

**Facultad de Ciencias Naturales y Exactas**

**Departamento de Ciencia de la  
Computación**

**TRABAJO DE DIPLOMA EN  
OPCIÓN AL TÍTULO DE  
LICENCIADO EN CIENCIA DE LA  
COMPUTACIÓN**

**Título:** Arquitectura Agentic RAG para consulta  
de textos pertenecientes a la Sala de Fondos  
Raros y Valiosos de la Universidad de Oriente

**Autor:** David Torres Rodríguez

**Tutores:** Dr. C. Miriela Escobedo Nicot

Dr. C. Wilkie Delgado Font

**Curso 2024-2025**

## **AGRADECIMIENTOS**

A mis familiares y amigos, siempre prestos a ayudar en todo momento.

A mis profesores, por su constante labor de educarme y hacerme un hombre de bien.

A todas aquellas personas que de disimiles formas han contribuido a la realización de este trabajo, muchas gracias.

## RESUMEN

Se aborda la problemática del acceso y consulta intuitiva del acervo documental patrimonial de la Universidad de Oriente, específicamente los libros de la Sala de Fondos Raros y Valiosos. El objetivo principal es diseñar e implementar un sistema de *chatbot* autónomo, basado en arquitecturas *Agentic RAG*, que aproveche la digitalización mediante OCR especializado de este fondo para ofrecer un acceso intuitivo, contextualizado y seguro al patrimonio, garantizando la soberanía tecnológica con software de código abierto y servidores locales. El proyecto se sustenta en los avances recientes en Recuperación Aumentada por Generación (RAG), que permiten combinar la comprensión semántica de modelos de lenguaje con mecanismos eficientes de búsqueda de información. La elección de arquitecturas *Agentic RAG* responde a la necesidad de agentes que razonen de forma autónoma, interpreten contextos complejos y gestionen interacciones dinámicas con los usuarios. Esto resulta especialmente pertinente en colecciones históricas, donde la variedad lingüística y el contexto cultural exigen adaptabilidad y precisión. Para alcanzar el objetivo propuesto se realiza el procesamiento digital de los libros, la integración de un modelo de lenguaje de código abierto local, el despliegue del flujo de agentes, la implementación de una base de datos vectorial y el desarrollo de herramientas con interfaz web. En esencia, la investigación propone una solución innovadora que combina la digitalización de documentos de gran valor patrimonial con Inteligencia Artificial avanzada para mejorar la accesibilidad, privacidad y gestión del conocimiento universitario.

## **ABSTRACT**

The project aims to address the issue of providing intuitive access to and consultation of the University of Oriente's heritage document collection, with a focus on the books in the Rare Texts Collection. The main objective is to design and implement an autonomous chatbot system based on Agentic RAG architectures that utilizes the digitization of this collection via specialized OCR to provide intuitive, contextualized and secure access to the heritage. This approach guarantees technological sovereignty through the use of open-source software and local servers. The project is based on recent advances in Augmented Retrieval by Generation (RAG), which combine the semantic understanding of language models with efficient information retrieval. The selection of Agentic RAG architectures is driven by the requirement for agents that can reason independently, interpret intricate contexts, and facilitate dynamic interactions with users. This is particularly relevant in historical collections, where linguistic variety and cultural context necessitate adaptability and precision. To achieve this, the books will be digitally processed, a local open-source language model will be integrated, the agent flow will be deployed, a vector database will be implemented and web interface tools will be developed. Essentially, the research proposes an innovative solution that combines the digitization of valuable heritage documents with advanced artificial intelligence to enhance accessibility, privacy and university knowledge management.

<b>INTRODUCCIÓN .....</b>	<b>6</b>
<b>CAPÍTULO I: FUNDAMENTOS TEÓRICOS .....</b>	<b>11</b>
1.1 Digitalización de libros pertenecientes al Fondo de Textos Raros de la Universidad de Oriente.....	11
1.2 Modelos de lenguaje de gran tamaño (LLM).....	12
1.3 Generación aumentada por recuperación.....	14
1.4 Agentic RAG .....	19
Conclusiones parciales .....	23
<b>CAPÍTULO 2: DISEÑO DEL SISTEMA .....</b>	<b>25</b>
2.1 Capa de procesamiento.....	25
2.2 Capa de agente inteligente .....	30
Configuración del agente con memoria.....	31
2.3 Capa de memoria .....	32
2.4 Capa de interfaz.....	34
Conclusiones parciales .....	36
<b>CAPÍTULO 3: RESULTADOS EXPERIMENTALES .....</b>	<b>37</b>
3.1 Resultados herramienta .....	37
3.2 Resultados del agente.....	40
Conclusiones parciales .....	44
<b>CONCLUSIONES GENERALES .....</b>	<b>45</b>
<b>RECOMENDACIONES .....</b>	<b>46</b>
<b>REFERENCIAS BIBLIOGRÁFICAS.....</b>	<b>47</b>

## INTRODUCCIÓN

Cuba posee un rico pasado histórico que, en el ámbito de las bibliotecas, se convierte en una caudal de materiales que contienen información valiosa, registrada en diversos soportes, provenientes generalmente de la cultura escrita, desde el siglo XV hasta nuestra época. Las bibliotecas son una puerta de acceso a la información y al conocimiento. Un espacio social relevante para la vida académica e investigativa. Sin embargo, a pesar del avance que ofrece el desarrollo tecnológico, poco se hace para responder a las necesidades crecientes en la gestión, manejo y conservación del patrimonio bibliográfico y documental que en ellas se conservan.

Las bibliotecas universitarias tienen una gran importancia en el entorno académico, autores como Guitián (2009) y Chiong (2012), sustentan que la biblioteca constituye uno de los núcleos más importantes de la universidad, congregan a profesores, investigadores y alumnos involucrados en el proceso de transformar la información en conocimiento. Asimismo, implementan servicios de información para atender y satisfacer las necesidades de la comunidad universitaria. Por tanto, la biblioteca universitaria de este siglo hay que pensarla y conformarla como una unidad generadora de información. La biblioteca de hoy tiene la obligación de trabajar afín de la complejidad de su naturaleza, ajustándose a las nuevas prácticas de investigación y aprendizaje, para servir como puntal a los procesos docentes e investigativos de trabajo de los usuarios, bajo el entorno digital, que está basado en las múltiples posibilidades de colaboración que ofrecen las TIC.

La Biblioteca “Francisco Martínez Anaya” de la Universidad de Oriente, atesora un acervo documental variado, que empíricamente es considerado Patrimonio Documental y Bibliográfico. Este acervo, se enfrenta a dos problemas: rescate y conservación. Siendo tangible el deterioro progresivo de los documentos tradicionales como digitales, por la desintegración natural de los soportes y por el inadecuado almacenamiento y mala manipulación de estos. Teniendo en cuenta estas consideraciones y a partir de la consulta de investigaciones realizadas, resulta impostergable la gestión de la información que contienen los documentos y libros que se encuentran en Sala de Fondos Raros y Valiosos, pues estos son considerados activos trascendentales para la Memoria

Institucional, siendo de interés común la salvaguarda y procesamiento adecuadamente de estos, aplicando de forma conveniente los conocimientos y herramientas que brinda varias ciencias.

En la actualidad, la UO promueve e intenciona el acceso y la difusión de documentos y bibliografía con valor patrimonial, como recursos de información que forman parte del Patrimonio de la Universidad, colaborando con los procesos de creación, generación y consolidación del conocimiento. La puesta en valor del Fondo Documental y Bibliográfico Institucional, resulta una prioridad dentro de los objetivos estratégicos concebidos, para dar cumplimiento a ello se inicia el proceso de digitalización de los documentos y libros más antiguos que se guardan en la Sala de Fondo Raros de la Biblioteca Central, cuya fecha extrema data de inicios del pasado siglo XX.

Para la socialización, visibilidad y acceso a esta información resulta factible el uso de las Tecnologías de la Información y las Comunicaciones (TIC) en el sentido de repositorios digitales, bibliotecas virtuales, sistemas de gestión documental (ejemplo AtoM, Omeka), metadatos estandarizados (Dublin Core), y plataformas de acceso abierto con herramientas de búsqueda semántica, complementadas con estrategias de difusión en redes sociales, exposiciones virtuales interactivas y APIs para integración con sistemas educativos. Estas tecnologías no solo democratizan el conocimiento preservado, sino que garantizan su preservación digital, interoperabilidad y alcance global, transformando patrimonio histórico en recurso vivo accesible 24/7 desde cualquier dispositivo. Cada esfuerzo o trabajo realizado para la preservación documental es un regalo para las presentes y futuras generaciones.

Entre las nuevas tecnologías para el manejo y búsqueda de información se encuentran los LLM (del inglés *Large Language Model*), una aplicación de la inteligencia artificial (IA) que permiten con lenguaje natural, a través de conversaciones, hacer búsquedas de información, incluso esta información se puede obtener con formatos específicos y referencias de su origen. Entre estas tecnologías se encuentran las *open source*, de las cuales muchas son gratis y libres de terceros, lo que permite mantener los datos privados y seguros en servidores locales y adaptarlos a necesidades específicas con

arquitecturas como los *Agentic RAG* (del inglés *Retrieval-Augmented Generation*), utilizados para el desarrollo de sistemas *chatbot* autónomos y localizados.

Un *Agentic RAG* es una evolución del modelo RAG tradicional, que incorpora agentes autónomos de IA para gestionar dinámicamente el proceso de recuperación y generación de información. A diferencia del RAG estático, un sistema *Agentic RAG* utiliza agentes inteligentes que toman decisiones, iteran y optimizan las fases clave del flujo de trabajo.

En este contexto sería interesante desarrollar un sistema de *chatbot* autónomo y localizado que permita a la comunidad intra y extra universitaria acceder a la información de los libros de la Sala de Fondos Raros y Valiosos. Con este enfoque no solo se mejoraría la accesibilidad y la gestión del conocimiento universitario, sino que también se asegura la privacidad y la soberanía tecnológica de la institución. En primer lugar, la digitalización de estos libros con tecnologías de captura y procesamiento OCR, facilitaría no solo su preservación física, sino también realizar búsquedas semánticas, visualización interactiva y conexión con sistemas de IA como *Agentic RAG* para *chatbots* de consulta contextualizada.

Considerando el panorama descrito anteriormente, el **problema de investigación** del presente trabajo es la carencia de un sistema autónomo, para acceder a los libros del Fondo de Textos Raros, que garantice accesibilidad eficiente y autonomía institucional, limitando la divulgación y preservación de fuentes históricas clave en la Universidad de Oriente (UO)."

En consecuencia, se define el desarrollo de un *chatbot* autónomo basado en *Agentic RAG* para consultar libros del Fondo de Textos Raros digitalizados, garantizando accesibilidad y soberanía tecnológica en la UO como **objeto de investigación** y como **campo de acción** la aplicación de tecnologías de inteligencia artificial (LLMs y *Agentic RAG*) en sistemas de gestión archivística universitaria para la consulta del patrimonio documental.

En este sentido, este trabajo tiene como **objetivo general**:



- Diseñar e implementar un sistema de *chatbot* autónomo basado en arquitecturas *Agentic RAG*, que aproveche la digitalización (OCR especializado) de la Sala de Fondos Raros y Valiosos de la Universidad de Oriente, para generalizar el acceso intuitivo, contextualizado y seguro a este patrimonio documental, garantizando soberanía tecnológica mediante software *open source* y servidores locales.

Para alcanzar ese objetivo general se han diseñado los siguientes **objetivos específicos**:

1. Procesar con OCR los libros del Fondo de Textos Raros digitalizados previamente.
2. Integrar un modelo de lenguaje de código abierto para su ejecución local.
3. Configurar y desplegar el flujo de agentes para gestionar las interacciones y tareas del *chatbot*.
4. Implementar una base de datos vectorial para almacenar y gestionar la información relacionada con los libros de la Sala de Fondos Raros y Valiosos.
5. Desarrollo de herramientas a ser utilizadas por el agente, con una interfaz web que permita a los usuarios interactuar con el *chatbot*.
6. Realizar pruebas de rendimiento y optimización del sistema para asegurar su eficiencia y robustez.

Como hipótesis se tiene que la implementación de un sistema de *chatbot* autónomo basado en arquitecturas *Agentic RAG*, utilizando modelos de lenguaje locales y digitalización OCR, mejora significativamente la accesibilidad y la gestión del patrimonio documental de la Sala de Fondos Raros y Valiosos de la Universidad de Oriente.

## **Estructura del texto**

El informe incluye un resumen en español e inglés que explica brevemente el objetivo y los principales resultados de la investigación. En la introducción, se presentan los elementos del diseño metodológico, incluyendo la justificación del estudio, el problema de investigación y los objetivos generales y específicos, así como la hipótesis. El desarrollo se organiza en tres capítulos. El primer capítulo abarca el marco teórico, se discuten los fundamentos de los LLM, RAG y *Agentic RAG*. El segundo capítulo detalla la implementación de la propuesta. Mientras que el tercer capítulo presenta los resultados y su evaluación. Finalmente, se incluyen las conclusiones generales y las referencias bibliográficas.

# **CAPÍTULO I: FUNDAMENTOS TEÓRICOS**

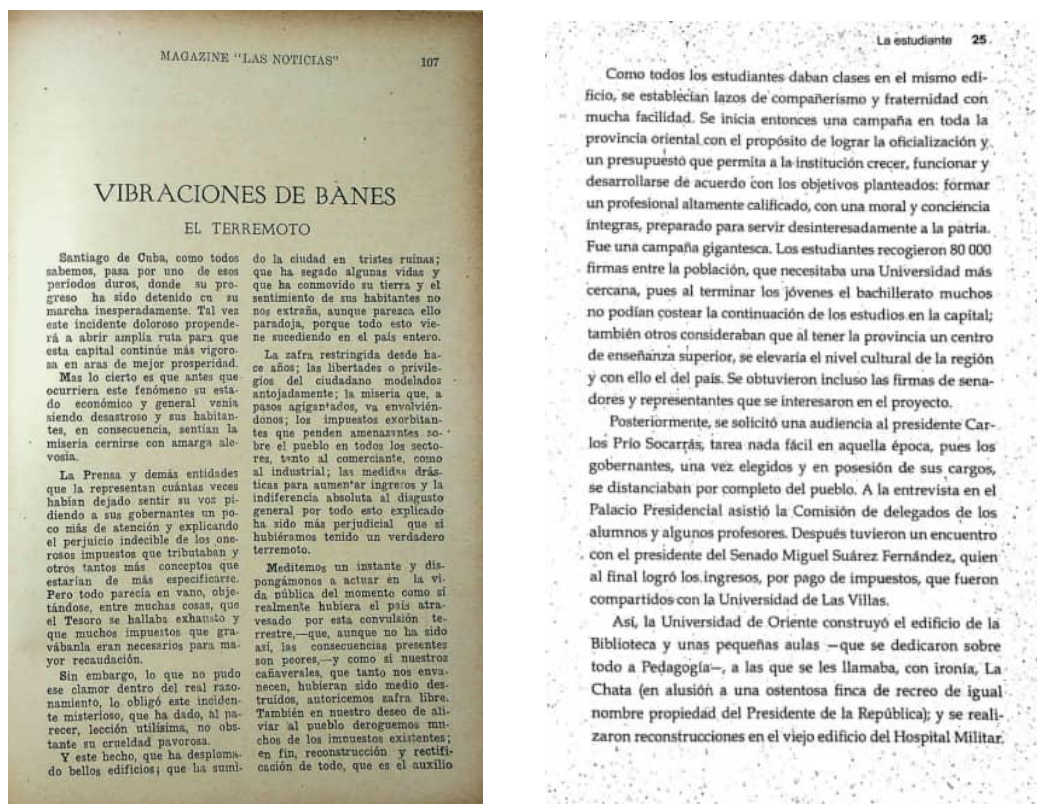
En este capítulo se presenta la fundamentación teórica de la investigación. Se describen los principios teóricos que sustentan el desarrollo de la investigación entre los que se pueden mencionar los LLM,

## **1.1 Digitalización de libros pertenecientes al Fondo de Textos Raros de la Universidad de Oriente**

La puesta en valor del acervo documental con valor patrimonial, en estos momentos resulta una prioridad dado que entre esta documentación se encuentran los libros de la Sala de Fondos Raros y Valiosos de la biblioteca Francisco Martínez Anaya de la UO, que ostentan valor excepcional. La información en ellos contenida (incluyendo primeras ediciones y tratados científicos únicos) permite reconstruir la memoria intelectual de la universidad y su contribución al desarrollo del oriente cubano.

Actualmente se está llevando a cabo un proceso de digitalización de estos libros con tecnologías de captura (Figura 1.1), que permite garantizar su preservación y facilitar el acceso a investigadores, estudiantes y al público en general sin riesgo de deterioro para los originales. Este proyecto utiliza equipos de para capturar imágenes fieles a los documentos, respetando su integridad y detalles históricos, todo lo cual facilitará estudios filológicos, históricos y científicos sobre estos materiales.

La digitalización no solo protege los textos raros y valiosos de posibles daños por manipulación constante o factores ambientales, sino que también democratiza el conocimiento, al hacer posible la consulta remota desde cualquier parte del mundo. Asimismo, se prevé la creación de un repositorio digital con herramientas de análisis interactivo. Con esta iniciativa, la Universidad de Oriente (UO) reafirma su compromiso con la salvaguarda de su legado cultural y la divulgación del saber atesorado en sus fondos documentales, asegurando que las futuras generaciones puedan conocer y valorar la riqueza académica e histórica que alberga su archivo.



**Figura 1.1** Ejemplo de dos páginas de libros de la Sala de Fondos Raros y Valiosos de la Universidad de Oriente.

Este proceso responde a una política de país, a la cual el Ministerio de Educación Superior le ha dado tratamiento. La investigación válida además la inter y transdisciplinariedad, favoreciendo no solo a la comunidad académica sino también a la comunidad extramuros, garantizando además las buenas prácticas en materia de gestión, uso y manejo de la información.

## 1.2 Modelos de lenguaje de gran tamaño (LLM)

Los modelos de lenguaje de gran tamaño (LLM, por sus siglas en inglés) (Chang et. al, 2024) son sistemas de aprendizaje profundo encargados del procesamiento del lenguaje natural que buscan comprender, analizar y generar texto. La arquitectura basada en *Transformers* (un conjunto de redes neuronales que consta de un codificador y un decodificador con capacidades de auto atención) y el volumen masivo de datos con los que son entrenados (cantidades masivas de datos textuales procesados para la

obtención de respuestas precisas) realizan tareas lingüísticas que van desde la traducción automática hasta la generación de contenido creativo (Myer, M., 2023).

La estructura permite que el modelo extraiga significados de secuencias de texto y comprenda las relaciones entre palabras y frases contenidas en ellas, gracias a su capacidad de procesar secuencias enteras en paralelo, a diferencia de las redes neuronales recurrentes (RNN) anteriores que procesaban las entradas de forma secuencial. Este enfoque basado en mecanismos de atención, como el *Transformer*, mejora la eficiencia y el rendimiento del modelo al analizar dependencias a largo plazo y asignar pesos diferenciados a cada elemento de la secuencia, permitiendo una comprensión más contextualizada del lenguaje. Además, al evitar el procesamiento paso a paso típico de las RNN, se eliminan problemas como el desvanecimiento del gradiente y se acelera significativamente el entrenamiento, lo que facilita el manejo de grandes volúmenes de datos textuales con mayor precisión.

La escala masiva de su arquitectura, frecuentemente con cientos de miles de millones de parámetros, funciona como una "memoria" o banco de conocimientos que se adquiere durante su entrenamiento (Myer, M., 2023), permitiendo incorporar cantidades enormes de datos procesados, derivando en un modelo de "gran tamaño", versatilidad y capacidad para adaptarse a múltiples tareas y contextos, lo que ha permitido su implementación en diversos campos y aplicaciones.

En el ámbito del procesamiento de información, los LLM se utilizan para resumir y extraer datos relevantes de documentos extensos mediante procesos de identificación, categorización, selección y extracción de información significativa de grandes volúmenes contenido.

## **LLAMA**

El modelo LLAMA (*Large Language Model Meta AI*) de Meta es una familia de modelos de lenguaje de IA de código abierto, conocidos por su eficiencia y rendimiento en diversas tareas de procesamiento de lenguaje natural (NLP). Entre sus múltiples ventajas están:

1. Arquitectura Avanzada: Basado en la arquitectura *Transformer*, similar a GPT, pero optimizado para un mejor rendimiento con menos recursos.
2. Variantes Escalables: Disponible en diferentes tamaños (LLAMA 1 y 2, con versiones de 7B, 13B, 65B y 70B de parámetros).
3. Open-Source: Meta liberó los pesos del modelo para investigación y uso comercial (bajo licencia específica), fomentando la innovación colaborativa.
4. Eficiencia: Diseñado para requerir menos potencia computacional en comparación con otros modelos de su categoría (como GPT-3).
5. Capacidades Multilingüe: Buen desempeño en varios idiomas, aunque principalmente optimizado para inglés.
6. *Fine-Tuning* y Adaptabilidad: Permite ajustes para aplicaciones específicas, como *chatbots* (LLAMA 2-CHAT).
7. Seguridad y Control: Incorpora mitigaciones para reducir sesgos y respuestas inapropiadas, aunque con limitaciones.
8. Sin fines de lucro: Meta lo desarrolló con un enfoque más accesible que modelos cerrados como los de OpenAI o Google.

### **1.3 Generación aumentada por recuperación**

La generación aumentada por recuperación (RAG, por sus siglas en inglés) (Zhaom et.al.; 2024) es un marco de trabajo en IA que integra sistemas de recuperación de información con modelos de lenguaje grandes (LLM) para mejorar la precisión, actualidad y relevancia contextual de las respuestas generadas. A diferencia de los LLM convencionales, que dependen exclusivamente de datos pre-entrenados, el RAG tradicional permite acceder dinámicamente a fuentes externas de conocimiento, lo que reduce las alucinaciones y garantiza respuestas basadas en información verificable. (Huang, Yu, Ma. et.. al; 2025). A continuación, se describen los componentes principales de un RAG tradicional

#### **1. Embeddings**

Los *embeddings* son representaciones vectoriales que capturan el significado semántico de textos, imágenes u otros datos. En RAG, estos vectores se generan mediante

modelos de *embedding* especializados (como BERT o GPT) que transforman palabras, frases o documentos completos en puntos dentro de un espacio multidimensional<sup>1</sup>..

El modelo `mxbai-embed-large:335m` es un modelo de *embeddings* de texto de alta calidad desarrollado por Mixed Bread AI. Está diseñado para generar representaciones vectoriales densas (*embeddings*) de textos, útiles para tareas como búsqueda semántica, *clustering*, clasificación de texto y más. Funcionaría de la siguiente forma: - durante la fase de indexación, los documentos se dividen en fragmentos (*chunks*) y se convierten en *embeddings*. En la recuperación, la consulta del usuario también se vectoriza, permitiendo buscar similitudes con los *embeddings* almacenados. La comparación matemática (mediante métricas como la similitud del coseno) identifica los fragmentos más relevantes para la consulta.

Algunas de las ventajas de este modelo son:

- *Open source* y gratuito: A diferencia de modelos cerrados como los de OpenAI (`text-embedding-ada-002`), `mxbai-embed-large:335m` es totalmente accesible.
- Alto rendimiento en tareas semántica: Logra un rendimiento competitivo en *benchmarks* como MTEB (Massive Text Embedding Benchmark), superando a muchos modelos de tamaño similar. Optimizado para búsqueda semántica, recuperación de información y similitud de textos.
- Eficiencia en velocidad y tamaño: Con 335 millones de parámetros, ofrece un equilibrio entre precisión y eficiencia computacional. Más rápido que modelos grandes como OpenAI `text-embedding-3-large`, pero con resultados cercanos en calidad.
- Soporte para contexto largo (hasta 512 tokens): Maneja textos relativamente largos sin perder calidad en la representación. Ideal para *chunks* de documentos en aplicaciones RAG (Retrieval-Augmented Generation).

---

<sup>1</sup> Esta información fue extraída del sitio web de Microsoft <https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/rag/rag-generate-embeddings>

- Multilingüe optimizado: Funciona bien en múltiples idiomas, aunque está especialmente optimizado para inglés. Supera a alternativas como e5-large en tareas multilingües.
- Fácil integración con otras herramientas: Compatible con librerías como Sentence-Transformers, Hugging Face Transformers y APIs como Pinecone, Weaviate o Milvus.
- Optimizado para RAG y aplicaciones de IA generativa: Proporciona *embeddings* densos que mejoran la recuperación de información en sistemas de preguntas y respuestas. Funciona bien en combinación con LLMs como Llama 3, GPT-4 o Mistral.
- Balance entre costo y rendimiento: Una alternativa rentable frente a modelos más caros de OpenAI o Cohere. Ideal para despliegues en entornos con recursos limitados.

Es una excelente opción para aplicaciones que requieren *embeddings* de alta calidad sin la sobrecarga computacional de modelos gigantes. Su equilibrio entre precisión, velocidad y soporte multilingüe lo hace ideal para proyectos de IA y búsqueda semántica.

## 2. Base de Datos Vectorial

Las bases de datos vectoriales (Han, Liu, and Wang., 2023) son sistemas optimizados para almacenar *embeddings* y realizar búsquedas de similitud a gran escala. En el contexto de los sistemas RAG, estas bases de datos almacenan representaciones vectoriales (*embeddings*) de documentos o fragmentos de conocimiento, generados mediante modelos, que capturan el significado semántico del texto. Cuando un usuario realiza una consulta, el sistema transforma la pregunta en un *embedding* y busca en la base de datos los fragmentos más similares utilizando algoritmos eficientes de vecinos más cercanos (k-NN o ANN). La información recuperada se proporciona luego al modelo generativo (como GPT) como contexto adicional, permitiendo que las respuestas sean más precisas, actualizadas y fundamentadas en fuentes externas. De esta manera, las bases de datos vectoriales puentean la limitación de memoria de los LLMs, evitando



alucinaciones y mejorando la calidad de las respuestas en dominios específicos, desde soporte técnico hasta investigación académica.

Entre las principales arquitecturas y métodos utilizados en este contexto destacan:

- **FAISS (Facebook AI Similarity Search)** – Desarrollado por Meta, permite búsquedas eficientes en espacios vectoriales mediante indexación avanzada y optimización para GPU.
- **Annoy (Approximate Nearest Neighbors Oh Yeah)** – Creado por Spotify, usa árboles aleatorios proyectados para búsquedas aproximadas con bajo consumo de memoria.
- **HNSW (Hierarchical Navigable Small World)** – Una estructura basada en grafos multicapa que logra alta precisión y velocidad en recuperación de vecinos cercanos.
- **Milvus** – Sistema de base de datos vectorial de código abierto que integra múltiples algoritmos ANN, incluyendo IVF (Inverted File Index) y HNSW.
- **Pinecone** – Plataforma gestionada especializada en búsqueda semántica, optimizada para aplicaciones en producción con escalabilidad automática.
- **Weaviate** – Base de datos vectorial con capacidades de búsqueda híbrida (texto + vectores) y soporte para modelos de lenguaje como BERT y GPT.
- **QDrant** – Motor de búsqueda vectorial escrito en Rust, diseñado para alto rendimiento y fácil despliegue en entornos cloud.

Estas arquitecturas son fundamentales en aplicaciones como motores de recomendación, búsqueda semántica, detección de similitud de documentos y sistemas de recuperación de información basados en IA. Su eficiencia radica en equilibrar precisión y velocidad, permitiendo operaciones en tiempo real sobre grandes volúmenes de datos vectoriales.

### 3. Memoria

En el contexto de RAG, la memoria (Zhang et.al; 2024.) (Zhang et.al; 2024) se refiere a la capacidad del sistema para retener y utilizar información contextual durante interacciones. Esto incluye:

- **Memoria a corto plazo:**

- a) Mantiene el contexto de la conversación actual (por ejemplo, el historial de mensajes en un *chatbot*) para generar respuestas coherentes.
- b) Se implementa mediante buffers o ventanas deslizantes que limitan la cantidad de tokens procesados.

- **Memoria a largo plazo:**

- a) Corresponde a la base de datos vectorial, que actúa como repositorio de conocimiento actualizable.
- b) Permite al sistema acceder a información nueva sin reentrenar el modelo, simplemente actualizando los fragmentos indexados.

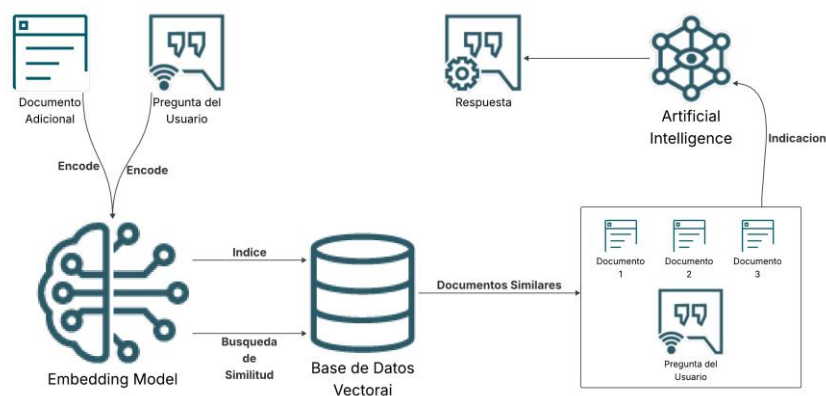
Los fragmentos recuperados y el historial de conversación se inyectan en el *prompt* (Aditi S., Abul E., Gaurav K., et.al.; 2024.) del LLM, guiando la generación de respuestas. Técnicas como el *chunking* (división de documentos en partes manejables) aseguran que la información relevante no se pierda por limitaciones de longitud de contexto.

#### ***Flujo de trabajo de un sistema RAG tradicional***

El flujo de trabajo de un sistema RAG tradicional es el siguiente, Figura 1.1:

1. **Indexación de Documentos:** Los documentos se dividen en fragmentos, se enriquecen con metadatos y se convierten en *embeddings*; estos y metadatos se almacenan en la base de datos vectorial
2. **Consulta del Usuario:** La consulta se vectoriza utilizando el mismo modelo de *embedding*. La base de datos vectorial devuelve los fragmentos más relevantes basados en similitud semántica.

3. **Generación de Respuesta:** El LLM recibe el *prompt* (Aditi S., Abul E., Gaurav K., et.al.; 2024.) con la consulta, los fragmentos recuperados y el historial de conversación. Genera una respuesta contextualizada, citando información verificable de las fuentes recuperadas.
4. **Actualización y Aprendizaje:** Nuevos documentos pueden añadirse a la base de datos vectorial sin modificar el LLM. Sistemas avanzados incorporan retroalimentación para ajustar la recuperación o generación en futuras interacciones.



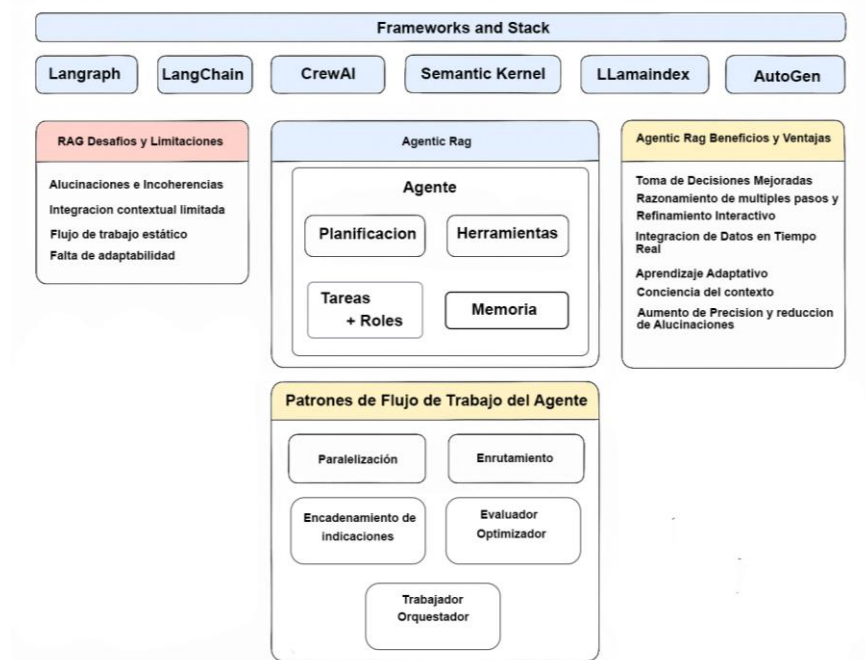
**Figura 1.1:** Flujo de trabajo de un sistema RAG tradicional.

## 1.4 Agentic RAG

*Agentic RAG (Retrieval-Augmented Generation con capacidades agénticas)* (Chidaksh R., Sagar S. S., and Venkataramana R. 2024.) es un paradigma emergente de IA donde los LLMs planifican autónomamente sus próximos pasos mientras extraen información de fuentes externas. A diferencia del RAG tradicional, que sigue un patrón estático de recuperación-generación, el *Agentic RAG* implementa un ciclo iterativo de interacciones entre el LLM y diversas herramientas, permitiendo un proceso de razonamiento dinámico (Huang and Chang, 2023).

Mientras que el RAG tradicional se enfoca principalmente en la fundamentación factual mediante la recuperación de información relevante, el *Agentic RAG* incorpora capacidades de planificación (Huang, Liu, et. al, 2024.) y adaptabilidad dentro de entornos complejos. Esta diferencia es crucial: el RAG tradicional ejecuta un flujo

predefinido, mientras que *Agentic RAG* posee su proceso de razonamiento, tomando decisiones autónomas sobre cómo abordar un problema basándose en la calidad de la información que encuentra, por tanto, se puede considerar que el *Agentic RAG* es una evolución del RAG permitiendo mayor adaptabilidad y manejo de procesos más complejos, Figura 1.2.



**Figura 1.2:** Resumen general del funcionamiento de un *Agentic RAG*.

El *Agentic RAG* presenta una estructura basada en tres componentes principales: agente, memoria a largo plazo y herramientas. Los mismos son descritos a continuación:

**Agentes Autónomos:** Este componente constituye el núcleo diferenciador de este sistema, otorgándole capacidades para:

- Planificar secuencias de acciones basadas en el objetivo del usuario
- Descomponer tareas complejas en subtareas manejables
- Decidir dinámicamente qué fuentes consultar (libros académicos, normativas universitarias o FAQs)

- Evaluar la calidad de la información obtenida y determinar pasos adicionales (Gou et.al, 2024.)
- Reescribir consultas fallidas o reformular estrategias de búsqueda

La capacidad distintiva que hace "agente" a un sistema es precisamente su autonomía para decidir la secuencia de pasos sin depender de un guion predefinido por humanos (Xu H., Weiwen L., Xiaolong C.,et.al.; 2024.)

**Memoria a Largo Plazo:** Los sistemas *Agentic RAG* mantienen estado y memoria a través de las distintas etapas de interacción, permitiéndoles:

- Recordar intentos anteriores y sus resultados
- Evitar bucles repetitivos de búsqueda
- Tomar decisiones más informadas conforme avanza la conversación
- Almacenar contexto específico por usuario para personalizar respuestas futuras<sup>[1]</sup>

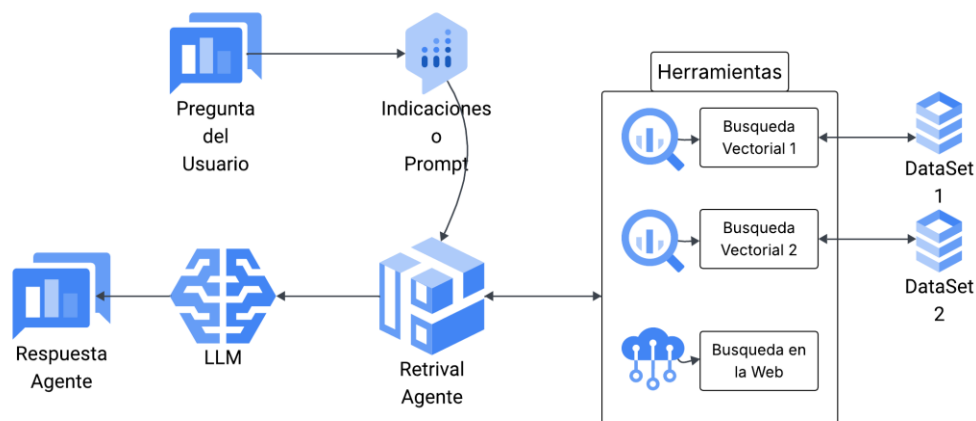
**Herramientas:** El sistema incorpora múltiples herramientas que pueden ser invocadas según sea necesario:

- Bases de datos vectoriales para buscar en contenido académico
- APIs para consultar sistemas institucionales
- Motores de búsqueda web para información complementaria
- Consultas a bases de datos estructuradas para información administrativa

Un flujo de trabajo *Agentic RAG* permite que agentes de IA gestionen tareas complejas de forma autónoma, accediendo dinámicamente a información actualizada, razonando y actuando de manera proactiva para resolver problemas del mundo real

Se pueden definir y ejemplificar las etapas principales del flujo de trabajo para la ejecución de un *Agentic RAG* de la siguiente forma (Aditi S, Abul E, Saket K, et.al, 2024.):

- **Comprensión del objetivo y contexto:** El agente de IA recibe una tarea o consulta, analiza el contexto y define de manera autónoma los pasos necesarios para cumplir el objetivo, considerando reglas, restricciones y la información disponible
- **Recuperación dinámica de información:** El agente identifica qué información necesita y la recupera en tiempo real desde múltiples fuentes externas (API, bases de datos, documentos, etc.), en lugar de depender solo de datos pre-entrenados o una única base de conocimiento
- **Generación aumentada y razonamiento multipaso:** El agente integra la información recuperada con su conocimiento interno, razona sobre los datos y elabora respuestas o soluciones contextualmente relevantes. Puede realizar razonamiento de varios pasos, ajustando dinámicamente el flujo de trabajo según los resultados intermedios
- **Toma de decisiones autónoma y ejecución:** Basado en el análisis y razonamiento, el agente decide las acciones a tomar, ejecuta tareas (como llamar a APIs, actualizar sistemas, enviar notificaciones) y coordina con otros agentes si es necesario
- **Aprendizaje y retroalimentación:** El sistema incorpora la retroalimentación de los resultados y del usuario para mejorar continuamente sus procesos y respuestas en futuras interacciones



**Figura 1.3:** Flujo de trabajo de un *Agentic RAG*.

## LangChain

LangChain es un *framework* de código abierto diseñado para simplificar la creación de flujos de IA basados en lenguaje, especialmente útil para sistemas RAG con capacidades *agentic* (autónomas). Su principal ventaja radica en su capacidad para integrar modelos de lenguaje (LLMs) como OpenAI, Anthropic o Llama 2 con bases de datos vectoriales (FAISS, Chroma, Pinecone), permitiendo una búsqueda semántica eficiente que enriquece las respuestas generadas.

En el contexto de los sistemas RAG, LangChain actúa como un orquestador modular, facilitando desde la carga y división de documentos (*Document Loaders* y *Text Splitters*) hasta la recuperación contextual de información (*Retrievers*) y el mantenimiento de memoria conversacional. Esto es clave para evitar alucinaciones en los LLMs, ya que el *framework* asegura que las respuestas se fundamenten en datos externos relevantes, recuperados mediante *embeddings* (como los de OpenAI o Hugging Face).

Además, sus capacidades *agentic* lo hacen ideal para aplicaciones autónomas: los agentes reactivos pueden tomar decisiones dinámicas usando herramientas (APIs, código Python o SQL), mientras que funciones de planificación automatizada (como ReAct o Self-Ask) descomponen tareas complejas en subtareas. LangChain también ofrece personalización avanzada mediante *prompt templates*, cadenas de procesos personalizadas (Custom Chains) y evaluación integrada de resultados.

Como plataforma *open source* con una comunidad activa, es ideal para desarrollar asistentes inteligentes, *chatbots* con acceso a datos en tiempo real y sistemas autónomos que requieren razonamiento multi-paso, consolidándose como una herramienta esencial para implementar IA conversacional robusta y escalable.

## Conclusiones parciales

A partir del análisis realizado en este capítulo, se concluye que la digitalización de los libros del Fondo de Textos Raros de la Universidad de Oriente constituye una acción estratégica esencial para preservar el patrimonio documental universitario y facilitar su consulta. Esta iniciativa no solo garantiza la conservación física de documentos valiosos,

sino que también permite el acceso al conocimiento, por los investigadores, estudiantes y el público general accedan a estos contenidos mediante plataformas digitales.

Por otra parte, la incorporación de LLM y arquitecturas como RAG y *Agentic RAG* representa un avance en los sistemas de recuperación y generación de información. Estas tecnologías permiten consultas más precisas, contextualizadas y dinámicas, optimizando la interacción entre los usuarios y el conocimiento almacenado. En especial, los *Agentic RAG* destacan por su capacidad para planificar, razonar y actuar de forma autónoma en entornos complejos, lo que los convierte en herramientas ideales para desarrollar *chatbots* localizados y seguros. Esta integración impulsa un nuevo paradigma de acceso a la información patrimonial, alineado con los principios de soberanía tecnológica y privacidad institucional.



## CAPÍTULO 2: DISEÑO DEL SISTEMA

El sistema propuesto está estructurado en cuatro capas principales, cada una con una función específica:

1. **Capa de Procesamiento (OCR, VECTORSTORE):** Combina modelos de visión local (Ollama) para el reconocimiento óptico de caracteres (OCR) y un almacenamiento vectorial (FAISS) para gestionar representaciones numéricas de texto.
2. **Capa de Agente Inteligente (Implementada con LangChain):** Se encarga de coordinar herramientas como modelos de lenguaje (LLM) y gestionar flujos de trabajo.
3. **Capa de Memoria:** Utiliza una base de datos relacional (SQLite) para almacenar información estructurada.
4. **Capa de Interfaz (Desarrollada con Streamlit):** Ofrece una interfaz interactiva para que los usuarios interactúen con el sistema.

La implementación de la primera capa se realiza una única vez, los resultados son almacenados y reutilizados posteriormente por el resto de las capas. Las capas de Agente Inteligente y de Memoria, se ejecutan de forma simultánea en la última capa.

A continuación, se describirán todas las fases mencionadas anteriormente.

### 2.1 Capa de procesamiento

Esta fase se centra en la digitalización y preprocesamiento de documentos, un paso clave en aplicaciones de Procesamiento de Lenguaje Natural (NLP).

Inicialmente se desarrolla la Integración de Google Cloud Vision API para Reconocimiento de Texto. En el marco del desarrollo de la aplicación, la extracción automatizada de texto a partir de imágenes constituyó un requerimiento fundamental. Para ello, se adoptó Google Cloud Vision API (v1) como solución OCR central, seleccionada por su robustez en el procesamiento de imágenes complejas, su soporte

para múltiples idiomas y su escalabilidad en entornos cloud. Este módulo operó como complemento crítico para la digitalización de documentos, permitiendo la transformación de contenidos gráficos en datos estructurados procesables por el sistema.

La implementación siguió un flujo estandarizado:

1. Preprocesamiento de imágenes:

- Las imágenes de entrada no fueron sometidas a un filtrado, por las potencialidades del framework
- Formatos admitidos: JPEG, PNG, BMP y TIFF, con resolución mínima recomendada de 300 ppp.

2. Invocación de la API:

- Mediante solicitudes HTTP/REST, se utilizó el **endpoint textDetection** (para texto disperso) y **documentTextDetection** (para documentos densos, preservando estructura semántica).
- Cada petición incluyó:
  - Credenciales de autenticación vía archivo de clave JSON.
  - Parámetros de configuración: idioma prioritario (es), y activación de corrección ortográfica (**enableTextDetectionConfidenceScore=true**).

3. Postprocesamiento:

- Los JSON de respuesta fueron parseados para extraer:
  - Cajas delimitadoras (*bounding boxes*).
  - Relaciones espaciales entre párrafos, líneas y palabras.

Esta forma de trabajo presenta ventajas relacionadas con:

- Alta precisión en fuentes impresas estándar (superior a soluciones *offline* como Tesseract en pruebas comparativas).
- Detección contextual de idiomas mixtos.
- Integración nativa con servicios GCP (*ej.: Cloud Storage para batch processing*).

Luego de procesados los documentos con el OCR, se trabaja con dos funciones principales:

### **cargar\_documentos()**

Objetivo: Cargar archivos de texto (.txt) desde un directorio específico.

Detalles:

- Usa **DirectoryLoader** para buscar archivos con extensión ``.txt`` en la ruta ``data/content/``.
- El parámetro **autodetect\_encoding** permite manejar diferentes codificaciones (UTF-8, ASCII, etc.).
- **TextLoader** indica que los archivos se cargarán como texto plano.
- Salida: Una lista de documentos cargados en memoria.

### **dividir\_textos(documentos):**

Objetivo: Fragmentar los documentos en partes más pequeñas para facilitar su análisis posterior.

Detalles:

- Emplea **RecursiveCharacterTextSplitter**, que divide el texto en fragmentos de tamaño fijo (``tamaño_fragmento=1000`` caracteres) con un solapamiento (``solapamiento_fragmento=200`` caracteres) para mantener el contexto entre segmentos.
- La función **len** calcula la longitud de cada fragmento.
- Salida: Una lista de fragmentos de texto procesados.

A continuación, se muestra un ejemplo del funcionamiento de esta última función.

Texto original (50 caracteres): "Los modelos de lenguaje grande (LLM) son útiles para NLP."

Configuración: `tamaño_fragmento=30`, `solapamiento_fragmento=10`.

Fragmentos resultantes:

1. "Los modelos de lenguaje grande (LLM)"

## 2. "(LLM) son útiles para NLP."

Los fragmentos resultantes están listos para pasos posteriores a continuación se genera el *embeddings* y hace el almacenamiento vectorial

La creación de una base de datos vectorial, es un componente clave para sistemas de búsqueda semántica o *chatbots* que trabajan con modelos de lenguaje. Transforma textos en vectores (*embeddings*) y los organiza para búsquedas eficientes. Para ello lo primero que se ejecuta es la siguiente función:

**crear\_embeddings(modelo)**

Objetivo: Generar representaciones numéricas (vectores) de textos usando un modelo de *embeddings*.

Detalles:

- Usa **OllamaEmbeddings** (un servicio hipotético de *embeddings*) para convertir texto en vectores.
- Calcula la dimensión de los *embeddings* (ejemplo 768, 1024) de un texto de prueba.
- Crea un índice **FAISS** (biblioteca de Facebook para búsqueda vectorial) usando la distancia L2 (euclidiana) para comparar vectores.
- Salida:  
*embeddings*: Función para generar vectores a partir de texto.  
índice: Estructura FAISS para búsqueda rápida.

A continuación, se muestra un ejemplo del funcionamiento de esta función.

```
# Documentos a almacenar:
```

```
texts = ["El gato come pescado", "Los perros ladran fuerte"]
```

```
# 1. Generar embeddings (simulado):
```

```
embeddings.embed_query("El gato come pescado") → [0.3, -0.2, 0.7]
```

```
embeddings.embed_query("Los perros ladran fuerte") → [0.5, 0.1, 0.6]
```

# 2. Búsqueda semántica:

```
query = "minino" # Sinónimo de "gato"
```

```
embedding_query = [0.28, -0.19, 0.72] # Similar al embedding del gato
```

# FAISS devuelve el documento más cercano:

```
vector_store.buscar(embedding_query) → "El gato come pescado"
```

Luego se configura la base de datos vectorial con la siguiente función:

```
vector_store = FAISS(  
    embedding_function=embeddings,  
    index=index,  
    docstore=InMemoryDocstore(),  
    index_to_docstore_id={},  
)  
  
vector_store.add_documents(texts)  
  
vector_store.save_local(PATH)
```

Objetivo: Preparar un almacén donde se guardarán los documentos y sus *embeddings*.

Detalles:

- Usa FAISS como motor de búsqueda vectorial.
- **InMemoryDocstore** almacena los textos originales en memoria.
- **mapeo\_indices** relaciona los vectores del índice con los documentos.
- Salida: Un **vector\_store** listo para almacenar y buscar documentos.

El flujo de trabajo de esta capa en general sería el siguiente:

- Crear *embeddings* e índice (ejemplo usando el modelo "mxbai-embed-large:335m").

- Configurar el almacén vectorial con FAISS.
- Añadir documentos fragmentados (texts) al almacén.
- Guardar el almacén en disco para reutilización (RUTA\_ALMACENAMIENTO).

## 2.2 Capa de agente inteligente

La herramienta utiliza la técnica RAG, que combina recuperación de información y generación de respuestas, con el objetivo de proporcionar respuestas precisas basadas exclusivamente en documentos de referencia.

El *Agentic RAG* razona y decide cuál de las herramientas que tiene disponible es más adecuada para la instrucción dada por el usuario. Por eso lo primero que hay que hacer es generar esas herramientas. Para este trabajo solo se genera una herramienta, pero se pueden usar otras para contextos diferentes. Aclarar que, para este caso específico, el *Agentic RAG* decide si quiere o no reutilizar la herramienta varias veces.

La función para generar la herramienta es la siguiente:

**`search_library(query)`**

Propósito: Buscar información en documentos académicos y generar respuestas verificables.

El flujo de trabajo es el siguiente:

- Configura un pipeline de QA (**`RetrievalQA`**) con reglas estrictas.
- Ejecuta la consulta contra los documentos.
- Procesa y almacena las fuentes de referencia.
- Devuelve la respuesta con sus referencias.

Luego se debe configurar la herramienta, utilizando la clase **`Tool`** la cual tiene los parámetros:

- Nombre descriptivo (**`BusquedaAcademica`**).
- Ejemplos de uso en la descripción.
- Una función *wrapper* que invoca el sistema.

Un ejemplo de herramienta RAG es el siguiente:

DEFINIR `academic_qa` COMO `RetrievalQA.from_chain_type` CON PARÁMETROS:

```
llm = self.llm

chain_type = "stuff"

retriever = self.retriever

prompt = PlantillaPrompt(
    plantilla = """
    Eres un asistente académico especializado...

    **Instrucciones estrictas:**

    1. Respuesta EXCLUSIVAMENTE basada en el contexto.
    2. Si no hay información: "No encontré información relevante..."
    3. Prohibido inventar información.
    4. Incluir referencia exacta (documento, sección/página).

    Contexto: {context}

    Pregunta: {question}

    Respuesta: """)
```

## Configuración del agente con memoria

Los componentes claves de esta capa son:

- Memoria conversacional (`ConversationBufferMemory`): Almacena el historial completo del chat en la variable `chat_history`.
- Plantilla de prompt especializado: Define la identidad del asistente y el formato de razonamiento.
- Configuración del agente:

- Usa el modelo `llama3` con baja temperatura (`0.2`) para respuestas conservadoras.
- `library_tool`: Para búsqueda en documentos (explicada anteriormente).
- Ejecutor del agente:
  - Maneja errores automáticamente (ejemplo: si el usuario hace preguntas ambiguas).
  - El modo `verbose=True` muestra el proceso interno paso a paso.

## 2.3 Capa de memoria

En esta investigación se emplea SQLite para guardar el historial de conversaciones con el *chatbot*, debido a las ventajas que tiene en cuanto a (*Junyan, Shiguo & Yijie, 2009*)

- Ligero y Autocontenido:
  - No requiere servidor externo (todo se guarda en un único archivo .db).
  - Ideal para aplicaciones simples o embebidas (ejemplo apps móviles, *bots* locales).
- Fácil Implementación
  - Soporte nativo en Python (módulo `sqlite3` incluido en la librería estándar).
  - Sintaxis SQL sencilla para crear/consultar tablas de mensajes.
- Bajo Coste de Mantenimiento:
  - No necesita configuración compleja ni administración de bases de datos.
  - Escala bien para miles de mensajes (en *chatbots* personales o pequeños).
- Estructura Flexible:
  - Permite guardar metadatos útiles
- Portabilidad:



- El archivo .db puede copiarse, respaldarse o compartirse fácilmente.
- Rendimiento Adecuado:
  - Óptimo para lecturas/escrituras secuenciales (si no hay alta concurrencia).

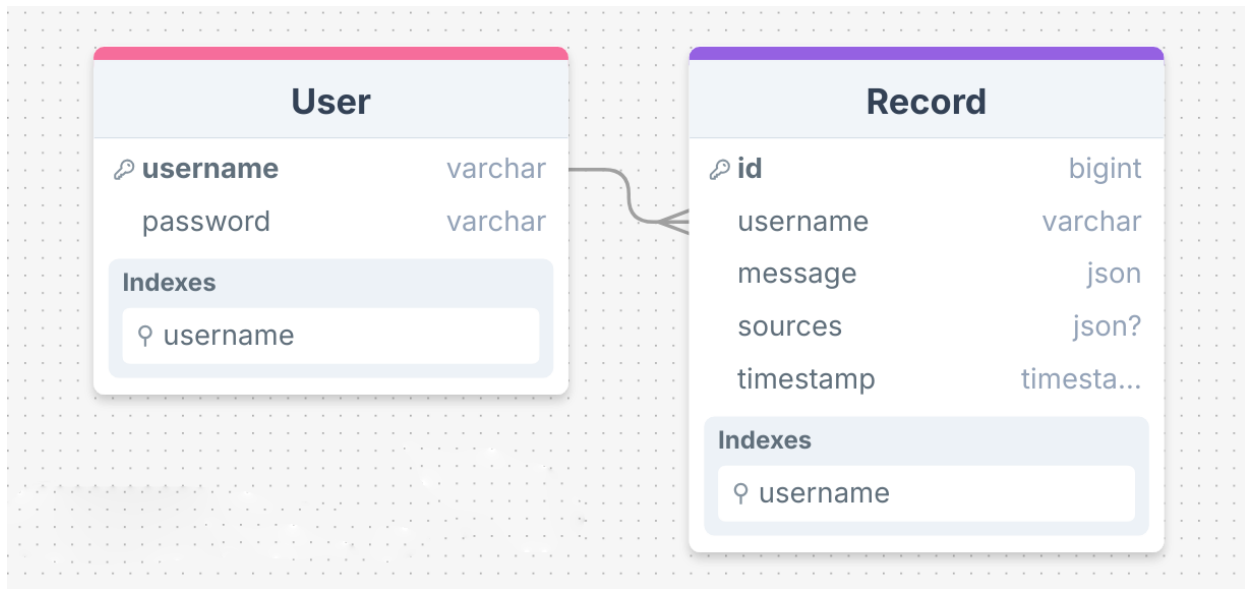
La base de datos diseñada cuenta con dos tablas: user y Record, Figura 2.1. Los campos para cada una se definen a continuación:

user:

- username: Tipo: String (cadena de texto), es la clave primaria (primary\_key=True), Tiene un índice (index=True) para búsquedas más rápidas.
- password: Almacena la contraseña del usuario.

Record:

- id: Clave primaria autoincremental (Integer), con índice para mejor performance
- username: Clave foránea (ForeignKey) que referencia al campo username en la tabla users. nullable=False indica que es obligatorio
- message: Almacena el mensaje en formato JSON (puede contener tanto la pregunta del usuario como la respuesta del *chatbot*)
- timestamp: Almacena la fecha y hora de la interacción (tipo DateTime)
- sources: Campo opcional (nullable=True) en formato JSON para almacenar fuentes de información usadas por el *chatbot*



**Figura 2.1** Diagrama relacional de la base de datos.

## 2.4 Capa de interfaz

Se emplea Streamlit el cual es un *framework* de Python diseñado para crear aplicaciones web interactivas de manera rápida y eficiente, sin requerir conocimientos avanzados en desarrollo *frontend*. Su principal ventaja radica en su simplicidad; se integra perfectamente con el ecosistema de Python, soportando pandas, NumPy y otras librerías clave. Su naturaleza *open source* y su facilidad de despliegue lo convierten en una opción popular para desarrolladores que buscan construir interfaces funcionales (Pokhrel et. al, 2024).

La configuración inicial establece el título, icono y diseño de la página. Se utiliza la base de datos explicada anteriormente para almacenar registros de conversaciones y credenciales de usuarios, manejada a través de un motor de base de datos (**engine**). La clase **StreamlitStreamingHandler** hereda de **BaseCallbackHandler** y se encarga de mostrar progresivamente las respuestas del asistente mediante un sistema de *streaming*, actualizando dinámicamente el contenedor de chat a medida que se generan nuevos *tokens* de texto.

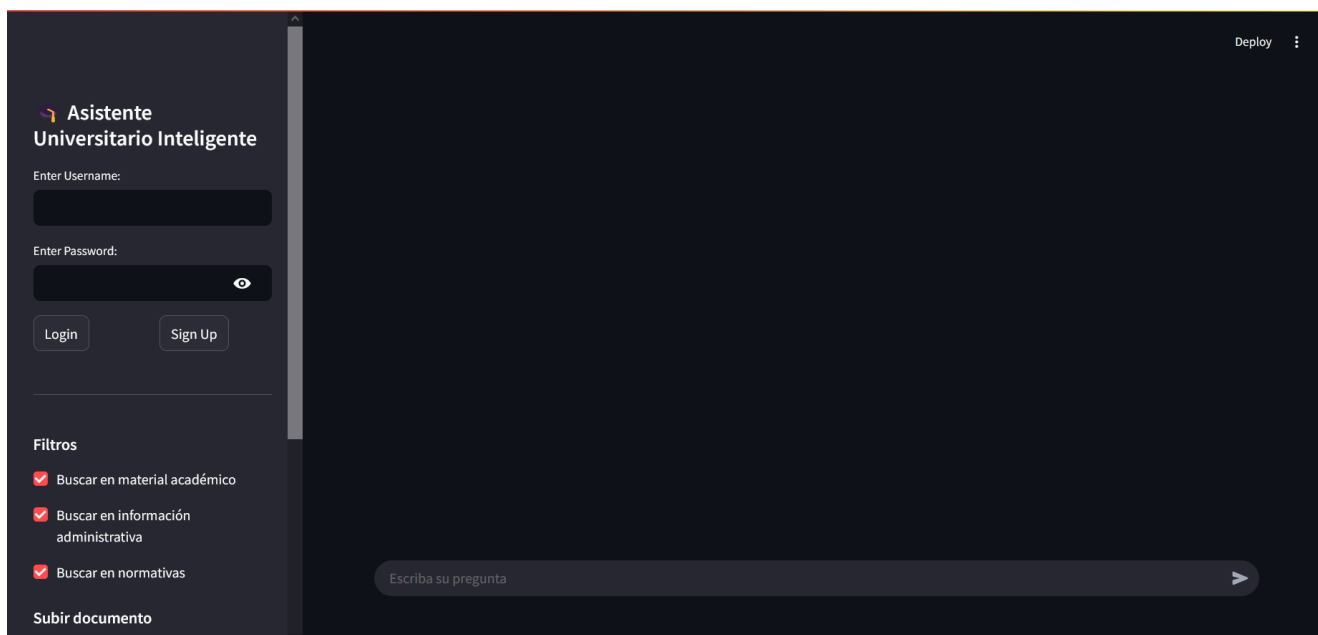
El estado de la sesión gestiona tres elementos clave: `disable_chat` (para controlar la interacción), `messages` (para almacenar el historial de la conversación) y `session_id` (para identificar la sesión). La interfaz de usuario (UI) incluye un contenedor principal para el chat y un campo de entrada condicionalmente deshabilitado. La función `write_record` muestra el historial de mensajes, diferenciando entre roles (usuario/asistente) y mostrando fuentes de información en secciones expandibles si están disponibles. La función `get_record` carga el historial previo desde la base de datos en orden inverso (del más reciente al más antiguo).

El agente de chat (**UniversityChatAgent**) se carga con caché para mejorar el rendimiento y se utiliza para procesar consultas y documentos subidos (PDF, DOCX, TXT). El panel lateral (*sidebar*) contiene, Figura 2.2:

- Autenticación: Campos para *username* y *password*, con botones de *login* y registro. El *login* verifica credenciales mediante `get_user`, habilita el chat y carga el historial si son válidas, mientras que el registro usa `create_user` para nuevos usuarios.
- Subida de archivos: Permite cargar documentos temporales que el agente procesa al enviar un mensaje.
- Nueva conversación: Un botón que limpia el historial y la base de datos para comenzar desde cero.

Cuando el usuario envía un mensaje, la aplicación:

- Almacena el mensaje en *messages* y la base de datos (`add_record`).
- Procesa documentos subidos previamente (si los hay).
- Aplica los filtros seleccionados.
- Muestra la respuesta del asistente con *streaming* (usando la clase mencionada), incluyendo un indicador de carga y fuentes citadas.
- Vuelve a habilitar el chat para continuar la interacción.



**Figura 2.2** Ejemplo de la interfaz gráfica.

## Conclusiones parciales

El diseño del sistema se fundamenta en una robusta arquitectura de cuatro capas (Procesamiento, Agente Inteligente, Memoria e Interfaz), lo que asegura una base sólida y modular que facilita su desarrollo, mantenimiento y escalabilidad. La combinación de OCR y la creación de *embeddings* almacenados en una base de datos vectorial (FAISS) permite un procesamiento eficiente de los documentos, crucial para la recuperación semántica de información. La integración exitosa del *Agentic RAG* con LangChain demuestra una conexión efectiva entre el modelo de lenguaje de gran tamaño y los flujos de trabajo de los agentes, permitiendo que el sistema no solo recupere datos, sino que también razone y actúe de manera inteligente. La inclusión de una capa de memoria basada en SQLite para el historial de conversaciones es fundamental para mantener el contexto en interacciones prolongadas, mejorando la coherencia y la naturalidad del diálogo. Finalmente, la interfaz de usuario desarrollada con Streamlit facilita la interacción del usuario, haciendo que el acceso al contenido de la Sala de Fondos Raros y Valiosos sea intuitivo y amigable.

## CAPÍTULO 3: RESULTADOS EXPERIMENTALES

La implementación de Google Cloud Vision API permitió procesar con alta eficacia los textos de la de Sala de Fondos Raros y Valiosos, digitalizando 12200 páginas en 1 hora y media. Esta constituye la entrada para la herramienta desarrollada.

### 3.1 Resultados herramienta

Para probar el agente se realizaron pruebas que usan los libros pertenecientes a la base de datos vectorial. Primero se probó la herramienta **search\_library** para comprobar el comportamiento de esta, realizando las siguientes pruebas:

#### Prueba 1

Entrada:

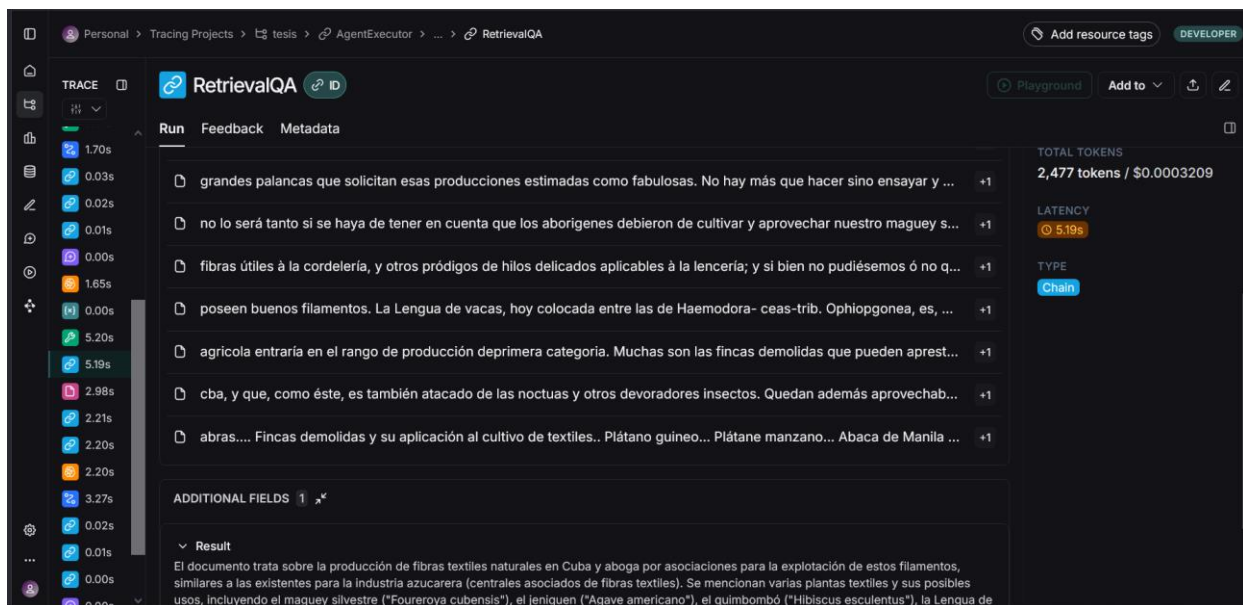
- Pregunta: “producción de fibras textiles naturales”

Salida:

- Documentos: Se encontraron 8 documentos dentro del libro “El Partido Liberal de Cuba”.
- Resultado: “El documento trata sobre la producción de fibras textiles naturales en Cuba y aboga por asociaciones para la explotación de estos filamentos, similares a las existentes para la industria azucarera (centrales asociados de fibras textiles). Se mencionan varias plantas textiles y sus posibles usos, incluyendo el maguey silvestre ("Foureroia cubensis"), el jeniquen ("Agave americano"), el quimbombó ("Hibiscus esculentus"), la Lengua de vacas ("Sanseveria guineensis"), el plátano guineo ("Musa sapientium") y el Espino de cerca ("Yuca gloriosa"). El índice del documento lista secciones dedicadas a "Cultivos y explotaciones", "Indiciación de textiles exóticos y natives...", "El ramié y su cultivo práctico...", "Fincas demolidas y su aplicación al cultivo de textiles...", "Plátano guineo...", "Cultivo de textiles que existau hoy en Cuba...", "Cultivo y producción del Quimbombó...", "Asociaciones

para el cultivo y explotación de textiles...", "Agave americano: Agave Kerailer Agave Mexicano...", y "Agave spicata: Foureroya cubanae Maguey criollo".

Con la herramienta Langsmith<sup>22</sup> se comprueba que se utilizaron 2,477 tokens, con un tiempo de respuesta de 5.19s, del cual 2.89s fue del vector store y 2.20s del LLM, Figura 3.1.



**Figura 3.1: Resultados de la herramienta visto desde Langsmith para la Prueba 1.**

## Prueba 2.

Entrada:

- Pregunta: "Terremoto 1932 Santiago de Cuba"

Salida:

- Documentos: Se encontraron 8 documentos, 7 eran del libro "El Terremoto de Santiago de Cuba ( 8 de febrero de 1932), Bernardo Redondo S.J" y uno del libro

<sup>22</sup> Langsmith es una plataforma desarrollada por LangChain para evaluar, depurar y optimizar agentes y aplicaciones construidas con modelos de lenguaje. Permite hacer seguimiento detallado del comportamiento de los LLMs, visualizar flujos de ejecución y comparar variantes de prompts o cadenas de procesamiento.

“Narraciones y Leyendas de Santiago de Cuba, Raúl Ibarra Albue” este último no contiene información relevante para esta pregunta

- Resultado: “El contexto proporcionado contiene información sobre el terremoto en Santiago de Cuba el 3 de febrero de 1932 ("MAGAZINE "LAS NOTICIAS" EL TERREMOTO EN SANTIAGO DE CUBA"). El terremoto ocurrió en la madrugada, a la 1:13 A.M., mientras la mayoría de los habitantes dormían ("En la madrugada del 3 de febrero de 1932, en los momentos en que la mayoría [...] de sus abnegados habitantes se entregaban al sueño..."). Se describe como un evento que sacudió la ciudad en todos sus barrios ("un violento terremoto sacudió en sus cimientos a la épica ciudad de Oriente, asolándola en todos sus barrios, antiguos i modernos"). Un submarino de la armada americana realizó estudios científicos para descubrir las causas del terremoto ("SUBMARINO DE LA ARMADA AMERICANA, que a las órdenes de los geólogos de la Universidad de Princenton, realizan estudios científicos para descubrir las causas que originaron el terremoto del día tres de Febrero en San- tiago de Cuba"). Además, se menciona una conmoción sísmica el 27 de marzo de 1932 que causó temor entre los habitantes de Santiago de Cuba de que se repitiera el terremoto del 3 de febrero ("este Libro Histórico del terremoto, hacemos una parada para dar- le cabida a unas líneas, sobre la conmoción sísmica del día 27 de Marzo de 1932...").”

Con Langsmith se comprueba que se utilizaron 2,799 tokens, con un tiempo de respuesta de 10.98s, del cual 7.60s fue del vector store y 3.30s del LLM.

Nota: La herramienta devuelve las palabras con las divisiones de sílabas tal como se encontró en el texto original, esto luego es arreglado por el agente.

A continuación, se muestra una tabla comparativa de ambos resultados.

**Tabla 3.1:** Comparación de pruebas.

	<b>Tokens</b>	<b>Tiempo Respuesta</b>	<b>TR Vector Store</b>	<b>TR LLM</b>	<b>Documentos relevantes (máximo 8)</b>
Prueba 1	2477	5.19s	2.89s	2.20s	8
Prueba 2	2799	10.98s	7.60s	3.30s	7

La Tabla 3.1 compara el rendimiento de dos pruebas en el RAG, donde se observa una variación significativa en el rendimiento, tanto en términos de procesamiento como de tiempos de respuesta. En la Prueba 1, el sistema manejó 2477 tokens y generó una respuesta en 5.19 segundos, de los cuales 2.89 segundos correspondieron a la recuperación de documentos en el almacén vectorial (vector store) y 2.20 segundos a la generación de la respuesta por el modelo de lenguaje (LLM). En contraste, la Prueba 2 procesó una mayor cantidad de tokens (2799), lo que se reflejó en un tiempo de respuesta total notablemente superior: 10.98 segundos. Este incremento se debió principalmente al aumento del tiempo de búsqueda en el Vector Store, que alcanzó los 7.60 segundos, mientras que el LLM empleó 3.30 segundos en generar la salida. Aunque en ambas pruebas se utilizaron un número similar de documentos relevantes (8 en la primera y 7 en la segunda), la segunda consulta implicó una recuperación más costosa, posiblemente por mayor complejidad semántica o densidad contextual. Es válido recalcar que ambas fueron respuestas correctas.

### 3.2 Resultados del agente

Para probar la capacidad de razonamiento y del comportamiento del agente se realizaron las siguientes pruebas.

**Prueba 1:** Libro ‘El Terremoto de Santiago de Cuba (8 de febrero de 1932), Bernardo Redondo S.J. ‘

Iteración: 1

Modelo: llama3

Pregunta: “terremoto 1932 Santiago de Cuba”



Respuesta: El terremoto que afectó a Santiago de Cuba el 3 de febrero de 1932 fue un evento significativo que causó considerable devastación. Tras este evento, el 27 de marzo de 1932 se produjo una conmoción sísmica adicional, lo que agravó aún más la situación.

#### Daños y Ayuda:

- Los daños causados por el terremoto se calcularon en 35 millones de pesos, una suma considerable para la época ("LOS DAÑOS DE LA CATAS TROFE SE CALCULAN EN 35 MILLONES DE PESOS").
- Se recibieron auxilios para ayudar a la población afectada ("EL TERREMOTO EN SANTIAGO DE CUBA").

#### Investigación Científica:

- Geólogos de la Universidad de Princeton realizaron estudios científicos en la zona afectada ("MAGAZINE 'LAS NOTICIAS'", "SUBMARINO DE LA ARMADA AMERICANA"), lo que subraya la importancia del evento para la comunidad científica.

#### Referencias Documentales:

- El evento fue cubierto por la revista "LAS NOTICIAS" ("MAGAZINE 'LAS NOTICIAS'").
- Bernardo Redondo S.J. escribió sobre el terremoto en "El Terremoto de Santiago de Cuba( 8 de febrero de 1932)"

Este terremoto no solo impactó a la población local, sino que también atrajo la atención de investigadores y medios de comunicación, dejando un registro detallado de sus efectos y las respuestas que generó.

#### **Prueba 2:**

##### Iteración 1:

Modelo: llama3

Pregunta: “que info tiene sobre la agricultura de textiles”

Respuesta: La agricultura de textiles se refiere al cultivo de plantas para obtener fibras naturales utilizadas en la fabricación de tejidos. Según la información encontrada en la biblioteca, un documento resalta la producción de fibras textiles naturales en Cuba y aboga por la creación de asociaciones para la explotación de estas fibras, siguiendo el modelo de las centrales azucareras. El documento menciona varias plantas textiles importantes en Cuba: \* \*\*Maguey silvestre (\*Foureroia cubensis\*)\*\* \* \*\*Jeniquen (\*Agave americano\*)\*\* \* \*\*Quimbombó (\*Hibiscus esculentus\*)\*\* \* \*\*Lengua de vacas (\*Sanseveria guineensis\*)\*\* \* \*\*Plátano guineo (\*Musa sapientium\*)\*\* \* \*\*Espino de cerca (\*Yuca gloriosa\*)\*\* Estas plantas tienen el potencial de ser utilizadas en la producción de textiles, ofreciendo alternativas a las fibras sintéticas y contribuyendo a una industria textil más sostenible. El documento también incluye secciones dedicadas a los cultivos, la indización de textiles exóticos y nativos, el ramié, y la aplicación de fincas demolidas al cultivo de textiles. La información disponible en la biblioteca se centra en el contexto cubano y proporciona detalles específicos sobre los tipos de plantas y enfoques para la producción de fibras textiles naturales en esa región.

Los resultados anteriores se pueden ver en la siguiente figura:

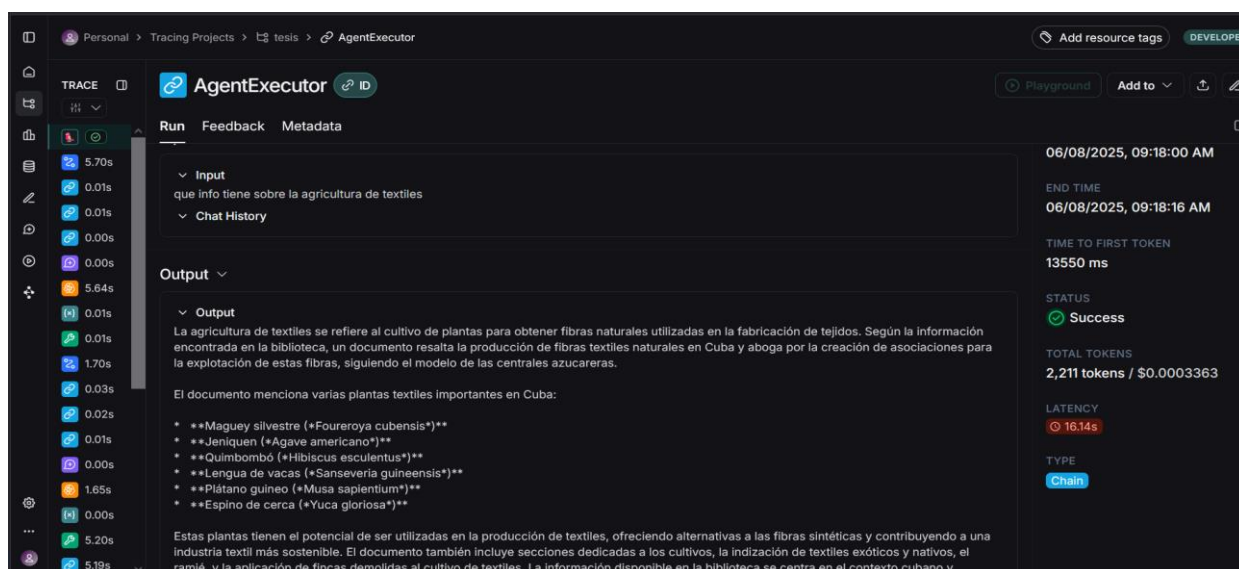


Figura 3.2: Resultados del Agente para la Prueba 2.

Iteración: 2 Pregunta basada en el contexto anterior

Query : podría buscar en el documento de que cultivos se hablan

Respuesta:

El documento "TEXTILES. Cultivo y explotación de las especies nativas de Cuba, y de otras exóticas aprovechables" por Don Sebastian Alfredo de Morales, menciona los siguientes cultivos textiles específicos:

Corojo criollo (crocomia lasiospatha o Gastrococcus armentalis) (p. 3)

Maguey (Agave americana L.) y sus variedades como Magaey, Honiquen, Cocuiza, Cabuyn (p. 4)

Quimbombó (Hibiscus esculentus) (p. 6)

Plátano guineo, plátano manzano, plátano burro y otros plátanos (p. 5)

Abacá de Manila (Musa textilis syleestris Rumph.) (p. 5)

Jeniquen y Heniquon (p. 2)

Malva-ross-Hibiscus mutabilia y otras Malvaceas (p. 2)

Lengua de vaca, o piel de majá (Sansevieria Guineensis o longas de vaca) (p. 2)

A continuación, se muestra una tabla comparativa de ambos resultados.

**Tabla 3.2:** Comparación de pruebas.

	<b>Tokens</b>	<b>Tiempo Respuesta Iteración 1</b>	<b>Tiempo Respuesta Iteración 2</b>	<b>Documentos Relevantes (máximo 8)</b>
Prueba 1	5,129	15.51s	---	7
Prueba 2	3,549	16.14s	6.77s	8

Después de realizar las pruebas anteriores, se pudo ver un buen comportamiento y una búsqueda acertada de la información, con bajo nivel de alucinación por parte del agente

y respuestas detalladas, las cuales incluyen las fuentes de donde fueron extraídas, Estas en su mayoría eran relevantes para el contexto de la pregunta. La velocidad de respuesta es menor a los 20 segundos lo cual es un desempeño adecuado.

### **Conclusiones parciales**

Las pruebas revelaron un rendimiento sólido tanto de la herramienta de búsqueda como del agente. La herramienta `search_library` demostró una capacidad precisa para encontrar información relevante con baja alucinación, como se evidenció en las pruebas sobre el "Partido Liberal de Cuba" y el "Terremoto 1932 Santiago de Cuba". Ambas pruebas mostraron tiempos de respuesta eficientes, con la mayoría de los documentos recuperados siendo directamente pertinentes a las consultas.

De manera similar, el agente exhibió un buen comportamiento y razonamiento, proporcionando respuestas detalladas con mínima alucinación, incluso identificando fuentes específicas para sus respuestas. Esto fue claro en la Prueba 1 sobre el terremoto, donde la información incluyó detalles sobre daños, ayuda e investigación científica, con referencias claras a publicaciones. La Prueba 2 sobre agricultura textil demostró la capacidad del agente para profundizar en la información, logrando identificar cultivos específicos en una iteración posterior. La velocidad de respuesta general del sistema se mantuvo consistentemente por debajo de los 20 segundos, lo que indica un desempeño eficiente y robusto.

## CONCLUSIONES GENERALES

Basándose en el análisis realizado, las conclusiones generales de este trabajo son las siguientes:

- La implementación de Google Cloud Vision API permitió procesar con alta eficacia los textos de la Sala de Fondos Raros y Valiosos, digitalizando 12200 páginas en 1 hora y media y preservando su estructura semántica.
- La combinación de LLM locales (Llama 3), arquitecturas *Agentic RAG* (LangChain) y bases vectoriales (FAISS) demostró ser un avance para acceder al patrimonio documental, ofreciendo respuestas contextualizadas, dinámicas y verificables en el *chatbot* autónomo, lo que optimizó la interacción usuario-conocimiento bajo principios de soberanía tecnológica.
- Los sistemas *Agentic RAG* validaron su idoneidad para entornos patrimoniales complejos al permitir razonamiento autónomo, gestión de contextos y ejecución local segura (FAISS/SQLite).
- Las pruebas experimentales confirmaron la robustez del sistema: baja tasa de alucinación, respuestas documentadas con fuentes precisas (7-8 documentos relevantes por consulta) y tiempos de respuesta inferiores a 20 segundos, cumpliendo los requisitos de eficiencia para consultas especializadas

## RECOMENDACIONES

Para futuras investigaciones, se pueden considerar las siguientes recomendaciones basadas en el trabajo realizado:

- Explorar la aplicación de este sistema a otros fondos documentales de la Universidad de Oriente o de otras instituciones. Esto podría incluir archivos históricos, tesis, revistas científicas o colecciones especiales
- Investigar la integración de capacidades multimodales en el *chatbot*. Esto significaría que el sistema no solo procese texto, sino también imágenes, gráficos, tablas y otros elementos visuales presentes en los documentos
- Devolver las fuentes de donde el *chatbot* extrajo la información.
- Continuar la investigación en la optimización de los modelos LLM y las arquitecturas RAG para reducir los requisitos computacionales y mejorar la eficiencia del sistema

## REFERENCIAS BIBLIOGRÁFICAS

1. 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), 1-45.
2. Chiong Lanegra, L. D. (2012). *Medición de la calidad del servicio de la Biblioteca de la Facultad de Letras y Ciencias Humanas de la UNMSM: una experiencia con LIBQUAL+ TM* (Doctoral dissertation, Universidad Nacional Mayor de San Marcos).
3. Gou, Z., Shao, Z., Gong, Y., Shen, Y., Yang, Y., Duan, N., & Chen, W. (2023). *Critic: Large language models can self-correct with tool-interactive critiquing*. *arXiv preprint arXiv:2305.11738*.
4. Guitián, M. V. (2009). *Una nueva visión de las Bibliotecas Universitarias en el contexto actual*. *Contribuciones a las Ciencias Sociales*, (2009-11).
5. Han, Y., Liu, C., & Wang, P. (2023). *A comprehensive survey on vector database: Storage and retrieval technique, challenge*. *arXiv preprint arXiv:2310.11703*.
6. Huang, J., & Chang, K. C. C. (2022). *Towards reasoning in large language models: A survey*. *arXiv preprint arXiv:2212.10403*.
7. Huang, X., Liu, W., Chen, X., Wang, X., Wang, H., Lian, D., ... & Chen, E. (2024). *Understanding the planning of LLM agents: A survey*. *arXiv preprint arXiv:2402.02716*.
8. Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., ... & Liu, T. (2025). *A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions*. *ACM Transactions on Information Systems*, 43(2), 1-55.
9. Junyan, L., Shiguo, X., & Yijie, L. (2009, May). *Application research of embedded database SQLite*. In *2009 International Forum on Information Technology and Applications* (Vol. 2, pp. 539-543). IEEE.

10. Ocaña-Fernández, Y., Valenzuela-Fernández, L. A., & Garro-Aburto, L. L. (2019). *Inteligencia artificial y sus implicaciones en la educación superior. Propósitos y representaciones*, 7(2), 536-568.
11. Pokhrel, S., Ganesan, S., Akther, T., & Karunarathne, L. (2024). *Building Customized Chatbots for Document Summarization and Question Answering using Large Language Models using a Framework with OpenAI, Lang chain, and Streamlit. Journal of Information Technology and Digital World*, 6(1), 70-86.
12. Ravuru, C., Sakhinana, S. S., & Runkana, V. (2024). *Agentic retrieval-augmented generation for time series analysis. arXiv preprint arXiv:2408.14484*.
13. Singh, A., Ehtesham, A., Gupta, G. K., Chatta, N. K., Kumar, S., & Khoei, T. T. (2024). *Exploring prompt engineering: A systematic review with swot analysis. arXiv preprint arXiv:2410.12843*.
14. Singh, A., Ehtesham, A., Kumar, S., & Khoei, T. T. (2024, May). *Enhancing ai systems with agentic workflows patterns in large language model. In 2024 IEEE World AI IoT Congress (AllIoT) (pp. 527-532). IEEE*.
15. Zhao, P., Zhang, H., Yu, Q., Wang, Z., Geng, Y., Fu, F., ... & Cui, B. (2024). *Retrieval-augmented generation for ai-generated content: A survey. arXiv preprint arXiv:2402.19473*.
16. Zhang, Z., Bo, X., Ma, C., Li, R., Chen, X., Dai, Q., ... & Wen, J. R. (2024). *A survey on the memory mechanism of large language model based agents. arXiv preprint arXiv:2404.13501*.