP facilitó el desarrollo incremental, resultando en un prototipo funcional acorde al objetivo general planteado.
- El prototipo demostró viabilidad en hardware modesto, proporcionando una base sólida para futuras mejoras en generación y actualización documental continua.

Recomendaciones

- Extender fuentes de información mediante agentes web que se integren con plataformas de investigación cubanas y repositorios académicos locales.
- Explorar lenguajes y técnicas de alto rendimiento (C/C++ o Mojo) para aprovechar mejor los recursos.
- Integrar un mejor preprocesamiento de los archivos, utilizando OCR y detección del layout y fórmulas matemáticas.
- Evaluar arquitecturas distribuidas, desplegando modelos ligeros y especializados en nodos heterogéneos para mejorar escalabilidad.
- Combinar fine-tuning con RAG para aumentar la fidelidad de las respuestas ajustando modelos a dominios concretos.
- Explorar GraphRAG para integrar grafos de conocimiento, para mejorar la precisión, contextualización y relevancia en la generación de respuestas basadas en grandes conjuntos de datos.

Bibliografía

Castro, O., Bruneau, P., Sottet, J.-S., & Torregrossa, D. (2023,). *Landscape of High-performance Python to Develop Data Science and Machine Learning Applications*. https://arxiv.org/abs/2302.03307

ChatGPT. (2025,). https://chatgpt.com/

ChatPDF AI | Chat with any PDF | Free. (2025,). https://www.chatpdf.com/

Chen, J., Xiao, S., Zhang, P., Luo, K., Lian, D., & Liu, Z. (2024,). *BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation*.

DeepSeek-AI. (2025,). *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. https://arxiv.org/abs/2501.12948

Elicit: the AI Research Assistant. (2025,). https://elicit.com/

Es, S., James, J., Espinosa-Anke, L., & Schockaert, S. (2023,). *RAGAS: Automated Evaluation of Retrieval Augmented Generation*. https://arxiv.org/abs/2309.15217

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., & Wang, H. (2024,). *Retrieval-Augmented Generation for Large Language Models: A Survey*. https://arxiv.org/abs/2312.10997

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Gradio. (2025,). https://github.com/gradio-app/gradio

Gugger, S., & Howard, J. (2020). *Deep learning for coders with fastai and PyTorch*. O'Reilly Media.

Gómez-Rodríguez, C. (2025,). *Grandes modelos de lenguaje: de la predicción de palabras a la comprensión?*. https://arxiv.org/abs/2502.18205

Han, H., Wang, Y., Shomer, H., Guo, K., Ding, J., Lei, Y., Halappanavar, M., Rossi, R. A., Mukherjee, S., Tang, X., He, Q., Hua, Z., Long, B., Zhao, T., Shah, N., Javari, A., Xia, Y., & Tang, J. (2025,). *Retrieval-Augmented Generation with Graphs (GraphRAG)*. https://arxiv.org/abs/2501.00309

Humata. (2025,). https://app.humata.ai/

Johnson, R., Gamma, E., Vlissides, J., & Helm, R. (2005). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. https://books.google.com/books?id=iyIvGGp2550C

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., & Stoica, I. (2023,). Efficient Memory Management for Large Language Model Serving with PagedAttention. *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2021,). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. https://arxiv.org/abs/2005.11401

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013,). *Efficient Estimation of Word Representations in Vector Space*. https://arxiv.org/abs/1301.3781

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., … Zoph, B. (2024,). *GPT-4 Technical Report*. https://arxiv.org/abs/2303.08774

Pinecone. (2025, mayo 15). *Rerankers and Two-Stage Retrieval*.

Pressman, R. S., & Maxim, B. R. (2019). *Software Engineering: A Practitioner's Approach* (9th ed.). McGraw-Hill Education.

Richards, M., & Ford, N. (2020). *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media, Incorporated. https://books.google.com/books?id=_pNdwgEACAAJ

Rossum, G. van, & Jr., F. L. D. (2011). *The Python Language Reference Manual: for Python version 3.2*. Network Theory Ltd.

Scholarcy. (2025,). https://www.scholarcy.com/

Sobernig, S., & Zdun, U. (2010,). Inversion-of-control layer. *Proceedings of the 15th European Conference on Pattern Languages of Programs*. https://doi.org/10.1145/2328909.2328935

Sommerville, I., & Galipienso, M. (2005). *Ingeniería del software*. Pearson Educación. https://books.google.com/books?id=gQWd49zSut4C

Taylor, J. T., & Taylor, W. T. (2021). Finite State Machines. En *Patterns in the Machine: A Software Engineering Guide to Embedded Development* (pp. 137-152). Apress. https://doi.org/10.1007/978-1-4842-6440-9_10

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023,). *Attention Is All You Need*. https://arxiv.org/abs/1706.03762

Ven, G. M. van de, Soures, N., & Kudithipudi, D. (2024,). *Continual Learning and Catastrophic Forgetting*. https://arxiv.org/abs/2403.05175

Washizaki, H. (ed.). (2024). *Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0*. IEEE Computer Society. https://www.swebok.org/

Yu, H., Gan, A., Zhang, K., Tong, S., Liu, Q., & Liu, Z. (2025). Evaluation of Retrieval-Augmented Generation: A Survey. En *Big Data* (pp. 102-120). Springer Nature Singapore. https://doi.org/10.1007/978-981-96-1024-2_8

Appendix

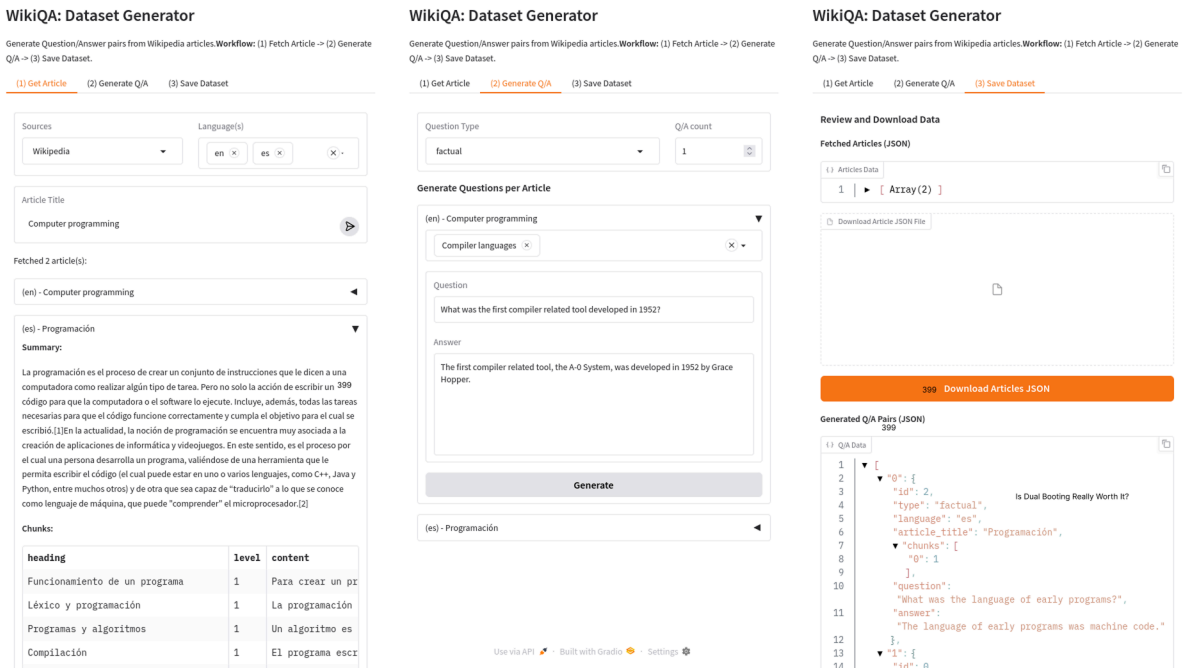


Figura 9: Herramienta de Evaluación (Elaboración propia)

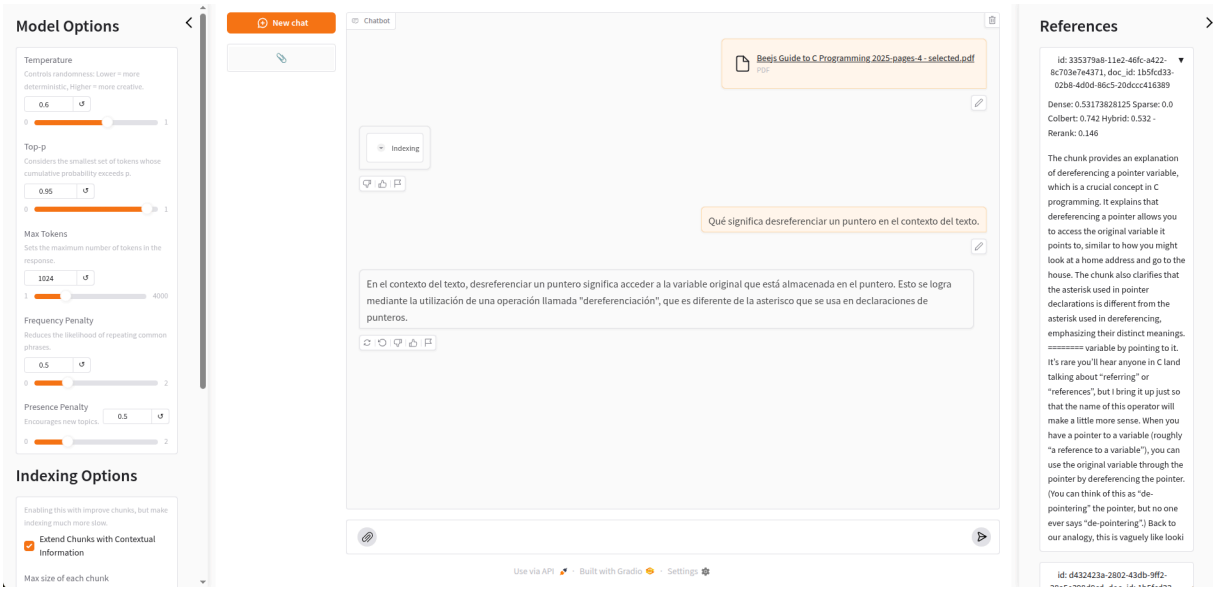


Figura 10: Prototipo (Elaboración propia)