Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE VALLADOLID

Grado en Ingeniería Informática
Mención en computación

Desarrollo de una aplicación basada en técnicas RAG y LLM para asistencia normativa

---

Alumno: Rodrigo García Rubio

Tutores:
Mª Aránzazu Simón Hurtado
Fernando Adolfo Tejerina

Índice general

Índice de figuras

Índice de tablas

Dedicado a mis padres
que me apoyaron pese a todo
y a mis amigos
por recordarme

Resumen

Hoy en día, muchas organizaciones y empresas recurren a la Inteligencia Artificial (IA) para manejar sus enormes cantidades de datos y usan chatbots entrenados para mejorar su atención al cliente y resolver las dudas menores de forma más eficiente. Sin embargo, esto puede dar lugar a muchas complicaciones y costos, por ejemplo, la dificultad de preparar el sistema necesario, el precio del hardware necesario para entrenar los modelos o cumplir con los requisitos de privacidad y personalización. Para solventar estos problemas, el objetivo de este proyecto será desarrollar una aplicación web para asistir en la búsqueda de información documental concreta para organizaciones potenciado por tecnología *Retrieval Augmented Generation* (RAG), es decir, un asistente de IA que no haga falta entrenar que responda preguntas sobre documentación. Esta aplicación resolvería varios de los problemas ya mencionados, ya que sería fácil de instalar y preparar, y no necesitaría la potencia exigida por el proceso de entrenamiento

Palabras claves: RAG, chatbot, Inteligencia Artificial, IA, Retrieval Assisted Generation.

Abstract

Lots of organisations these days are looking towards Artificial Intelligence (AI) as an efective way to manage large amounts of data, and turn to trained chatbots to improve customer service and handle simple inquiries more efficiently. However, these solutions can bring along new complications and extra costs, such as the difficulty of setting up the necessary system, the cost of the hardware required to train the models or how to achieve the required privacy and personalization especifications. The objective of this project is, thus, to create a *Retrieval Augmented Generation* (RAG) technology-powered web application capable of assisting in the search and retrieval of documented information. In other words, an AI assistant that answers based on documentation without any training. This app would be easy to install and set up and would require only the necessary hardware to run, solving several of the previously mentioned problems in one swoop.

Keywords: RAG, chatbot, Artificial Intelligence, AI, Retrieval Assisted Generation.

Estructura de la Memoria

Este documento sigue la estructura que se describe a continuación:

En el capítulo 1 se introduce el proyecto y se expresan motivaciones principales y objetivos de su realización. También se indican posibles aplicaciones para el producto y se explica el entorno en el que se encuentra la tecnología usada.

En el capítulo 2 se concreta la metodología usada en el desarrollo del producto, las herramientas que se han empleado para ello, la arquitectura básica en la que se basará el producto y las abreviaturas empleadas.

En el capítulo 3 se explica la planificación temporal del proyecto y la gestión de riesgos.

El capítulo 4 corresponde al análisis software del proyecto en el que se identifican usuarios de la aplicación y requisitos funcionales y no funcionales. También se explican en profundidad los casos de uso de la aplicación, la organización de la aplicación en paquetes y el desarrollo de los casos de uso entre los componentes de la aplicación.

En el capítulo 5 se explica el Diseño Software de la aplicación.

En el capítulo 6 se describe el procedimiento usado para evaluar los componentes del sistema y optimizar la configuración del sistema.

En el capítulo 7 se explica el funcionamiento de los componentes externos y la implementación final del proyecto.

En el capítulo 8 se indican las pruebas realizadas para evaluar las características de la aplicación.

En el capítulo 9 se reúnen las conclusiones obtenidas del proyecto y se discuten los resultados y otras opciones no exploradas que se dejan como líneas futuras de investigación.

En el anexo se incluyen manuales de instalación, gestión y mantenimiento del sistema.

Parte I

Memoria del Proyecto

Capítulo 1

INTRODUCCIÓN Y OBJETIVOS

1.1. Introducción

Este trabajo pretende poner en funcionamiento una aplicación que compense una de las principales deficiencias que se detectan habitualmente en los modelos de lenguaje "*LLM*"(Large Language Model). Esta dificultad reside en que cuantos más documentos se usan para entrenar un modelo, este producirá respuestas más generales que pueden ser erróneas y no estar en consonancia con las preguntas del usuario.

Sin embargo, entrenarlo con textos específicos, como manuales no públicos o grandes cantidades de un tipo concreto de documento, para acotar las respuestas y mejorar la precisión requiere de un hardware más específico y caro e implementar la solución puede resultar muy difícil y requerir un equipo de expertos. Esto puede aumentar los gastos tanto como para que las *LLM* dejen de compensar al posible cliente.

Una de las mejores alternativas, al menos en lo relativo a la relación calidad-precio de los modelos de lenguaje, es aplicar técnicas de *RAG*, o *Retrieval Augmented Generation*. Resumidamente, las técnicas *RAG* permiten a un *LLM* acceder a un conocimiento concreto, preciso y explícitamente relacionado con la petición del usuario, sin necesidad de incurrir en gastos los gastos excesivos que exige el correcto entrenamiento de una *LLM* [15] [10].

El propósito de este proyecto es, por tanto, mostrar la capacidad de las técnicas RAG para mejorar la utilidad de las LLM a través de una aplicación web sencilla, que permitirá **resolver dudas concretas sobre, por ejemplo, largos textos legales, como la ley educativa**.

Esta aplicación podría usarse en quioscos digitales en bibliotecas u oficinas para resolver dudas sencillas y reducir la demanda de asistentes humanos, o como una herramienta adicional en páginas web ya existentes para ayudar a personas que tengan dudas sencillas o no estén familiarizadas con el material original o la estructura del sitio web.

1.2. Estado de la cuestión

La tecnología RAG [12] es relativamente reciente, pero ya tiene varios campos de aplicación e investigación, como en la medicina. En este ámbito, los modelos de lenguaje ampliados con RAG resultan extremadamente útiles, dado que el conocimiento requerido es muy concreto, es muy abundante y no todo está disponible al público general, por lo que no se puede usar como entrenamiento para modelos de lenguaje.

Por ejemplo, la eficiencia de las técnicas RAG ha sido probada en investigaciones sobre sus aplicaciones al análisis de pacientes médicos [15], donde se han llegado a encontrar mejoras del 18 % de precisión respecto a ChatGPT en varias especialidades, y otro estudio posterior, más exhaustivo, encontró mejorías notables en precisión, comprensibilidad, satisfación de usuario y resiliencia a ataques de entrada maligna respecto a otros modelos de LLM, como ChatGPT-4, BingChat, y Bard AI [29].

Otro estudio sobre la viabilidad de los LLM mejorados mediante RAG en la interpretación de directrices clínicas en hepatología [10] ha obtenido resultados que indican que añadir el sistema RAG a una LLM ya existente puede mejorar la precisión de las respuestas a partir de un 43 % hasta una precisión de entre 67 % hasta un 99 %, dependiendo del tipo y cantidad de preprocesado de la documentación alimentada al sistema.

También existen otras aplicaciones dirigidas a usuarios finales, aunque, debido a lo reciente que es esta tecnología, la mayoría están en fase de desarrollo, excepto NotebookLM [16] de Google. Esta aplicación web permite al usuario subir sus propios documentos para que el LLM de Google, Gemini-1.5, pueda trabajar sobre estos, creando resúmenes, tablas, gráficos o incluso audio.

Sin embargo, esta aplicación de Google está orientada al usuario final, y no permite alojarla de forma local y exponerla al público, que es el propósito de la aplicación que se desarrolla en este TFG. Además, todos los documentos que se suben a NotebookLM se alojan en los servidores de Google, añadiendo un riesgo a la protección de datos inasumible si se quiere usar con documentos privados.

1.3. Objetivos del trabajo

El objetivo general de este proyecto es desarrollar una aplicación web que permita la búsqueda de información concreta de forma rápida aprovechando la potencia de las LLMs combinadas con técnicas de RAG (Retrieval Augmented Generation).

A partir de esta aplicación, podrán crearse variantes modificadas para ajustarse a las distintas necesidades de sus entornos o tipos de datos que manejen, pero la base sería la misma.

Los objetivos específicos del proyecto serían los siguientes:

- **Redactar un plan de desarrollo del software**:
  establecer una metodología de desarrollo y planificar los pasos a seguir, así como documentar el proceso.

- **Construir aplicación web funcional base**:
  el objetivo es obtener un producto usable y funcional que sirva como prueba de concepto para estudiar las mejoras de la utilización de técnicas RAG junto con LLMs.

- **Crear un sistema de distribución sencillo**:
  será un única imagen Docker o comprimido-ejecutable, de forma que sea fácil de instalar, ejecutar, mantener, actualizar y desinstalar como fuese necesario.

Los objetivos académicos del proyecto son los siguientes:

- **Aplicar lo aprendido durante la carrera**:
  aplicar los diversos conocimientos adquiridos a lo largo del curso en las distintas asignaturas, como planificación, bases de datos y diseño de interfaces.

- **Profundizar en el conocimiento sobre LLMs, prompt engineering y bases de datos vectoriales**:
  aprender más sobre nuevas herramientas, y sobre formas novedosas de usarlas en conjunto, combinando grandes modelos de lenguaje con bases de datos de vectores embebidos.

1.4. Campos de aplicación de este trabajo

Se definen los siguientes campos de aplicación como ejemplos de uso del sistema para establecer el alcance de la aplicación que se desarrollará en este proyecto:

1. Se podría instalar la aplicación en un quiosco de atención al cliente en secretarías del estado, con el objetivo de permitir a los usuarios resolver sus propias dudas dejando libres a los ayudantes humanos de las oficinas para que se encarguen de problemas más importantes.

   Estos quioscos solo necesitarían un navegador web y acceso a la red local, ya que en la misma habría un servidor ejecutando el servicio, o en una oficina centralizada a la que se conecte de forma remota. Podrían resolver dudas legales pequeñas referenciando una Base de Datos (BD) de documentos legales, como la ley educativa, de herencias, etc.

2. Podría establecerse un servidor local en centros educativos para que los profesores puedan resolver dudas respecto a la ley educativa. El coste de la implantación de este servicio de forma local y auto-gestionada podría ser algo elevado, pero permitiría poder añadir datos sensibles y confidenciales, como notas de estudiantes o gestión de inventario, haciendo la información más accesible para el profesorado.

3. En universidades podría usarse como combinación de los dos anteriores casos de uso: para los estudiantes, podría resolver dudas respecto a la normativa, plazos de matrículas o condiciones para ciertos procesos administrativos, aliviando presión para el negociado, mientras que para los profesores, podría ayudar a buscar e interpretar ciertos datos tabulados y hacer inferencias sobre estos, hasta cierto punto, así como resolver dudas sobre las tareas que conlleva cada puesto administrativo y la organización de la universidad. Además, las bibliotecas cuentan con más recursos y no es de extrañar que cuenten con una sala de servidores ya funcionales y un equipo de expertos, por lo que los gastos de implementación serían mucho menores que en un colegio normal.

Capítulo 2

Metodología

En este capítulo se detallarán las cuestiones metodológicas, es decir, las metodologías y herramientas que se han utilizado para plantear el trabajo.

2.1. Proceso de desarrollo

Debido al uso del desarrollo por componentes para implementar la aplicación y al control y conocimiento casi completo de los requisitos del programa, el desarrollo en cascada, como se muestra en la Figura 2.1, se convierte en una opción viable como modelo de desarrollo para la aplicación del proyecto.

La adherencia al esquema no tendría que ser completamente estricta, y podría permitirse volver a fases del desarrollo anteriores, como volver a la fase de diseño si se encuentra un error importante o repetir la implementación y pruebas en bucle varias veces si fuese necesario.

La fase de mantenimiento, dado que esta aplicación solo se está desarrollando como parte de este TFG y no hay planes para implementarla de forma seria en ningún contexto real, consistiría en la creación de manuales de instalación, puesta en funcionamiento y mantenimiento básico, que se añadirían al Anexo.

2.2. Herramientas utilizadas

La aplicación se basa en varias tecnologías, algunas de ellas son esenciales, mientras que otras pueden ser reemplazadas sin mucho problema.

Las esenciales serían las siguientes:

- **Ollama** [17]: permite ejecutar LLMs de forma local a partir de archivos *gguf* o *bin* como un servicio de Windows, e incluye una librería de modelos ya adaptados para ejecutarse localmente. El reemplazo más parecido sería LM Studio, pero no

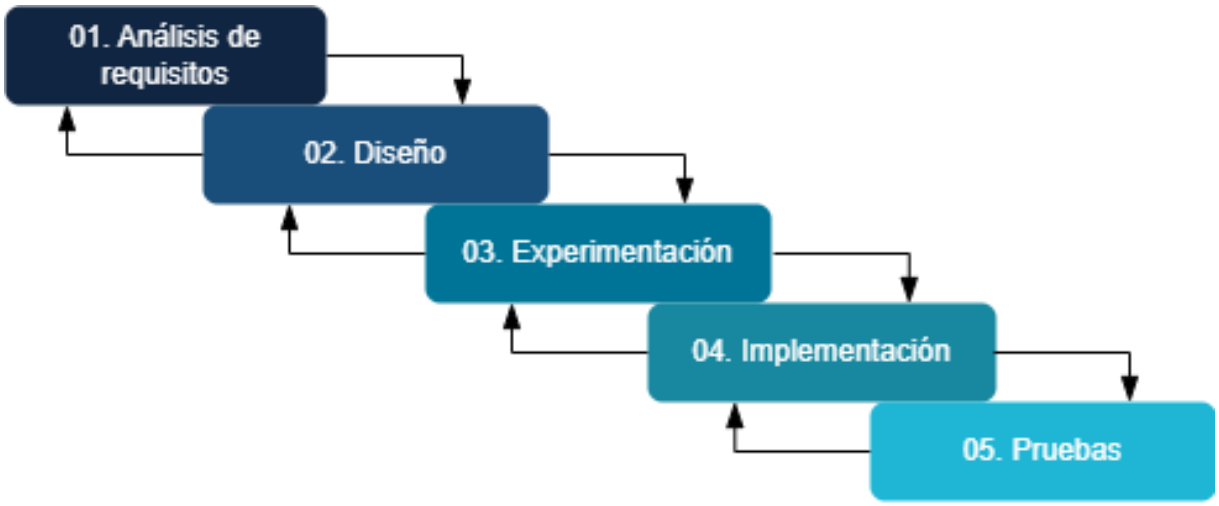


Figura 2.1: Modelo de desarrollo en cascada

ofrece las mismas ventajas de Ollama, tales como ejecutarse como servicio de Windows de forma automática, una interfaz simple para descargar y ejecutar modelos e integración con la librería Langchain.

- **Langchain** [11]: librería que contiene gran cantidad de utilidades que permiten conectar de forma sencilla LLM, prompt, y base de datos, mediante su sistema propio de "cadenas". Muy optimizado y relativamente fácil de usar, comparado con hacerlo todo a mano.
- **Docker**: entorno y motor de ejecución de contenedores para usar Chroma de forma más sencilla.

Otras herramientas usadas directamente en la aplicación, aunque no sean irreemplazables, son:

- **Chroma** [8]: base de datos vectorial. Cada base de datos Chroma organiza los documentos en Colecciones nombradas, lo que permite mantener conjuntos de documentos separados y asignar distintas propiedades y restricciones a cada uno. Cada documento es transformado
- **Gradio** [21]: librería de interfaces rápida para aplicaciones de chat.
- **Visual Code Studio**: editor de texto usado para programar la aplicación y gestionar

Por último, otros recursos involucrados en la realización del TFG:

- **Overleaf**: editor online de LaTeX.

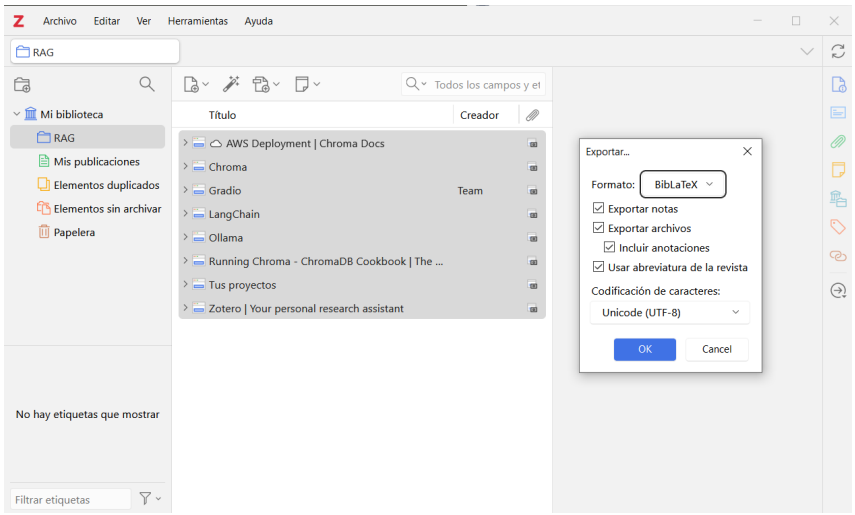


Figura 2.2: Exportar bibliografía parcial desde Zotero

- **Zotero**: aplicación para elaborar la bibliografía.
- **Draw.io**: aplicación web para creación de diagramas UML de forma simple
- **Visual Paradigm**: herramienta de modelado visual y gestión de proyectos software. En este proyecto se ha usado principalmente para realizar varios esquemas y diagramas asistiendo en el análisis y diseño de la aplicación.
- **Equipo de desarrollo**: ordenador personal con GPU dedicada.
- **Equipo de pruebas**: máquina virtual proporcionada por la universidad.

Capítulo 3

Planificación

En este capítulo se aboradarán las cuestiones relativas a la planificación del trabajo.

3.1. Planificación temporal

Para la planificación temporal estimada, se ha usado como referencia principalmente la experiencia personal realizando proyectos de desarrollo durante la carrera. Sin embargo ninguno se acerca a la escala de un TFG, por lo que también se han usado referencias externas, como otros TFGs, para estimar la longitud de las actividades.

La estimación inicial de las tareas del proyecto se puede ver en la tabla 3.1 y su planificación se encuentra en el diagrama de Gantt en la Figura 3.1. La longitud de las tareas en el diagrama y en el la tabla no se corresponden ya que que el diagrama solo muestra la duración como el tiempo entre la fecha de inicio y fin de cada tarea, mientras que en la tabla se muestran las horas dedicadas reales.

Tabla 3.1: Planificación temporal de las tareas del proyecto

Nombre	**Fecha de inicio**	**Fecha de fin**	**Duración**
Planificación	14/08/2024	23/08/2024	16
Análisis	24/08/2024	03/09/2024	33
Diseño	04/09/2024	18/09/2024	45
Experimentación	19/09/2024	05/10/2024	51
Codificación	06/10/2024	01/11/2024	81
Pruebas	02/11/2024	17/11/2024	50
Documentación	28/09/2024	17/11/2024	24

3.2. Gestión de riesgos

La gestión de riegos tendrá en cuenta todos los riesgos que puedan afectar a la finalización exitosa de este proyecto, considerando la probabilidad de cada uno y su impacto si se diese. También consiste en calcular el impacto de cada uno y establecer planes para su prevención y para reducir su impacto.

1. Planificación pobre:
   Una mala planificación puede afectar al resto de fases del desarrollo en cascada y acumular errores, lo que forzaría replantear la organización del proyecto puede que desde el principio.

2. Tecnologías nuevas:
   Este proyecto usa muchas herramientas y técnicas nuevas, en desarrollo continuo que pueden presentar fallos, o quizás carezcan de ciertas opciones comunes que pueden afectar al proyecto en calidad o tiempo.

3. Uso de herramientas con poca experiencia:
   Aunque poseo cierta experiencia, no es suficiente como para considerarme un experto al usar estas herramientas, por lo que pueden surgir imprevistos con detalles sobre su uso que no podría prever.

4. Problemas durante la implementación de la aplicación:
   Por falta de experiencia técnica personal a la hora de llevar a cabo proyectos de desarrollo, podrían surgir problemas durante la implementación ralentizando el proyecto.

5. Análisis de requisitos limitado:
   La ausencia o cambio de un requisito debido a un cambio de idea en la dirección del proyecto podría llevar a cambios en cascada a través de todo el desarrollo.

6. Problemas en la ejecución de las pruebas:
   Si la instalación o ejecución de la aplicación falla por motivos externos, como fallos en la máquina virtual de la universidad, habría que cambiar el planteamiento de las pruebas o posponerlas.

7. Menor disponibilidad de la prevista:
   Si el máster que estoy cursando al mismo tiempo que llevo a cabo este proyecto resulta llevar más tiempo del esperado, el proyecto deberá posponerse.

8. Fallo de equipamiento del proyecto:
   Aunque sea improbable, el fallo de mi ordenador personal podría retrasar substancialmente el proyecto hasta que una substitución sea preparada, además de causar la pérdida del código y documentación del proyecto, aumentando el impacto. Es poco probable, pero conviene considerar esta posibilidad.

En la tabla 3.2 puede verse la matriz de impacto y probabilidad utilizada para calcular la importancia de los riesgos anteriores.

Probabilidad	Amenazas				
0.90	0.05	0.09	0.18	0.36	0.72
0.70	0.04	0.07	0.15	0.28	0.56
0.50	0.03	0.05	0.1	0.2	0.4
0.30	0.02	0.03	0.06	0.12	0.28
0.10	0.005	0.01	0.02	0.04	0.08
	Muy Baja	Baja	Moderada	Alta	Muy Alta
	0.05	0.1	0.2	0.4	0.8

Tabla 3.2: Matriz de probabilidades e impacto

La tabla 3.3 indica la calificación de cada uno de los riesgos del proyecto previamente mencionados. Como se puede ver, no hay muchos riesgos, pero destaca el 4 (Problemas durante la implementación) ya que puede llevar más tiempo que el resto y requerirá más atención.

Los riesgos se gestionarán de las siguientes formas:

1. Para mitigar los efectos de una mala planificación, se permitirá volver atrás en el proceso de desarrollo para modificar los aspectos de la planificación que hayan quedado desfasados o sean erróneos a medida que se avance durante el desarrollo.

2. Se pueden encontrar herramientas alternativas para algunas de las herramientas usadas y se mantendrá registrado el número de versión de las herramientas en desarrollo, para evitar que nuevas actualizaciones rompan sistemas ya funcionales.

3. Se dedicará tiempo adicional a la preparación y estudio de dichas herramientas y componentes para evitar dificultades más adelante.

4. Se dedicará tiempo exclusivamente para solucionar estos problemas. Si son propios del diseño de la implantación, se volverá a una fase anterior para replantear el diseño, creando o modificando varios modelos de implantación.

5. Se revisarán las necesidades con usuarios potenciales reales para minimizar el riesgo de requisitos incompletos o correctos. Para minimizar el impacto si fuese a ocurrir, se tenderá a incluir requisitos más estrictos.

6. Se establecerá una opción de implantación por defecto si las pruebas no fueran posibles. Varias de las pruebas podrían ejecutarse en mi máquina personal y extrapolar a equipos de mayor tamaño.

Riesgo	Probabilidad	Impacto	Calificación
1	0.8	0.3	0.24
2	0.6	0.2	0.12
3	0.7	0.3	0.21
4	0.4	0.7	0.28
5	0.5	0.4	0.20
6	0.3	0.9	0.27
7	0.8	0.1	0.08
8	0.1	0.9	0.8

Tabla 3.3: Calificación de los riesgos

7. Se ajustará la planificación como se vea adecuado para ajustarse a las nuevas restricciones de tiempo.

8. Se harán copias de seguridad de mi equipo personal, además de copias en servicios de almacenamiento en la nube. El servicio de Overleaf guarda los documentos en la nube de forma automática y el proyecto se mantendrá al día en Github.

3.3. Seguimiento del proyecto

La planificación inicial del proyecto no se llegó a cumplir al final. Al mismo tiempo que completaba este proyecto, también me encontraba cursando el Máster de Ingeniería Informática de la UVa y terminando unas prácticas en una empresa externa. La planificación anterior se realizó suponiendo que tanto el máster como las prácticas serían mucho más manejables de lo que resultaron ser, dando lugar a varios retrasos.

Además, durante la planificación inicial predije que la fase de experimentación sería mucho más corta, lo cual resultó ser incorrecto y acabó siendo una de las fases que más tiempo consumieron debido a varias dificultades técnicas, los problemas de disponibilidad ya mencionados y a un mayor peso de la etapa en el proyecto total que el predicho inicialmente.

El desarrollo temporal del proyecto se puede ver en la tabla de 3.4 y en el diagrama de Gantt de la Figura 3.2

3.4. Costes del proyecto

El presupuesto para el desarrollo de este proyecto se va a dividir en dos categorías: El hardware necesario para llevar a cabo el proyecto y el capital humano.

Tabla 3.4: Duración real de las tareas

Nombre	Fecha de inicio	Fecha de fin	Duración
Planificación	14/08/2024	23/08/2024	10
Análisis	24/08/2024	23/09/2024	32
Diseño	24/09/2024	15/11/2024	54
Experimentación	16/11/2024	22/01/2025	70
Codificación	23/01/2025	12/03/2025	50
Pruebas	13/03/2025	14/04/2025	33
Documentación	24/09/2024	13/05/2025	51

Esta sección no representa costes reales del proyecto, al usar equipamiento que existente o proporcionado por la UVa, sino estimaciones sobre su coste potencial.

Hardware necesario

- Ordenador portátil: el ordenador usado para el desarrollo del sistema y redacción de la memoria es un MSI GF63 del 2020. Cuenta con tarjeta gráfica dedicada Nvidia GTX 1650 Ti. Tiene una vida útil estimada de 5 años. Coste estimado: 800 €.
- Máquina virtual: equipo usado durante la fase de experimentación y pruebas del proyecto. Cuenta con una tarjeta gráfica dedicada Nvidia A2-1Q. Disponibilidad completa y remota 24/7. Coste estimado de un equipo físico con capacidades similares: 1.500 €. Vida úti estimada: 3 años.

Debida a la larga vida útil del Ordenador Portátil, el coste asociado a su uso durante los 9 meses de duración del proyecto es de 120 €, de acuerdo a la operación

$$\text{Coste} = \frac{800\ €}{5\ \text{años} * 12\ \text{meses}} * 9\ \text{meses} = 120€$$

Para la máquina virtual, debida a su disponibilidad continua y por simplificar los cálculos, se tratará como un equipo físico. Por tanto, su coste asociado al proyecto será de 375 €, según la operación

$$\text{Coste} = \frac{1500\ €}{3\ \text{años} * 12\ \text{meses}} * 9\ \text{meses} = 375€$$

Software necesario

El software utilizado para el desarrollo software del proyecto es el siguiente:

- Visual Studio Code: IDE para toda la programación. Coste: gratis.
- Overleaf: IDE web para redacción de la memoria. Coste: 79 €al año para estudiantes.

- Ollama: parte del programa desarrollado. Ejecuta LLMs. Coste: gratis.
- Chroma: componente del programa desarrollado. BD vectorial de documentos y *embeddings*. Coste: gratis.
- Docker: herramienta externa para la ejecución de un componente esencial (Chroma) mediante virtualización. Coste: gratis.
- Visual Paradigm: herramienta de modelado visual para esquemas y diagramas complejos. Coste: 6 €/mes.
- Draw.io: web app de creación rápida y flexible de diagramas. Coste: gratis.

De estos programas, solo Overleaf y Visual Paradigm ofrecen servicios de pago, mientras que el resto son gratuitos.

Al repartir los costes entre el tiempo dedicado al proyecto, la licencia anual de Overleaf supone un gasto de 59,25 €. El coste de 9 meses de suscripción a Visual Paradigm es de 54 €.

Capital humano

El proyecto requiere de un desarrollador de software con conocimientos de modelos de lenguaje y desarrollo de aplicaciones web para llevar a cabo el proyecto. El coste anual basándonos en la plataforma Glassdoor es de 22.000 €.

A continuación se calcula el coste del desarrollador para las 300 horas de trabajo que el proyecto debe durar.

Cálculo del coste:

Horas anuales de trabajo: 1.760 (considerando una jornada laboral de 8 horas diarias y 220 días laborables al año).

Coste por hora: 22.000 €/ 1.760 horas aproximadamente 12,5 €por hora.

Coste para 300 horas: 12,5 €/hora x 300 horas = 3.750 €

Total capital humano: 3.750 €

En la Figura 3.5 se muestra una tabla con el resumen del presupuesto potencial total del proyecto, en términos aproximados.

Presupuesto total del proyecto		
Categoría	Recurso	Coste (€)
Hardware	Ordenador portátil	120
	Máquina virtual	375
	Total	495
Software	Overleaf	59,25
	Visual Paradigm	54
	Total	113,25
Capital humano	Desarrollador software	3.750
Total		4.358,25

Tabla 3.5: Presupuesto del proyecto

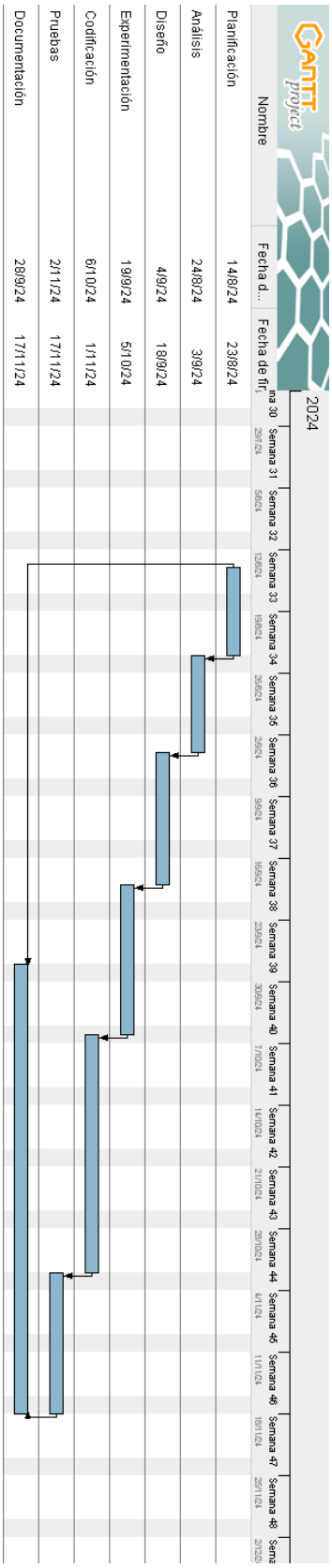


Figura 3.1: Diagrama de Gantt sobre la planificación temporal del proyecto

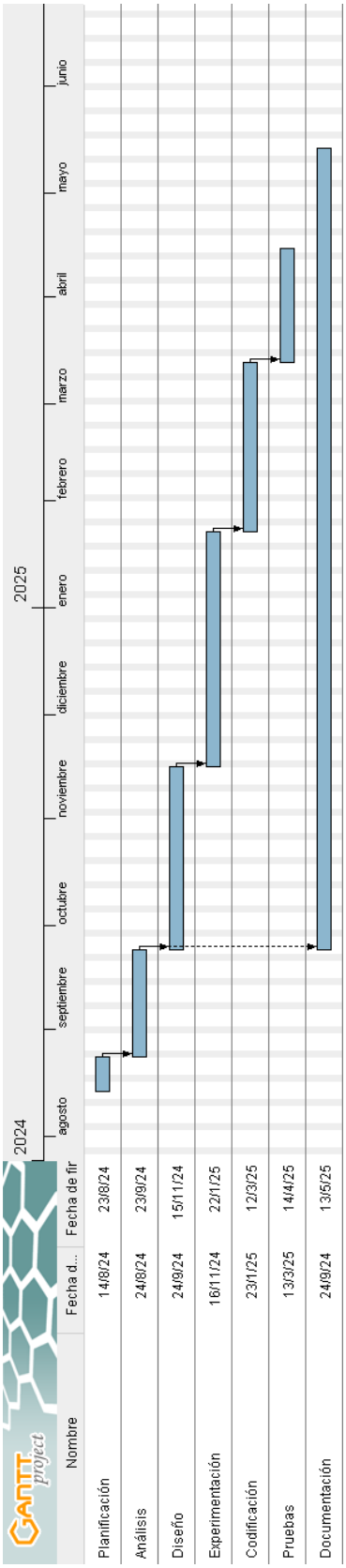


Figura 3.2: Diagrama de Gantt sobre la planificación temporal del proyecto

Capítulo 4

Análisis

En esta fase del desarrollo se detallan los objetivos y necesidades del proyecto, especificando las funcionalidades que presentará y los componentes por los que estará formado en forma de requisitos.

Los requisitos se han obtenido a partir de entrevistas con representantes del público objetivo del proyecto, siendo estos profesores de colegio que serían los que usasen el sistema si fuese instalado.

4.1. Identificación de usuarios

Habría dos usuarios de la aplicación, el que la usa realizando preguntas al sistema a través de un navegador web, y el encargado de instalarla en un servidor, mantenerla y poblar la base de datos, que sería el administrador del sistema.

- **Usuario/Cliente:** Estudiante o profesor que accede a la página web del servicio y realiza alguna pregunta sobre la información de la base de datos. No es necesario que tenga conocimientos técnicos de ningún tipo, ya que el servicio se ofrece a estudiantes y profesores de cualquier ámbito, técnico o no.
- **Administrador:** técnico de la organización que instala el sistema. No es necesario que conozca el funcionamiento completo del sistema debido a su naturaleza modular, pero debe saber por qué componentes está formado y cómo se comunican entre sí para poder diagnosticar el problema. También puede conocer alguna técnica simple de procesado de datos para mejorar el funcionamiento de la aplicación.

4.2. Requisitos

A continuación se presentan los requisitos del proyectos que se obtuvieron a partir de los objetivos de los clientes.

4.2.1. Requisitos funcionales

Los requisitos funcionales definen la funcionalidad y características esperadas por los usuarios, es decir, cómo se comporta el sistema desde el punto de vista de los usuarios.

- RF-1: El sistema deberá permitir al usuario realizar cualquier pregunta escrita.
- RF-2: El sistema deberá responder la pregunta del usuario correctamente.
- RF-3: El sistema deberá permitir al usuario personalizar el tipo de respuesta.
- RF-4: El sistema deberá indicar al usuario la o las fuentes que ha usado, y el fragmento o fragmentos que referencia en la respuesta.
- RF-5: El sistema deberá mostrar el historial de mensajes realizados durante la sesión al usuario.
- RF-6: El sistema deberá facilitar al administrador añadir documentos a su librería de conocimiento.

4.2.2. Requisitos no funcionales

Estos requisitos indican cómo se deben alcanzar los requisitos funcionales dictando características y restricciones del sistema.

- RNF-1: El sistema deberá presentar una GUI al usuario a través de una página Web.
- RNF-2: El cliente debe poder acceder al sistema de forma remota al sistema.
- RNF-3: El cliente solo podrá interactuar con la aplicación mediante un navegador Web.
- RNF-4: La comunicación entre el servidor Web y el servicio Ollama se realizará mediante el protocolo HTTP.
- RNF-5: La comunicación entre el servidor Web y la base de datos vectorial se realizará mediante el protocolo HTTP
- RNF-6: El sistema deberá mostrar el resultado de las preguntas del cliente a través de una página web.
- RNF-8: El sistema solo mantendrá el historial de conversación durante la sesión con el cliente.
- RNF-9: El sistema solo responderá preguntas cuyas respuesta pueda encontrar en su librería de documentos.

- RNF-10: El sistema deberá informar al cliente de cualquier fallo que se haya producido durante la resolución de una pregunta a través de la GUI Web.
- RNF-11: si un error puntual ocurre durante la ejecución, el sistema deberá volver a un estado neutral que permita volver a recibir preguntas.
- RNF-12: El sistema no podrá ser accesible si Ollama o la base de datos se desconectan de forma definitiva, comunicándoselo así a cualquier cliente que intente acceder durante el periodo de desconexión.
- RNF-13: El administrador deberá poder configurar el sistema exclusivamente cambiando valores de un archivo de configuración externo al programa.

4.3. Casos de uso

Los casos de uso describen cada uno de los posibles flujos de operación del sistema por los usuarios, definiendo cómo usarán (o se espera que usen) el sistema y cómo responderá este.

4.3.1. Diagramas de casos de uso

En la Figura 4.1 se muestra el diagrama de casos de uso para los usuarios.

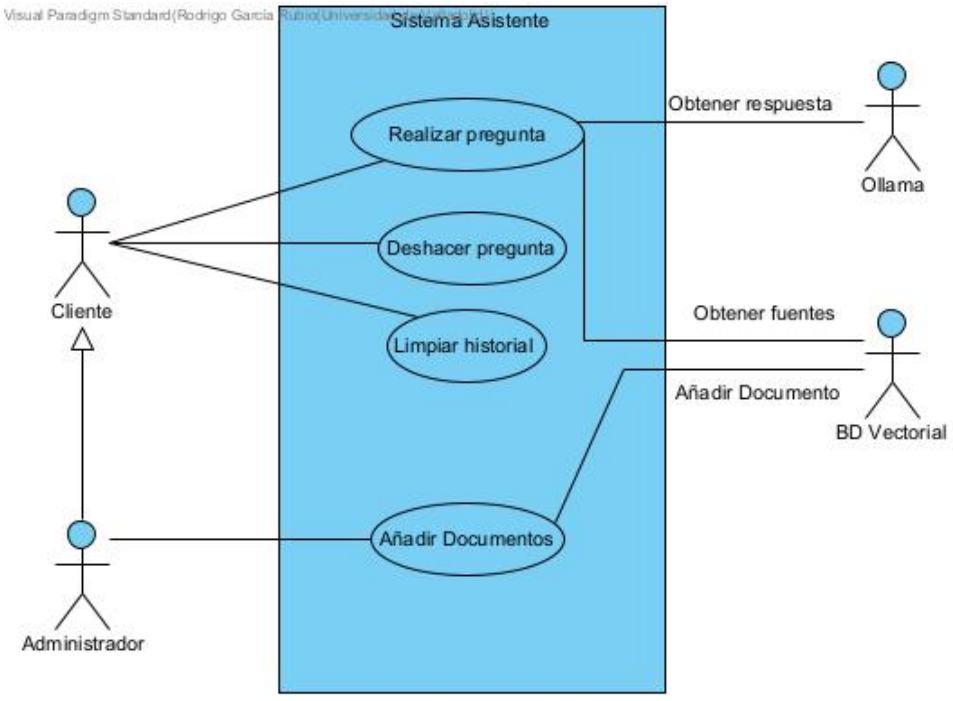


Figura 4.1: Diagrama de casos de uso

4.3.2. Especificación de casos de uso

En la especificación de los casos de uso se define la secuencia principal de interacciones entre el usuario y el sistema y las posibles alternativas que puedan presentarse durante el flujo. También se indican condiciones iniciales para que pueda comenzar un caso de uso, y las condiciones que se deben dar una vez haya terminado para que se considere finalizado.

Cabe mencionar que el administrador tendrá acceso a partes del sistema inaccesibles para los clientes, como el hardware donde se ejecuta o los archivos de instalación de la aplicación, por lo que su forma de interacción con el sistema será bastante distinta a la de los usuarios normales.

CU-1	**Realizar pregunta**
Actor	Cliente
Descripción	Realiza una pregunta al sistema RAG-LLM de la aplicación
Precondición	El sistema se comunica correctamente con Ollama y la base de datos, que debe estar poblada con documentos, y la interfaz está activa.
Secuencia normal	1. El Cliente introduce su pregunta y la envía. 2. El sistema pasa la pregunta y el historial de mensajes anteriores a Langchain, que lo comunica a Ollama y a la Base de Datos 3. El sistema recibe la respuesta de Langchain y la muestra al Cliente.
Postcondición	La nueva respuesta es correcta y se muestra a continuación de la lista de mensajes anteriores
Excepciones	
Variaciones	**Acción**
1a	El Cliente elige reintentar la pregunta anterior 1a.1 El sistema elimina la pregunta y respuesta anteriores del historial y las oculta del Cliente 1a.2 Ir al paso 2
2a	El sistema falla al resolver la pregunta pero sigue funcionando 2a.1 El sistema avisa al Cliente 2a.2 Ir al paso 1
2b	El sistema falla al resolver la pregunta y es incapaz de volver a funcionar 2b.1 El sistema avisa al Cliente e impide introducir nuevas preguntas 2b.2 Fin de sistema

Tabla 4.1: Caso de uso 1: Realizar pregunta

CU-2	**Añadir documentos**
Actor	Administrador
Descripción	Añadir documentos a la base de conocimiento del sistema RAG
Precondición	La Base de datos está activa y ejecutándose.
Secuencia normal	1. El Administrador inicia el script para añadir documentos a la Base de Datos. 2. El sistema se conecta a la Base de Datos. 3. El sistema pide al Administrador documentos para introducir. 4. El Administrador introduce los documentos. 5. El sistema procesa los documentos y los introduce en la Base de Datos. 6. El sistema informa al Administrador del éxito de la operación y termina el script.
Postcondición	La Base de datos acaba poblada con los documentos provistos por el Administrador.
Excepciones	
Variaciones	**Acción**
2a	El sistema no consigue conectarse con la Base de Datos 2a.1 El sistema informa al Administrador del error y ofrece reintentar la conexión 2a.2 Ir al paso 2
4a	El Administrador cancela la operación 4a.1 Termina el script y termina el escenario
4b	Algunos o todos los documentos introducidos por el Administrador no pertenecen a uno de los tipos de documentos permitidos 4b.1 El sistema avisa al Administrador del error y le recuerda los tipos de documentos aceptables 4b.2 Ir al paso 3
5a	El sistema falla al introducir los documentos en la Base de Datos 5a.1 El sistema informa al Administrador del error y de los documentos que consiguió introducir en la Base de datos antes del error 5a.2 Ir al paso 3

Tabla 4.2: Caso de uso 2: Añadir documento

CU-3	Deshacer pregunta
Actor	Cliente
Descripción	El Cliente deshace la última pregunta realizada y la elimina del historial.
Precondición	El componente de interfaz del sistema está activo.
Secuencia normal	1. El Cliente deshacer la última pregunta. 2. El sistema elimina la pregunta y respuesta del historial. 3. El sistema elimina la pregunta y respuesta de la interfaz.
Postcondición	Si al menos había una pregunta y respuesta en el historial, entonces la pareja pregunta-respuesta más reciente ha sido borrada.

Tabla 4.3: Caso de uso 3: Deshacer pregunta

CU-4	Limpiar historial
Actor	Cliente
Descripción	El cliente elimina el historial de preguntas y respuestas de la sesión actual manualmente
Precondición	El componente de interfaz del sistema está activo.
Secuencia normal	1. El Cliente limpia el historial de preguntas. 2. El sistema vacía el historial de preguntas y respuestas. 3. El sistema elimina todas la preguntas y respuestas de la interfaz.
Postcondición	El historial de preguntas y respuestas del Cliente está vacío.

Tabla 4.4: Caso de uso 4: Limpiar historial

4.3.3. Diagrama de paquetes

La Figura A.2 muestra la organización lógica de la aplicación en paquetes. Esta separa el programa principal, formado por la aplicación web y el servidor, de las librerías y servicios externos, como Ollama y Chroma, y del código de soporte. Muestra, además, las relaciones entre los paquetes, indicando las dependencias entre ellos.

4.4. Diagramas de actividades

En esta sección se muestra el flujo de acciones tomadas en los casos de usos indicados previamente.

4.4.1. CU-1: Realizar Pregunta

Este caso de uso se ilustra en la Figura A.4.

4.4.2. CU-2: Añadir Documentos

Este caso de uso se ilustra en la Figura 4.2.

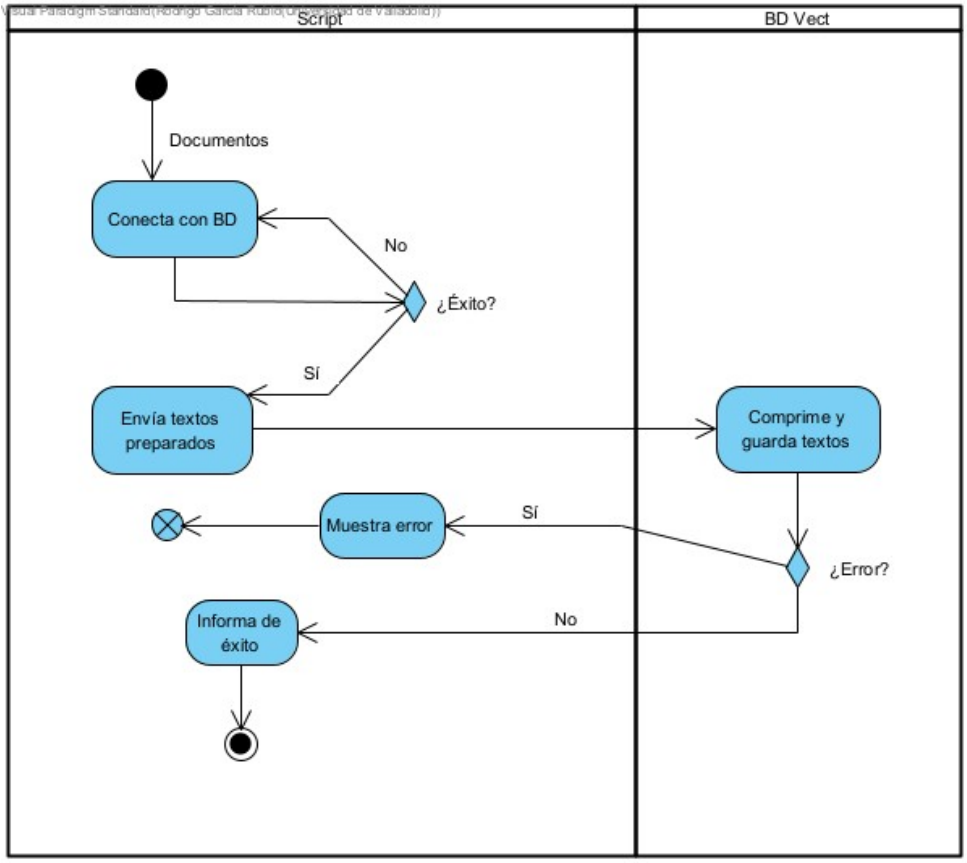


Figura 4.2: Diagrama de actividades del CU-2: Añadir documentos a la base de datos

4.4.3. CU-3: Deshacer pregunta

Este caso de uso se ilustra en la Figura 4.3.

4.4.4. CU-4: Limpiar historial

Este caso de uso se ilustra en la Figura 4.4.

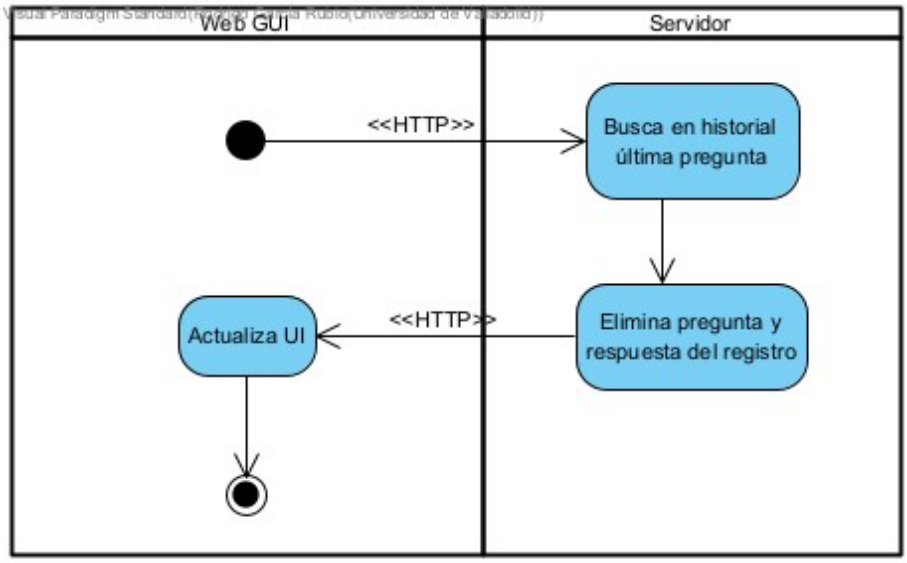


Figura 4.3: Diagrama de actividades del CU-3: Deshacer una pregunta

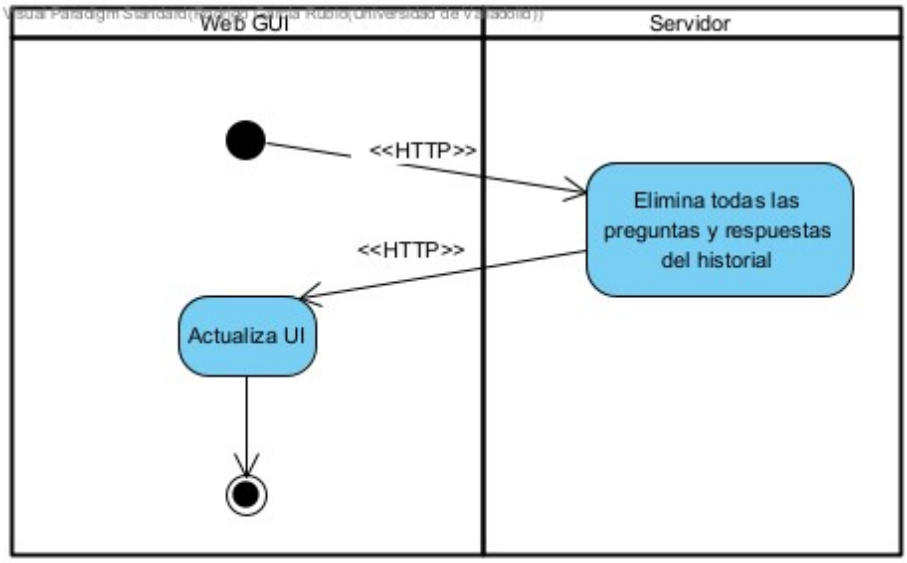


Figura 4.4: Diagrama de actividades del CU-4: Limpiar el historial de preguntas

Capítulo 5

Diseño

En esta sección se especifica el funcionamiento del sistema y el comportamiento de sus componentes. Sin embargo, debido a que la aplicación se plantea como varios componentes externos unidos para dar el servicio deseado, este capítulo se centrará en explicar las partes desarrolladas en este proyecto. El funcionamiento individual de los componentes será explicado en profundidad en la sección de Implementación.

Además, para encontrar la mejor configuración de los componentes se realizarán varias pruebas modificando los parámetros y el diseño de la aplicación. Por ello, se indicarán también los diseños alternativos de la aplicación y las distintas formas en las que se conectan los componentes externos.

Dejando de lado los componentes externos, la aplicación desarrollada se divide en dos funcionalidades principales: el procesamiento de la entrada del usuario y la preparación y gestión de documentos en la base de datos vectorial. El procesamiento y respuesta de las preguntas del usuario se lleva a cabo por la aplicación principal, recibiendo la entrada por la aplicación web del usuario y generando la respuesta en el servidor, usando la base de datos para almacenar el conocimiento y el modelo de lenguaje para crear una respuesta coherente.

5.1. Arquitectura del sistema

La arquitectura básica de la aplicación se basaría en un sistema cliente-servidor en el que el cliente ejecuta una aplicación web en su navegador que se conecta al servidor, donde se ejecutan las tareas pesadas, como el LLM y la base de datos usando hardware más potente. Este puede estar lejos del cliente y permitiría un uso entre varios usuarios de forma simultánea.

Esta arquitectura básica se representa en el diagrama de componentes UML de la Figura 5.1, aunque debido a la naturaleza compartimentada del programa también sería posible separar la ejecución los modelos LLM y la base de datos en otros servidores distintos del principal sin necesidad de cambios.

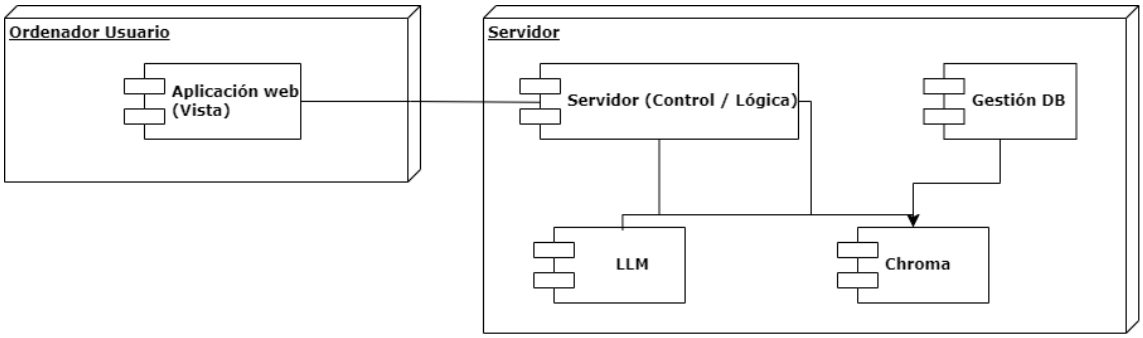


Figura 5.1: Arquitectura básica de la aplicación

Para explicar mejor el concepto de la arquitectura RAG, también se incluye un diagrama simplificado sobre su preparación previa y funcionamiento durante la ejecución (Figura A.5). Todo el sistema RAG reside en el lado del servidor y su implementación será el foco de la experimentación, modificando partes del procesamiento o búsqueda afectando lo menos posible a la arquitectura general del servicio. Estos cambios se verán reflejados tanto en la aplicación principal como en el programa de gestión de la base de datos.

5.2. Estructura de clases

El programa se organizará en las clases que se pueden observar en la Figura A.6.

Como se puede observar, la aplicación se centra en la clase Servidor, que mantiene acceso al modelo LLM y a la base de datos, y contiene los métodos con la funcionalidad principal.

La clase Langhchain, de la librería homónima, es usada por la clase Servidor para simplificar el acceso al LLM y a la BD, reduciéndolo a un solo método.

Por otro lado, la clase Gradio genera la aplicación web y la interfaz y llama a los métodos del Servidor según reciba instrucciones del usuario mediante la interfaz.

La clase Gestor BD, como es una herramienta para administradores, no tiene relación con Gradio ni Langchain y accede directamente a la base de datos para manipularla.

5.3. Interfaz de usuario

La Interfaz del usuario será gestionada por el módulo Gradio. Este componente realiza varias funciones en el sistema, una de las cuales es generar el lado del cliente de la aplicación, la página web, a partir de una estructura base definida en una clase Python, usando los elementos de interfaz incluidos en su librería como métodos cuyos parámetros se pueden editar.

Esta aplicación web funcionará de interfaz completa para el usuario, y también incluye una API HTTP para usar el servicio mediante un programa en vez de mediante la interfaz. Este servicio API se puede gestionar y restringir como se vea necesario de forma independiente de la interfaz gráfica. Como Gradio también gestiona el servidor Python y sus interfaces, genera dicha API automáticamente a partir de los elementos incluidos en la interfaz en el servidor y en la aplicación, de forma que se conectan de forma transparente con el resto del servicio.

Lo único que se decide entonces son los elementos de la interfaz, cómo se manejarán sus datos y otros comportamientos adicionales de la interfaz editando los parámetros de los métodos que los representan.

Esta interfaz deberá disponer, según los requisitos de funcionalidad, de:

- Elemento para mostrar respuestas y mensajes pasados.
- Elemento para introducir un nuevo mensaje.
- Elemento para cambiar el tipo de procesamiento y respuestas.

El componente que muestre las respuestas deberá hacerlo desde la primera pregunta hasta la más reciente, y el elemento que acepta las nuevas preguntas deberá permitir que se escriba directamente en la aplicación y una forma simple para indicar que lo envíe.

5.4. Servidor

Las entradas de los usuarios se gestionan en un servidor central de forma secuencial. Esto es debido a que procesar una entrada requiere ejecutar un modelo de LLM, lo cual, debido a limitaciones de memoria y computación del ordenador usado para el desarrollo, limita la aplicación a procesar una pregunta cada vez. Añadir gestión asíncrona de peticiones solo incrementaría la complejidad sin aportar ningún beneficio, ya que el cuello de botella creado por el LLM seguiría estando ahí. Además, el servidor incluido en el paquete Gradio usa un sistema de gestión de cola de peticiones, de forma que los usuarios podrán seguir usando el sistema y enviar sus peticiones a la cola.

En un equipo más potente, como un servidor profesional con mayor capacidad de memoria y cómputo, capaz de ejecutar varios modelos de forma concurrente, se podría añadir el procesamiento asíncrono de peticiones.

Este servidor está formado por una única clase "Servidor" con varios métodos que procesan los datos recibidos de la aplicación web del cliente como una *pipeline*. La base de datos vectorial y el LLM son componentes de esta pipeline, accedidos mediante Langchain.

Para ver cómo se comporta el servidor durante el procesamiento normal de una pregunta del usuario, se ha añadido un diagrama de secuencia en la Figura A.3 que representa el caso de uso de Realizar pregunta (Tabla 4.1).

5.5. Gestor de la Base de Datos

La gestión de la base de conocimiento la realizará una aplicación separada e independiente de la aplicación web, desarrollada por mi mismo, que contiene los servicios principales. Este programa distinto permitirá introducir, consultar, modificar y eliminar documentos de la base de datos vectorial de forma remota o local. También permitirá al administrador crear nuevas colecciones (conjunto de documentos y sus vectores) en la BD con distintas configuraciones y gestionarlas también.

Consistirá en una clase principal con un método responsable para cada funcionalidad del programa. Uno de ellos será el responsable del procesamiento y preparación de los documentos para añadirlos a la BD, incluyendo todas las transformaciones necesarias para mejorar la búsqueda de forma automática y uniforme.

Esta aplicación se usará a través de la terminal para mejorar el rendimiento y la velocidad de desarrollo, ya que solo será usada por el administrador del sistema, un usuario experto, y no por un público general.

Esta herramienta es necesaria para gestionar la base de datos por dos motivos: los documentos requieren varias etapas de procesamiento antes de poder poblar la base de datos, que esta herramienta aplica automáticamente, y manejarla manualmente forzaría a los administradores a aprender varios comandos nuevos y usarlos manualmente, lo que la hace más difícil de usar y puede dar lugar a errores, ya que este tipo de BD (BD vectoriales) no se parecen a las BD relacionales más comunes.

Capítulo 6

Entorno de experimentación

En este capítulo se explicarán los métodos usados para probar la eficacia del sistema bajo diferentes configuraciones y estructuras, tanto respecto a los elementos bajo escrutinio que se han alterado en cada iteración, como a las herramientas y procesos empleados para evaluarlos y obtener los datos necesarios para compararlos y extraer conclusiones.

6.1. Herramientas

Para la experimentación se usará mi ordenador personal, un ordenador portátil equipado con una tarjeta gráfica dedicada. Las pruebas además se ejecutarán a través de una máquina virtual local sobre dicho ordenador para minimizar interferencias externas y simular un entorno nuevo en cada prueba.

Las características del ordenador son las siguientes:

- Intel i7-10750H @ 2.60 | 6 Núcleos
- 16 GB memoria RAM DDR4
- Nvidia GeForce GTX 1650 Ti (Capacidad de Cómputo 7.5)

Los documentos usados en las pruebas serán varios documentos sobre leyes educativas, como boletines oficiales de Castilla y León, debido a su disponibilidad y a su naturaleza, que se ajusta bien al uso de la aplicación por su dificultad de lectura, longitud y formato estructurado. Los documentos usados, concretamente, son:

- Decreto infantil 37/2022 [3]
- Decreto primaria 38/2022 [4]
- Decreto secundaria 39/2022 [5]
- Decreto primaria EDU/423/2024 [6]

No se usarán demasiados en este proyecto para acelerar el proceso de experimentación y desarrollo, pero suficientes y con suficiente variedad como para simular un sistema real que contenga todos los documentos relevantes.

6.2. Ragas

Otro elemento imprescindible para la experimentación será la librería *Ragas* [25]. Esta permite evaluar la calidad de las respuestas de sistemas LLM de acuerdo a varios parámetros, tales como adecuación y precisión del contexto extraído, la sensibilidad al ruido del documento o la relevancia de la respuesta final a la pregunta inicial.

Ragas permite evaluar tanto la calidad de interacciones aisladas (pregunta-respuesta) así como de conversaciones con varias respuestas concatenadas, pero para este sistema nos centraremos en las consultas individuales para mayor eficiencia. Cada uno de estos experimentos consiste en una pregunta del usuario, el contexto que se ha encontrado y extraído de la Base de Conocimiento, la respuesta generada por el LLM a partir del contexto y la respuesta final correcta. Algunos de estos ejemplos se crearán a mano para tener un conjunto de casos reales con los que probar el sistema y otros serán generados por RAGAS para aumentar el tamaño del conjunto de pruebas mediante un sistema aparte basado en grafos de conocimiento [24].

El sistema que usa es bastante complejo y consiste, de forma resumida, en dividir el documento en fragmentos, extraer datos y características de cada uno y relacionarlos en nodos, creando un grafo de conocimiento como el de la Figura A.7. Esto permite crear experimentos concretos para simular ciertas especificaciones de respuestas o tipos de usuarios, a partir de los cuales se generan los pares (pregunta-respuesta) finales, listos para probar el sistema real. Cada prueba usará 5 de estos ejemplos sintéticos, además de los ejemplos reales que serán al menos 5.

6.3. Diseño de experimentos

Se han diseñado diferentes experimentos según el componente que se modifique en cada uno. En cada sección se comparan varias posibles opciones para dicho componente según su rendimiento respecto a una configuración "base".

Esta configuración "base" consiste en las siguientes opciones, que serán explicadas en cada uno de los experimentos:

- Procesamiento de documentos: Documentos originales
- Estrategia de fragmentación y almacenamiento: Directa
- Preprocesamiento de preguntas: Ninguna
- Monitorización y mejora de resultados: Ninguna

- Modelo de generación de respuesta: llama3.1:8b-instruct-q4_K_M
- Fragmentador: RecursiveCharacterSplitter
- Modelo de vectorización: all-MiniLM-L6-v2
- Algoritmo de búsqueda de documentos: distancia l2 al cuadrado

Para elegir cuál de las opciones para un componente es la más adecuada, se comparará el rendimiento del sistema "base" descrito iterando sobre las distintas alternativas para dicho componente. Es decir, en cada sección se comparará el rendimiento de varios sistemas "base" idénticos excepto por el componente a probar para observar sus diferencias.

El rendimiento se medirá usando cuatro métricas:

- Tiempo de ejecución: latencia entre el envío de una pregunta y la generación de su respuesta.
- Recuperación de contexto: porcentaje de información relevante de la base de conocimiento extraída exitosamente en el contexto [22].

  Para obtener esta medida se extraen las afirmaciones individuales de las respuestas de referencia, es decir, las afirmaciones que responden a la pregunta, y se comprueba el porcentaje de estas que se puede encontrar en contexto extraído en la búsqueda. Por ejemplo, un 1.0 significaría que toda la información necesaria para responder la pregunta se puede encontrar en el contexto recuperado. La fórmula sería la siguiente:

$$\text{Recuperación del contexto} = \frac{\text{Número de afirmaciones de la referencia en el contexto}}{\text{Número de afirmaciones total en referencia}}$$

- Fidelidad: porcentaje de información del contexto contenida en la respuesta final [23].

  Para calcular esta medida:

  1. Identifica todas las afirmaciones en la respuesta.
  2. Verifica cada afirmación para ver si se puede inferir del contexto recuperado.
  3. Calcula la puntuación de fidelidad utilizando la fórmula:

$$\text{Fidelidad} = \frac{\text{Número de afirmaciones de la respuesta en el contexto}}{\text{Número de afirmaciones total en respuesta}}$$

- Relevancia: porcentaje de información relevante extraída de la base de conocimiento contenida en el mensaje final [26].

  El método para obtener esta medida es el siguiente:

1. Genera un conjunto de preguntas artificiales (por defecto 3) basadas en la respuesta usando un modelo LLM. Estas preguntas están diseñadas para reflejar el contenido de la respuesta.
2. Calcula la similitud del coseno entre el *embedding* (representación vectorial numérica de un texto) de la entrada del usuario (E_0) y el de cada pregunta generada (E_{g_i}).
3. Toma el promedio de estas puntuaciones de similitud del coseno para obtener la Relevancia de la Respuesta.

$$\text{Relevancia} = \frac{1}{N}\sum_{i=1}^{N} \text{similitud coseno}(E_0, E_{g_i})$$

Se han usado estas métricas ya que permiten medir de forma independiente cada uno de los aspectos del sistema RAG:

- La recuperación de la información importante de la base de datos.
- La extracción de la información del contexto a la respuesta.
- La relevancia y calidad general de la respuesta final.

Experimento 1: Modelo de vectorización de documentos

Para obtener los *embeddings* de los documentos para realizar las búsquedas se necesita un modelo de vectorización. Este proceso obtiene un vector de números reales que representará la información contenida en el fragmento de forma que conceptos similares tengan valores cercanos en el espacio de valores del vector, por lo que fragmentos que hablan del mismo tema generarán vectores parecidos.

Los modelos de vectorización que se usarán en este proyecto proceden del proyecto Sentence Transformers (SBERT) [18]. Se probará el rendimiento de los siguientes modelos de vectorización para comprobar cual es el más adecuado:

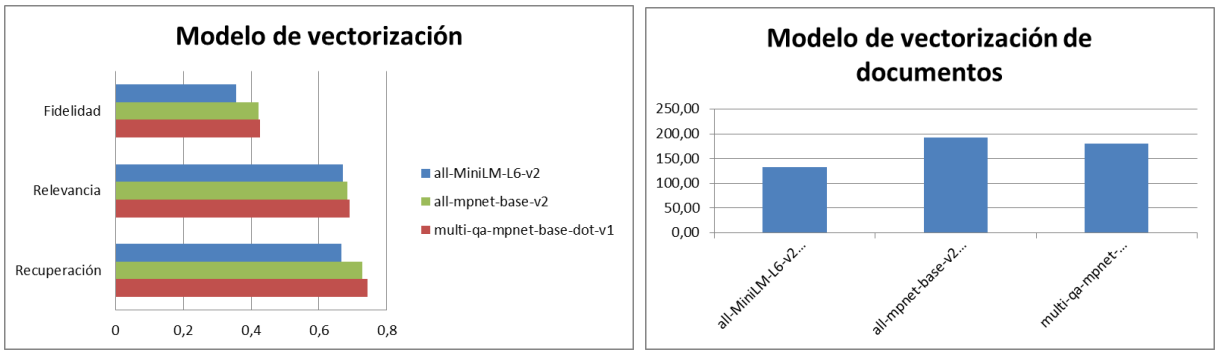
- all-MiniLM-L6-v2: el modelo más eficiente y de propósito general ofrecido en SBERT. Buen rendimiento y velocidad para el tamaño (80MB).
- all-mpnet-base-v2: el modelo más grande que ofrecen (420 MB) y el que presenta el mejor rendimiento general.
- multi-qa-mpnet-base-dot-v1: modelo especializado en búsqueda semántica, preparado para situaciones de pregunta-respuesta. Mismo tamaño que el anterior (420 MB), pero con peor rendimiento general exceptuando en las búsquedas.

Usando estos modelos se crearán tres colecciones distintas, cada una con vectores *embedding* generados usando cada uno de los modelos, y se comparará el rendimiento de la aplicación con cada una de las colecciones. Las tres tienen los mismos documentos y el resto de componentes de la aplicación son iguales.

Los resultados se pueden observar en la Tabla 6.1 y la Figura 6.1.

Componente	Recuperación	Relevancia	Fidelidad	Tiempo (s)
all-MiniLM-L6-v2	0,667	0,671	0,356	133,00
all-mpnet-base-v2	0,728	0,684	0,423	192,38
multi-qa-mpnet-base-dot-v1	0,743	0,69	0,427	180,40

Tabla 6.1: Modelos de vectorización de documentos



(a) Medidas de Modelos de vectorización de documentos

(b) Tiempos de Modelos de vectorización de documentos

Figura 6.1: Modelos de vectorización de documentos

Como se puede observar, el modelo *multi-qa-mpnet-base-dot-v1* ha mostrado el mejor rendimiento de los tres modelos usados, al presentar mejores métricas de calidad de respuesta y acelerar la búsqueda respecto al segundo mejor *all-mpnet-base-v2*.

Experimento 2: Preprocesamiento de documentos

Antes de añadir los documentos a la base de conocimiento, algunos tipos de documentos pueden tener varios elementos superfluos que no añaden información, como cabeceras, títulos o índices, así como elementos difíciles de procesar, como tablas o imágenes.

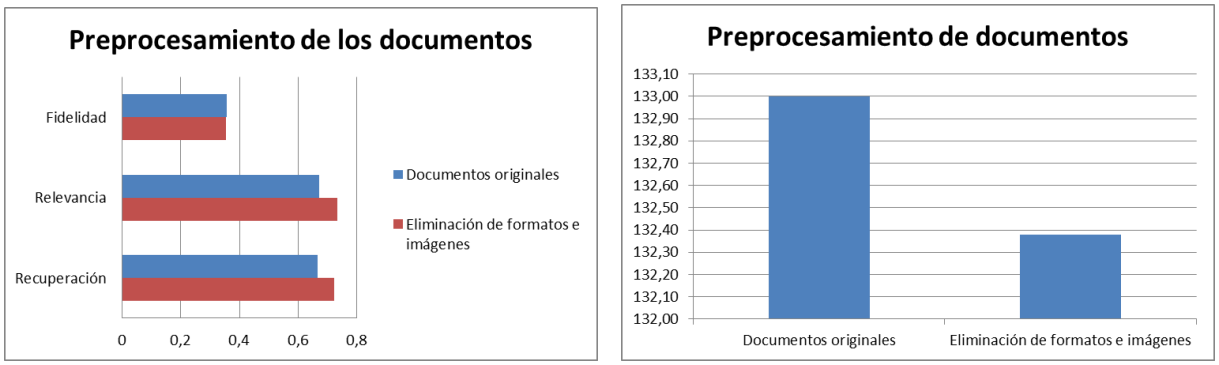
Estos elementos pueden afectar el rendimiento del sistema RAG, bien reduciendo la calidad de los *embeddings* o afectando a la capacidad del LLM para entender el contexto y generar una respuesta.

Esta sección compara el rendimiento del sistema ”baseçon dos colecciones distintas: en una de ellas se han introducido los documentos sin ningún tipo de procesamiento y en la segunda se han eliminado todos los elementos innecesarios o difíciles de procesar.

Los resultados se pueden observar en la Tabla 6.2 y la Figura 6.2.

Componente	Recuperación	Relevancia	Fidelidad	Tiempo (s)
Documentos originales	0,667	0,671	0,356	133,00
Eliminación de formatos e imágenes	0,722	0,734	0,355	132,38

Tabla 6.2: Preprocesamiento de documentos



(a) Medidas de Preprocesamiento de documentos (b) Tiempos de Preprocesamiento de documentos

Figura 6.2: Preprocesamiento de documentos

Como se puede ver en las Figuras 6.2, al eliminar los elementos de formato innecesarios y otros elementos que el LLM no puede procesar correctamente la calidad de las respuestas mejora ligeramente y se reducen un poco los tiempos de búsqueda.

Experimento 3: Estrategia de fragmentación y almacenamiento de documentos

El método Directo para añadir los documentos a la base de datos consiste en dividirlos en varios fragmentos y vectorizarlos para obtener los *embeddings*. Después ambos se almacenan en la BD y los vectores de *embeddings* serán sobre los que se realizará la búsqueda y cada uno estará enlazado a su fragmento original, que será el que se devuelve como respuesta.

Otra opción más compleja es la fragmentación Padre-Hijo. Este método divide el documento original en fragmentos padre, como el método anterior pero algo más grandes, y después los divide cada uno en fragmentos hijo más pequeños. Estos serán los fragmentos de los que se obtendrán los *embeddings* y que se usarán en la búsqueda.

Otra diferencia es que no devuelven el fragmento hijo del que se obtuvo el *embedding*, sino el fragmento padre de la división original. De esta forma el LLM obtiene todo el contexto relevante, al tener fragmentos más grandes, sin perder precisión, al realizar la búsqueda sobre fragmentos más pequeños.

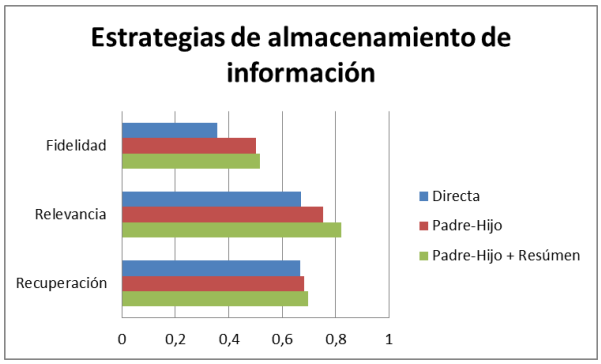
Una modificación al método anterior que requiere modificaciones mínimas del sistema consiste en añadir resúmenes generados automáticamente con una LLM de los fragmentos padre al espacio de búsqueda junto con los fragmentos hijo. Estos resúmenes condensan la información relevante de los fragmentos padre, por lo que pueden ser útiles en las búsquedas.

Para este experimento se usan tres colecciones distintas, una para cada alternativa, y las versiones de la aplicación utilizadas para compararlas solo serán distintas en el mecanismo de acceso a las colecciones para poder utilizarlas, ya que el sistema Padre-Hijo añade algo de complejidad.

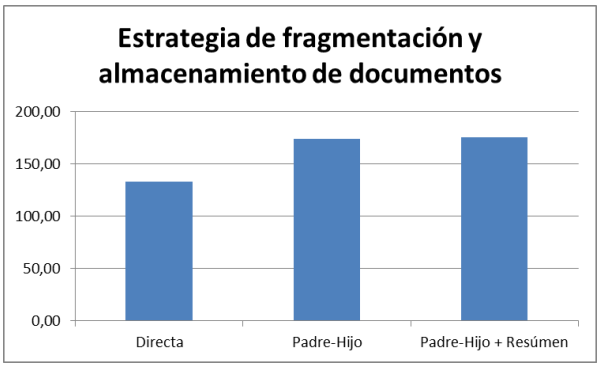
Los resultados se pueden observar en la Tabla 6.3 y la Figura 6.3.

Componente	**Recuperación**	**Relevancia**	**Fidelidad**	**Tiempo (s)**
Directa	0,667	0,671	0,356	133,00
Padre-Hijo	0,683	0,752	0,502	174,15
Padre-Hijo+Resumen	0,6983	0,8225	0,5171	175,42

Tabla 6.3: Estrategias de fragmentación y almacenamiento de documentos



(a) Medidas de Estrategias de fragmentación de documentos



(b) Tiempos de Estrategias de fragmentación de documentos

Figura 6.3: Estrategias de fragmentación y almacenamiento de documentos

Las Figuras 6.3 muestran que la estrategia de almacenamiento Padre-Hijo mejora sustancialmente la fidelidad y relevancia de las respuestas cambio de una latencia de respuesta algo peor. La mejora de la relevancia incrementa aún más al incorporar los resúmenes a la BD sin incrementar la latencia de respuesta.

Experimento 4: Preprocesamiento de preguntas

Las preguntas realizadas por el usuario se vectorizan y convierten en *embeddings* que se comparan con los fragmentos embebidos de la base de datos para obtener el contexto

relevante.

Se pueden realizar varias modificaciones sobre este sistema básico. Una de ellas es la generación de preguntas, que consiste en hacer que un LLM genere preguntas similares a la original, pero usando palabras o expresiones distintas. Esto cambiará el *embedding* resultante y permitirá encontrar contextos que no se habrían explorado con la pregunta original.

Para ilustrar el funcionamiento de la generación de preguntas se presenta el siguiente ejemplo: el usuario realiza la pregunta: "¿Cuál es la forma principal de garantizar que los estudiantes con discapacidad sea accesible a la educación?". El LLM encargado la recibe y, siguiendo las intrucciones del prompt de sistema, devuelve las siguientes preguntas en el formato correcto:

- ¿Qué tipo de tecnologías o recursos se utilizan para facilitar la inclusión de estudiantes con discapacidad?
- ¿Cómo se implementan medidas para brindar acceso igualitario en el sistema educativo?
- ¿Qué medidas específicas se han implementado en la comunidad educativa para promover la inclusión y el acceso igualitario a la educación?
- ¿Cómo se han diseñado programas y políticas que beneficien a los estudiantes con discapacidad?

Después se realizaría una búsqueda en la BD por cada una de las preguntas anteriores, incluyendo la original.

Como este sistema requeriría realizar varias búsquedas, dando lugar a un contexto recuperado mucho mayor, resulta conveniente usar también un sistema de filtrado de contextos por relevancia llamado Rerank. Este usa un modelo de vectorización específico llamado Cross-Encoder [7] que compara la relevancia de los contextos, los ordena y filtra los peores.

También se puede añadir un sistema de extracción de filtros a partir de la pregunta usando un modelo LLM. Estos filtros podrán usarse para reducir el espacio de búsqueda usando los metadatos de los fragmentos, y el resto de la pregunta se usará para buscar sobre los fragmentos restantes.

Este experimento requiere crear tres variantes de la aplicación para las tres alternativas del experimento, aparte de la versión "base". Además, también es necesario añadir metadatos manualmente a los documentos de la base de datos.

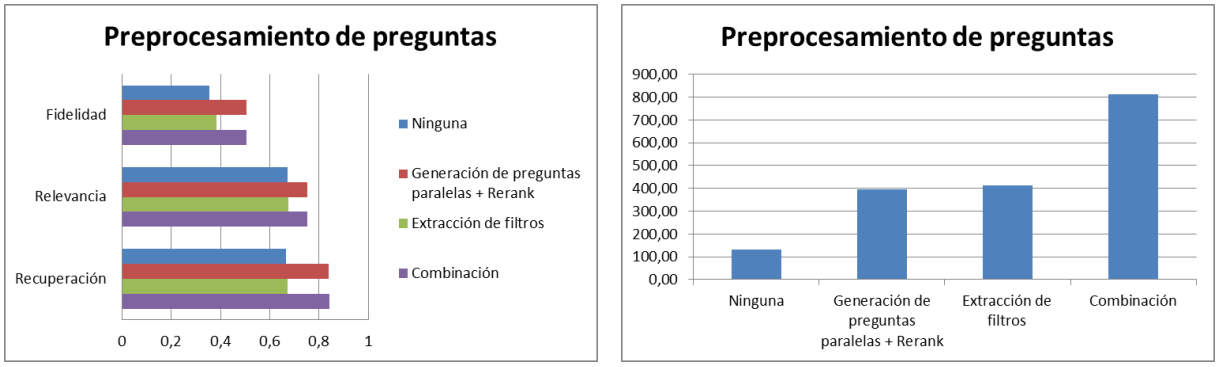
Por último, se comprobará el resultado de combinar ambas técnicas.

Los resultados se pueden observar en la Tabla 6.4 y la Figura 6.4.

Como se puede observar en las Figuras 6.4, tanto la generación de preguntas como la extracción de filtros aumentan bastante el tiempo de respuesta, aunque solo el primer

Componente	Recuperación	Relevancia	Fidelidad	Tiempo (s)
Ninguna	0,667	0,671	0,356	133,00
Generación preguntas+Rerank	0,84	0,752	0,505	394,81
Extracción de filtros	0,672	0,677	0,384	412,21
Generación+Filtros	0,843	0,753	0,504	811,02

Tabla 6.4: Preprocesamiento de preguntas



(a) Medidas de Preprocesamiento de preguntas (b) Tiempos de Preprocesamiento de preguntas

Figura 6.4: Preprocesamiento de preguntas

método mejoras importantes en fidelidad, relevancia y recuperación de contexto. El método combinado también presenta esta mejora generalizada de la calidad, pero aumenta mucho la latencia de respuesta.

Experimento 5: Monitorización y mejora de resultados

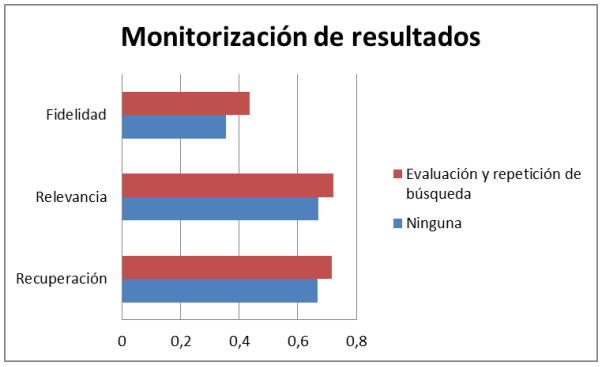
Por defecto, la primera respuesta generada se da por buena y se la muestra al usuario, pero también se puede usar un LLM para analizarla, comprobar su relevancia y calidad y decidir si se debería repetir el proceso desde el principio o si es aceptable para mostrar al usuario.

Para probar este sistema se crea una variante de la aplicación que lo incorpora sin cambiar ningún otro componente y usando la misma colección que la aplicación base.

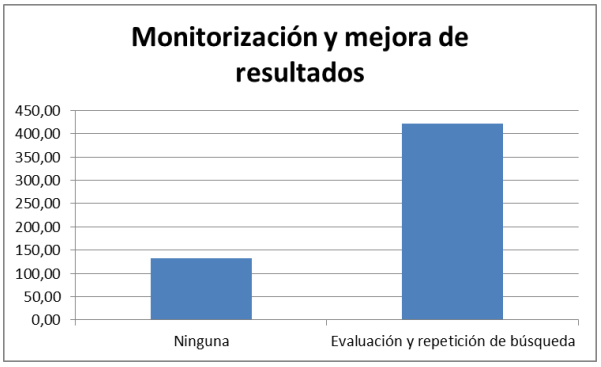
Los resultados se pueden observar en la Tabla 6.5 y la Figura 6.5.

Componente	Recuperación	Relevancia	Fidelidad	Tiempo (s)
Ninguna	0,667	0,671	0,356	133,00
Evaluación y repetición de búsqueda	0,715	0,722	0,436	421,59

Tabla 6.5: Monitorización y mejora de resultados



(a) Medidas de Monitorización y mejora de resultados



(b) Tiempos de Monitorización y mejora de resultados

Figura 6.5: Monitorización y mejora de resultados

En las Figuras 6.5 se puede ver como la mejora de la calidad de las respuestas obtenida mediante la evaluación y repetición de la búsqueda viene acompañada de un gran aumento del tiempo de espera para la pregunta.

Experimento 6: Modelo de generación de respuesta

Aunque existen modelos más grandes, potentes y mejores en general, por limitaciones técnicas de las máquinas de pruebas disponibles (ordenador portátil y máquina virtual) se probarán los siguientes tres modelos con tamaños distintos. Los tres son las versiones *instruct* de sus modelos originales, por lo que están especializados en conversaciones de texto y ha sido cuantizados para reducir su tamaño y latencia respecto al modelo original.

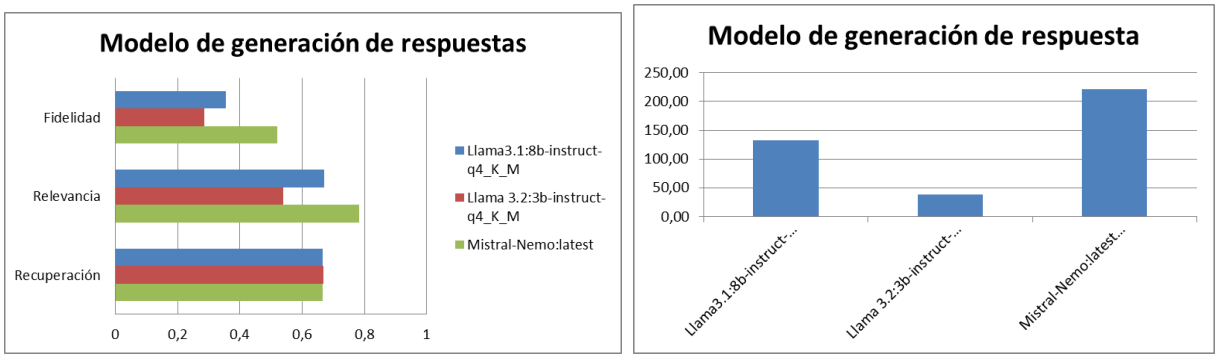
- Llama3.1:8b-instruct-q4_K_M: modelo relativamente pequeño, pero eficiente. De los mejores con su tamaño con 8 mil millones de parámetros.
- Llama 3.2:3b-instruct-q4_K un modelo que trata de comprimir la eficacia del anterior en un paquete más pequeño, que con solo 3 mil millones de parámetros es comparable con otros modelos de mayor tamaño. Es la versión *instruct*, por lo que está especializado en conversaciones de texto, y ha sido cuantizado para reducir su tamaño y latencia respecto al modelo original.
- Mistral-Nemo:12b-instruct-2407-q4_K_M: modelo basado en el original Mistral 7B, creado con colaboración con Nvidia y el más grande que se probará en este proyecto con 12 mil millones de parámetros. Es la versión *instruct*, por lo que está especializado en conversaciones de texto, y ha sido cuantizado para reducir su tamaño y latencia respecto al modelo original.

Para comparar los modelos solo es necesario ejecutar la aplicación cambiando el modelo usado, lo que no requiere modificaciones del programa ni de la colección.

Los resultados se pueden observar en la Tabla 6.6 y la Figura 6.6.

Componente	Recuperación	Relevancia	Fidelidad	Tiempo (s)
Llama3.1:8b-instruct-q4_K_M	0,667	0,671	0,356	133,00
Llama 3.2:3b-instruct-q4_K_M	0,668	0,539	0,285	38,69
Mistral-Nemo:latest	0,665	0,782	0,519	221,71

Tabla 6.6: Modelos de generación de respuesta



(a) Medidas de Modelos de generación de respuesta (b) Tiempos de Modelos de generación de respuesta

Figura 6.6: Modelos de generación de respuesta

Experimento 7: Fragmentador de documentos

Hay varias formas de dividir los documentos según el criterio usado y la herramienta utilizada:

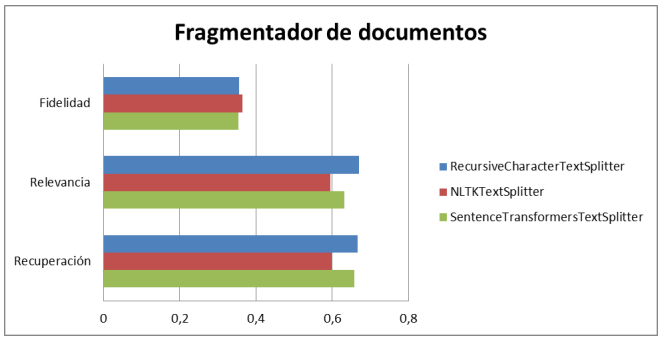
- RecursiveCharacterSplitter: divide los documentos según el número de caracteres que debe contener cada fragmento. Respeta ciertos caracteres, como saltos o fines de línea, manteniendo líneas y párrafos de texto intactos para conservar su significado y coherencia.
- NLTKTextSplitter: estima la longitud del documento, no en caracteres, sino en tokens reconocidos por un LLM , y lo divide según esta cuenta aproximada. Para esto, usa un modelo de tokenización de la librería NLTK.
- SentenceTransformers: estima la longitud del documento, no en caracteres, sino en tokens reconocidos por un LLM , y lo divide según esta cuenta aproximada. Para esto, usa un modelo de tokenización de la librería SentenceTransformers.

Para este experimento es necesario crear tres colecciones distintas de documentos usando los mismos parámetros excepto el fragmentador. Estas colecciones se utilizarán con la misma aplicación para observar sus diferencias en la calidad de las respuestas finales.

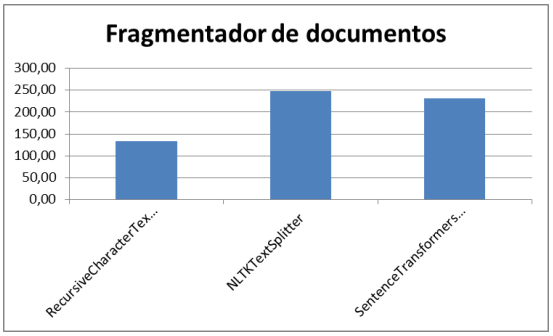
Los resultados se pueden observar en la Tabla 6.7 y la Figura 6.7.

Componente	Recuperación	Relevancia	Fidelidad	Tiempo (s)
RecursiveCharacterTextSplitter	0,667	0,671	0,356	133,00
NLTKTextSplitter	0,599	0,595	0,364	247,39
SentenceTransformersTextSplitter	0,658	0,632	0,355	231,84

Tabla 6.7: Fragmentadores de documentos



(a) Medidas de Fragmentadores de documentos



(b) Tiempos de Fragmentadores de documentos

Figura 6.7: Fragmentadores de documentos

Como se puede ver en las Figuras 6.7, RecursiveCharacterTextSplitter y SentenceTransformersTextSplitter alcanzan métricas de calidad muy similares y ambos mejores que NLTKTextSplitter. Sin embargo, el primer modelo consigue un menor tiempo de respuesta que los otros dos.

Experimento 8: Algoritmo de búsqueda por similitud de contextos

La búsqueda del contexto se realiza comparando la distancia entre el *embedding* de la pregunta y los *embeddings* de los fragmentos de la base de datos. Esta distancia se puede calcular usando varias fórmulas:

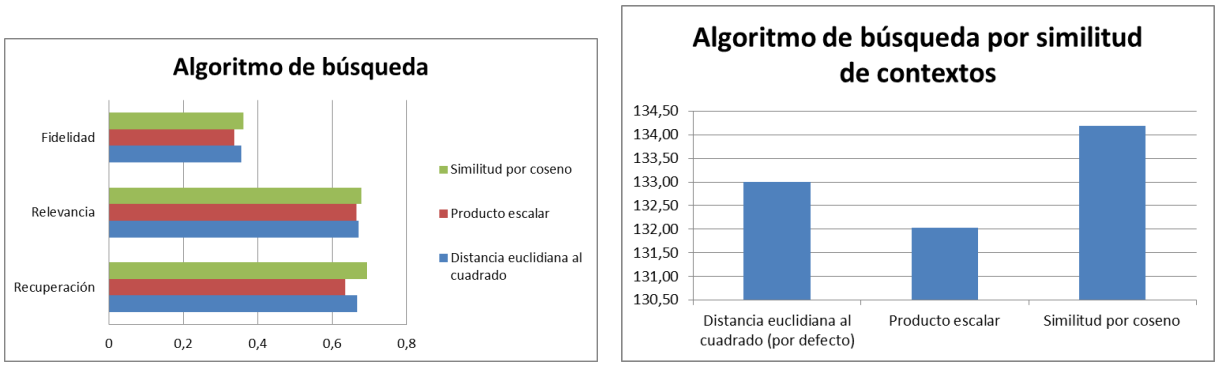
- Distancia l2 al cuadrado
- Producto escalar
- Similitud por coseno

Comparar las fórmulas de distancia requiere crear tres colecciones con parámetros y documentos idénticos, pero cada una configurada con una distancia distinta. Se usará la misma aplicación para usar y comparar el rendimiento de las colecciones.

Los resultados se pueden observar en la Tabla 6.8 y la Figura 6.8.

Elemento	Recuperación	Relevancia	Fidelidad	Tiempo (s)
Distancia l2 al cuadrado	0,667	0,671	0,356	133,00
Producto escalar	0,635	0,665	0,337	132,03
Similitud por coseno	0,693	0,677	0,361	134,19

Tabla 6.8: Algoritmos de búsqueda por similitud



(a) Medidas de Algoritmo de búsqueda por similitud

(b) Tiempos de Algoritmo de búsqueda por similitud

Figura 6.8: Algoritmo de búsqueda por similitud

Aunque las tres fórmulas de distancia obtengan puntuaciones de calidad y latencias de respuesta similares, como muestran las Figuras 6.8, la similitud por coseno parece conseguir la mejor calidad de respuestas, aunque también es la que más latencia produce. El producto escalar es la que menor puntuaciones de calidad obtiene, aunque parece ser el más rápido y la distancia L2 al cuadrado queda entre ambas en métricas de calidad y tiempo.

6.4. Discusión de resultados

Según las pruebas realizadas, la mejor combinación de componentes para la aplicación final serían:

1. multi-qa-mpnet-base-dot-v1
2. Eliminación de formatos e imágenes
3. Almacenamiento Padre-Hijo añadiendo resúmenes
4. Preguntas paralelas + Rerank
5. Sin monitorización

6. Mistral-Nemo:12b
7. RecursiveCharacterSplitter
8. Coseno

Esta combinación asegura un rendimiento adecuado y respuestas precisas, pero conlleva bastantes costes de procesamiento y tiempo debido al procesamiento adicional que requieren estas técnicas más complejas, ya que varias de ellas conllevan la repetición de uno o varios pasos de su respectivo proceso.

Por ello, se usará también un segundo conjunto de componentes que, aunque ofrezcan resultados algo menos precisos que los anteriores, son más eficientes y tardan menos en responder como opción alternativa a usar en la implementación del sistema final.

1. all-MiniLM-L6-v2
2. Eliminación de formatos e imágenes
3. Almacenamiento Padre-Hijo con resumen
4. Sin procesamiento de preguntas
5. Sin monitorización
6. Mistral-Nemo:12b
7. RecursiveCharacterSplitter
8. Coseno

Al probar el rendimiento de ambas configuraciones, se han obtenido los datos de la Tabla 6.9.

Elemento	**Recuperación**	**Relevancia**	**Fidelidad**	**Tiempo (s)**
Conf. Compleja	0,8418	0,9074	0,6189	467,98
Conf. Simple	0,8202	0,8764	0,6023	141,86

Tabla 6.9: Algoritmos de búsqueda por similitud

Como se puede ver, la versión simple es un 69,69 % más rápida que la versión compleja aunque use casi todos los mismos componentes, lo cual hace que sea una alternativa interesante para la implementación.

En estas configuraciones no se han incluido ni la extracción de filtros de las preguntas ni la monitorización de las respuestas ya que se ha considerado que no mejoran suficiente la calidad de las respuestas respecto a los costes que conllevan: la monitorización hace que

el tiempo de espera sea inconsistente y a veces puede duplicar el tiempo de respuesta, y la búsqueda con filtros depende del uso de metadatos manuales en la base de datos, que no siempre estarán disponibles, y en ocasiones puede afectar negativamente al resultado, ya que modifica la pregunta original.

Capítulo 7

Implementación del sistema final

Una vez hemos encontrado la mejor combinación de componentes, arquitecturas y configuraciones, podemos pasar a la implementación del sistema final, añadiendo una interfaz gráfica con Gradio, estableciendo Ollama como el servicio de ejecución de LLMs, añadiendo características adicionales y creando la herramienta de gestión de la base de conocimiento.

7.1. Componentes externos

Chroma

Chroma es una base de datos vectorial optimizada para almacenar, indexar y recuperar embeddings de manera eficiente, facilitando la búsqueda semántica basada en similitud. Permite gestionar grandes volúmenes de datos, integrándose con modelos de inteligencia artificial y frameworks como LangChain. Su diseño escalable y su API sencilla hacen que sea ideal para aplicaciones como recuperación aumentada de información (RAG), chatbots inteligentes y motores de recomendación, brindando una solución rápida y flexible para consultas en espacios vectoriales.

En este proyecto se ha usado como base de datos principal para guardar los embeddings de los documentos ejecutándose como un contenedor de Docker, accediendo por conexión HTTP mediante su cliente de Python y manejándola mediante su integración con Langchain, en vez de directamente por su API.

```
1
2 import chromadb
3 from langchain_huggingface import HuggingFaceEmbeddings
4 from langchain_chroma import Chroma
5
6 CHROMA_IP   = "localhost"
7 CHROMA_PORT = 7888
8 CHROMA_COLLECTION = "cos_qa-mpnet_aug"
9
```

```
10          cliente = chromadb.HttpClient(CHROMA_IP, CHROMA_PORT)
11          embeddings = HuggingFaceEmbeddings(model_name="sentence-
    ↪ transformers/multi-qa-mpnet-base-dot-v1")
12          chroma = Chroma(collection_name=CHROMA_COLLECTION,
    ↪ embedding_function=embeddings, client=cliente, collection_metadata
    ↪ ={"hnsw:space": "cosine", "hnsw:search_ef": 20})
```

Como se puede ver en la línea 11 del código, también se usa un modelo de vectorización externo procedente del repositorio de modelos de Sentence Embedding [18] en HuggingFace, al que se accede a través de la función de Langchain `HuggingFaceEmbeddings`, que permite descargar un modelo de *embedding* de HuggingFace y ejecutarlo.

Ollama

Ollama es una plataforma diseñada para ejecutar modelos de inteligencia artificial de manera local, optimizando su rendimiento y accesibilidad. Facilita la descarga, gestión y ejecución de modelos de lenguaje como Llama y Mistral sin necesidad de infraestructura en la nube. Su enfoque en la eficiencia permite a los desarrolladores y empresas utilizar IA avanzada en dispositivos personales o servidores privados, mejorando la privacidad y reduciendo costos. Con una API sencilla e integración con herramientas populares, Ollama es ideal para desarrollar chatbots, asistentes inteligentes y aplicaciones de procesamiento de lenguaje natural (PLN) de forma ágil y escalable.

En este proyecto se usa como un servicio de Windows en constante ejecución como plataforma para ejecutar modelos LLM de forma local, como Llama3.1 o Mistral-Nemo. Se accede a su funcionalidad mediante conexión HTTP usando su integración con Langchain, que usa la API de Ollama.

```
1 from langchain_ollama import ChatOllama
2
3 OLLAMA_URL = "http://localhost:11434"
4 MAIN_LLM   = "mistral-nemo:latest"
5 QUERY_LLM  = "llama3.2:1b"
6
7         self.llm = ChatOllama(base_url=OLLAMA_URL, model=MAIN_LLM)
8         self.multyQueryGenrator = ChatOllama(base_url=OLLAMA_URL, model=
    ↪ QUERY_LLM)
```

El primer objeto `llm` llama al modelo *Mistral-Nemo* [1], que es más potente y se usará para leer los contextos y generar la respuesta final.

El segundo objeto, `multyQueryGenerator` usa un modelo más pequeño: *llama3.2:1b* [9] [14]. Este solo se usará para generar las preguntas paralelas si se el usuario lo permite.

Gradio

Gradio es una herramienta de código abierto que permite crear interfaces web interactivas de forma sencilla para modelos de inteligencia artificial y aplicaciones de machine

learning. Su API intuitiva facilita a los desarrolladores y científicos de datos desplegar y compartir modelos sin necesidad de conocimientos avanzados en desarrollo web. Compatible con Python, Gradio permite la creación de demos accesibles desde un navegador, integrándose fácilmente con frameworks como TensorFlow, PyTorch y Hugging Face. Su enfoque en la accesibilidad y facilidad de uso lo hace ideal para visualizar resultados, recopilar retroalimentación y facilitar la interacción con modelos de IA en tiempo real.

En este proyecto se ha usado como forma sencilla de crear una interfaz adecuada para interactuar con el sistema, así como servidor para la aplicación aprovechando el servidor Uvicorn que trae incluido.

```
import gradio as gr
from rag import rag


def update(complexity: bool, rag: rag):
    rag.complexity = complexity

asistente = rag()

interface = gr.Chatbot(label="Chat time!", type="tuples")
with gr.Blocks() as demo:
    chatbot = gr.ChatInterface(type="tuples", fn=asistente.adapter,
    ↪ chatbot=interface, title="RAG chatbot de prueba", examples=[
    "¿Cuales son las posibles calificaciones finales que se le pueden
    ↪ dar a un estudiante de educación primaria en Castilla y León?",
    "Qué cuestiones se abordan desde las áreas de Ciencias y Lengua
    ↪ Castellana y Literatura?",
    "¿Qué implica la expresión en la etapa de primaria?"])
    check = gr.Checkbox(value=True, show_label=True, label="Búsqueda
    ↪ mejorada", info="Aumenta la calida a costa de mayor tiempo de
    ↪ procesamiento")
    check.input(rag.something(asistente), inputs=check, trigger_mode="
    ↪ once")

if __name__ == "__main__":
    demo.launch()
```

Aplicación RAG

La aplicación se divide en dos secciones: la interfaz, que se ejecuta en el cliente, toma las entradas y muestra las salidas, así como permite al usuario personalizar su experiencia; y el servidor que contiene toda la lógica de la aplicación y conecta con el resto de servicios y componentes, como la base de datos y la plataforma de LLM. Este servidor se ejecuta sobre una distribución de Uvicorn incluida en Gradio.

La sección de la interfaz se puede encontrar en el apartado de Gradio anterior, por lo que esta parte se centrará en el servidor, que implementa la búsqueda por la base de conocimientos, una mitad del sistema RAG.

Los métodos para el procesamiento de cada pregunta se estructuran como se muestra en la Figura 7.1

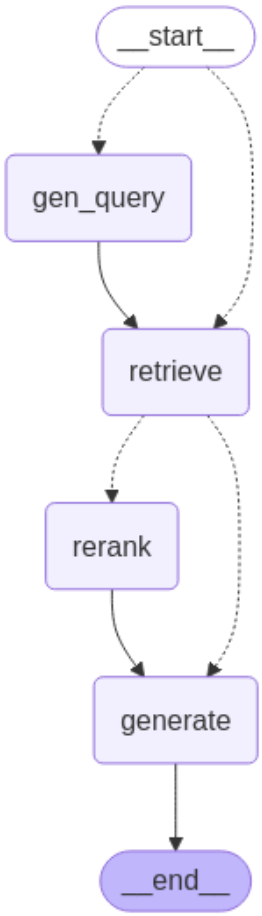


Figura 7.1: Flujo de procesamiento de cada pregunta que recibe el sistema RAG

Por defecto, se ejecutará la ruta más larga, que incluye `gen_query` y `rerank`, que son las funciones más costosas de la aplicación en términos de tiempo y capacidad de cómputo. Sin embargo, el usuario tiene a su disposición un interruptor que desactiva estas funciones, acelerando la búsqueda aunque probablemente reduciendo la calidad de las respuestas, como se indicó al final del Capítulo 6.

Los métodos del sistema son los siguientes:

```
def gen_query(self, state: State):

    prompt = self.query_gen_prompt.invoke(state["question"])
    raw_queries = self.multyQueryGenrator.invoke(prompt)

    queries = LLMtoList(raw_queries.content)

    queries.append(state["question"])
    print(queries)
```

```
10
11      return {"queries": queries}
```

La función `gen_query` toma la pregunta inicial del usuario y utiliza un modelo de lenguaje para generar 5 preguntas nuevas que buscan la misma información, pero con otras palabras, cubriendo mejor el espacio de vectores y asegurando una búsqueda más exhaustiva.

Para ello también usa un prompt distinto al principal con un ejemplo del formato de salida, ya que la respuesta de este modelo debe poder transformarse a un array de cadenas de texto, por lo que debe adherirse todo lo posible al formato indicado.

También usa otro modelo de lenguaje más pequeño y rápido que el principal, usado para generar la respuesta final. Esto se debe a que esta tarea no requiere de tanta potencia de razonamiento, ya que solo necesita procesar una sola pregunta en vez varios fragmentos de documentos. Este modelo rápido es, en este caso, Llama3.2: 1b Instruct [14], que es similar a llama3.1:8b, pero algo más pequeño y eficiente.

```
1
2  def retrieve(self, state: State):
3
4      retrieved_docs = []
5      try:
6          for query in state["queries"]:
7              retrieved_docs.extend(self.retriever.invoke(query))
8      except KeyError:
9              retrieved_docs.extend(self.retriever.invoke(state["question"
       ↪ ]))
10
11     print("Documentos encontrados: \n" + str(retrieved_docs) + "\n\n")
12     return {"context": retrieved_docs}
```

La función `retrieve` recupera los documentos relevantes de la base de conocimiento según la similitud de sus vectores con la pregunta del usuario, es decir, de acuerdo a cómo de relacionados estén sus temas.

En caso de haber más de una pregunta debido al paso anterior de generación de preguntas paralelas, entonces busca cuatro documentos para cada pregunta, hasta 24, aunque se pueden repetir. Más tarde estos documentos se reducirán a un número manejable por el LLM final, pero ahora el recuperador de contexto deja todos al siguiente paso.

```
1
2  def rerank(self, state: State):
3
4      print("Tamaño contexto inicial: ", len(state["context"]))
5
6      model = HuggingFaceCrossEncoder(model_name="BAAI/bge-reranker-v2-m3"
       ↪ )
7
8      compressor = CrossEncoderReranker(model=model, top_n=4)
9      final_context = compressor.compress_documents(state["context"],
       ↪ state["question"])
```

```
10
11      print("Documentos finales: ", final_context)
12      return {"context": final_context}
```

La función `rerank` recibe todos los documentos encontrados de la fase de recuperación y los reordena y reduce, pero solo si se han generado preguntas paralelas previamente, en la función `gen_query`, ya que se produce un exceso de documentos.

Debido a esto, se necesita una función como esta que elimine documentos duplicados y determine cuales son los mejores documentos de los recibidos, devolviendo el mismo número de documentos que se habrían recibido sin la búsqueda extendida, 4.

Para hacer esto, usa un modelo de embedding especial, *bge-reranker-v2-m3* de la *Beijing Academy of Artificial Intelligence* [13], [7], entrenado específicamente para obtener la relevancia del contexto respecto a una pregunta.

```
1
2  def generate(self, state: State):
3
4      print("Tamaño contexto final: ", len(state["context"]))
5      docs_content = format_docs(state["context"])
6
7      print("Contexto final: ", docs_content)
8      messages = self.prompt.invoke({"input": state["question"], "context"
       ↪ : docs_content})
9      response = self.llm.invoke(messages)
10
11     print(("Respuesta generada: ", response))
12     return {"answer": response}
```

La función `generate` produce la respuesta final usando el contexto de los documentos recuperados en los pasos anteriores mediante la LLM principal, Mistral-Nemo [1]. También se añaden la pregunta inicial de usuario y un mensaje de sistema para indicar al LLM cómo debe comportarse. El resultado de este paso será el mensaje final que se mostrará al usuario a través de la interfaz.

Sobre los métodos opcionales y la ruta de ejecución a tomar, por defecto, usa la ruta más larga, llamada "Búsqueda Mejorada“, que incluye generación de preguntas similares y comparación de los contextos obtenidos, por lo que requiere más tiempo de procesamiento en la GPU y tarda más en responder. Si el usuario desactiva en la interfaz la Búsqueda Mejorada, entonces se saltará estos pasos y solo realizará la búsqueda sobre una pregunta.

Contando con todos los pasos posibles, el esquema del sistema RAG completo para la sección de búsqueda queda como se muestra en la Figura A.8:

Como, se dijo anteriormente, esto solo representa la mitad del sistema RAG, la parte de búsqueda. A continuación se mostrará el funcionamiento de la sección de incorporación de documentos a través del Gestor de la Base de Datos.

Gestor de base de datos

Para permitir a los administradores manejar la base de datos de Chroma, se ha creado un administrador de BD que implementa la configuración correcta de Chroma. Esto permite a los administradores añadir, eliminar o consultar las colecciones activas en cualquier momento y con acciones mínimas.

Los administradores solo necesitan dar un directorio válido en la máquina host y el gestor se encargará de cargar todos los documentos en formato `.pdf` o texto plano con extensión `.txt`. También eliminará formatos, tablas y encabezados de los PDFs y realizará la fragmentación multinivel, creación de resúmenes y vectorización de forma automática antes de añadir las fuentes a la base de conocimiento. Por último, creará una copia del archivo original en una carpeta designada, que será la que los usuarios reciben al acceder a las fuentes de las respuestas que obtengan. Estas funciones se muestran en el código a continuación.

```
def add_docs(collection: Chroma, ampliar_db=False):
    """Pide el directorio de fuentes que añadir a la colección que recibe y procesa todos los archivos válidos automáticamente
    para añadirlos a la colección que recibe.
    También puede decidir si añadir resúmenes o no según la decisión previa del administrador.

    Argumentos:
        - collection: wrapped de Langchain para colecciones de Chroma
        - ampliar_db: por defecto "False", evita que se creen resúmenes de los documentos cargados. "True" reactiva este comportamiento.
    """


    ~~~~  Carga y eliminación de formato de archivos ~~~~

    # los fragmentadores usados sobre los documentos
    parent_splitter = RecursiveCharacterTextSplitter(chunk_size= 10000)
    child_splitter = RecursiveCharacterTextSplitter(chunk_size= 1000)

    #almacén de documentos originales
    if ( bool(collection.get()["ids"]) ):
        with open(f"{collection._collection_name}.pkl", 'rb') as inp:
            store = pickle.load(inp)
    else:
        store = InMemoryStore()

    # retriever Padre-Hijo de Langchain
    # dados una base de datos vectorial y un almacenamiento clave-valor, organiza automáticamente los documentos padre y fragmentos hijo
    # si se le dan fragmentadores para ambos tamaños, también puede hacer la fragmentación multinivel por si mismo
    retriever = ParentDocumentRetriever(
        vectorstore=collection,
```

```
        docstore=store,
        child_splitter=child_splitter,
        parent_splitter=parent_splitter,
    )

    print("Introduciendo archivos")
    ids = [str(uuid.uuid4()) for doc in docs]
    retriever.add_documents(documents=docs, ids=ids)
    print("Archivos introducidos")

    # actualiza el almacen de documentos
    with open(f"{collection._collection_name}.pkl", 'wb') as outp:
        pickle.dump(store, outp, pickle.HIGHEST_PROTOCOL)

    # carga un modelo pequeño y lo prepara para resumir documentos
    llm = ChatOllama(base_url="http://localhost:11434", model="llama3
    ↪ .2:1b")
    prompt = ChatPromptTemplate.from_template("Resume el siguiente texto
    ↪ :\n\n{doc}")

    # si está activado, genera resúmenes para cada fragmento padre y lo
    ↪ añade a la BD vectorial para ser indexado junto con los fragmentos
    ↪  hijos
    # esto puede mejorar los resultados de la búsqueda, comprimiendo la
    ↪ información de un fragmento padre en vez de separándola, pero se
    ↪ tarda mucho
    if ampliar_db:
        print("Generando resúmenes")
        parent_docs = store.mget(ids)
        for doc in parent_docs:
            metadata = doc.metadata
            doc = doc.page_content
            query = prompt.invoke(doc)
            resumen = llm.invoke(query)
            retriever.vectorstore.add_texts([resumen.content], [metadata
    ↪ ])
        print("Resúmenes creados e introducidos")

    print("Los archivos han sido incorporados a la base de datos de
    ↪ forma existosa.")
```

El proceso de carga de documentos se muestra de forma gráfica en la Figura A.9.

Esto representa la otra mitad del sistema RAG, el de preparación de documentos e incorporación a la base de conocimiento.

El gestor también es capaz de listar los documentos que contiene la colección seleccionada y eliminar una colección y su almacén de documentos. Se usa mediante la terminal en la que se ejecuta, introduciendo los datos que pide a medida que se avanza. Una vez termina una orden, vuelve al menú inicial y no permite operaciones consecutivas en la misma colección. Esto asegura una mayor robustez y estabilidad, que es lo que se busca en un programa de administración de sistemas como este, al no mantener el estado entre

operaciones individuales.

Distribución

Aunque el propósito de la aplicación es demostrar la capacidad de los sistemas RAG, es necesario crear un sistema de distribución de la aplicación para permitir que otras personas puedan usarla como base para continuar su desarrollo.

Para ello, el código Python de la aplicación y del gestor se han compilado en dos ejecutables, y estos, junto con la imagen Docker del contenedor de Chroma y el instalador de Ollama han sido comprimidos en un comprimido ejecutable. De esta forma, solo es necesario instalar Ollama con el instalador provisto y crear el contenedor de Docker a partir de la imagen para comenzar a usar la aplicación.

Capítulo 8

Evaluación del Sistema

En este capítulo se realizan una serie de pruebas para comprobar que el funcionamiento del sistema alcanza la funcionalidad esperada. Para ello se realizan dos tipos de pruebas: pruebas de rendimiento y pruebas de calidad.

Las pruebas de rendimiento serían similares a las realizadas durante el apartado de experimentación, consistiendo en obtener y analizar varias métricas automáticas de calidad para la salida del sistema, así como medir el tiempo total de procesamiento y de carga de documentos, tanto en mi máquina personal como en la máquina virtual.

Para llevar a cabo las pruebas para la calidad real de las respuestas, se realizó un estudio de usabilidad con varios profesores para obtener su opinión sobre la utilidad de la herramienta y la usabilidad de la interfaz.

Pruebas de rendimiento

Para estas pruebas, se usará el mismo sistema presentado previamente en la sección de experimentación para realizar pruebas de forma automatizada y obtener puntuaciones de rendimiento sobre varios aspectos de la calidad del sistema y la velocidad de procesado de pregunta..

Además, también se presentará el tiempo de carga de documentos, que es el tiempo que tarda en procesar los documentos para introducirlos en la BD. Esta métrica no es tan importante como la velocidad de respuesta, ya que no afecta a los clientes, pero merece la pena tenerla en cuenta.

Estas pruebas se realizarán para ambos modos de ejecución de la aplicación final, con Búsqueda Mejorada (lenta) y sin ella, en la máquina virtual remota proporcionada por la universidad.

Como se puede ver en la tabla de resultados 8.1, la calidad de la recuperación del contexto y de la generación de la respuesta ha mejorado respecto a las pruebas individuales de la experimentación, lo cual era esperado. También se observa un descenso de la calidad de los resultados entre el sistema con Búsqueda Mejorada y el normal, aunque mucho

Sistema	Recuperación	Relevancia	Fidelidad	T(s) Respuesta	T(s) Carga
RAG (mejorado)	0,8429	0,9091	0,6202	272,989	21.099,275
RAG (simple)	0,8217	0,8832	0,6039	73,673	1.952,613

Tabla 8.1: Datos obtenidos de las pruebas en la máquina virtual

menor que respecto a las pruebas previas.

También se puede observar que el tiempo de ejecución se ha reducido mucho respecto a los obtenidos en la fase de experimentación, aproximadamente un 44,87 % más rápido. Esta marcada disminución en la latencia indica una alta dependencia del rendimiento de la aplicación de la potencia de la GPU de la máquina en la que se ejecuta, especialmente porque una mejor GPU es la única ventaja de la máquina virtual sobre el ordenador de desarrollo.

Pruebas de calidad

Para estimar correctamente la calidad real de las respuestas, la mejor opción es obtener las opiniones de usuarios potenciales sobre el sistema.

Para ello, se ha realizado un estudio de usabilidad simple con 3 profesores de primaria y secundaria. Para este estudio se ejecutó el sistema en el mismo ordenador en el que fue desarrollado, mi ordenador personal, y se realizó en el mismo centro educativo en el que trabajan.

El guión del estudio se puede encontrar en el anexo. Tras presentar la aplicación y el propósito del estudio, se les presentó la interfaz de la aplicación y se les pidió realizar tres preguntas sobre los temas de los documentos de la base de conocimiento. Tras realizar las preguntas, se les realizó un pequeño cuestionario sobre su experiencia y opiniones sobre la interfaz y calidad de las respuestas.

Las preguntas de este cuestionario eran:

- Para las respuestas a cada una de las tres preguntas:
  - ¿La respuesta de la aplicación resuelve su pregunta?
  - ¿La respuesta es completa, es decir, incluye toda la información necesaria para responder su pregunta?
  - Del 1 al 5, ¿cómo de legible y comprensible es la respuesta, sin tener en cuenta la precisión o relevancia del texto respecto a la pregunta?
  - Del 1 al 5, ¿cómo calificaría en general la calidad de la respuesta, teniendo en precisión, vocabulario, legibilidad, tono y otros aspectos?
- Del 1 al 5, ¿le han resultado útiles las fuentes en los mensajes?

- Del 1 al 5, ¿cómo de fácil de usar es la aplicación?

Este cuestionario obtiene las opiniones de los profesores para cada una de las preguntas respondidas para evitar la influencia de errores sobre la evaluación de resultados correctos si se evaluaran todas juntas. Tras el cuestionario se preguntó a los usuarios si tenían alguna otra opinión sobre el sistema para recogerla de forma independiente del cuestionario.

A partir de este cuestionario se recogen los datos mostrados en la Tabla 8.2, que muestran las respuestas de los usuarios sobre los resultados proporcionados por el sistema.

		Profesor 1	**Profesor 2**	**Profesor 3**
Pregunta 1	Utilidad	No	Si	Si
	Completitud	No	No	Si
	Comprensibilidad	4	4	5
	Calidad	2	4	5
Pregunta 2	Utilidad	Si	Si	Si
	Completitud	Si	Si	Si
	Comprensibilidad	5	4	5
	Calidad	5	4	4
Pregunta 3	Utilidad	Si	Si	Si
	Completitud	Si	No	Si
	Comprensibilidad	4	5	5
	Calidad	4	4	5
Utilidad de fuentes		2	3	4
Facilidad de uso		5	5	5

Tabla 8.2: Resultados del estudio de usuarios

Como se puede observar, aunque el número de profesores que ha resuelto el cuestionario es limitado y habría que aumentarlo para poder extraer conclusiones, en la mayoría de casos el sistema funciona sin ningún problema, y solo presenta 1 fallo total (no útil para la pregunta) y 3 parciales (respuesta parcial). Esto indica que, incluso con la capacidad de cómputo limitada de la máquina usada, el sistema es suficientemente eficiente como para dar un servicio útil. También se puede ver que, aunque las respuestas sean generalmente útiles, a veces tienen errores de comprensibilidad u otros tipos que afectan la calidad final.

Los usuarios también han reportado que la herramienta es muy fácil de utilizar. Uno de ellos usó la opción de acelerar las respuestas y quedó satisfecho. Además, valoraron positivamente poder acceder a los archivos originales desde la aplicación para poder contrastar la información.

También han comentado que el sistema es lento comparado con otras herramientas como ChatGPT. Aunque la comparación resulta evidente, no es realmente justo comparar uno de los mejores modelos de lenguaje actuales ejecutándose en la nube de servidores de OpenAI con un sistema RAG ejecutándose en un ordenador portátil al límite de sus capacidades. Este es un problema que se puede solucionar con hardware más potente, como se pudo ver en las pruebas de rendimiento al usar la máquina virtual con una GPU más potente.

Capítulo 9

Conclusiones y trabajo futuro

En este capítulo se desarrollan las conclusiones de los objetivos planteados y se proponen líneas de trabajo futuras.

Conclusiones

Se ha logrado alcanzar el objetivo principal de crear la aplicación web con toda la funcionalidad proyectada, y además con resultados favorables que indican el éxito de la experimentación previa a la fase de implementación que apuntan a un mayor margen de mejora en el futuro.

También se ha redactado el plan de desarrollo de software, incluyendo la planificación del proyecto y el análisis y diseño de la aplicación, y se ha podido llevar a cabo sin mayores modificaciones, aparte de los retrasos mencionados en la sección de Seguimiento del Proyecto.

La experimentación sobre los componentes, la implementación final del sistema y las pruebas también han sido documentadas para completar el informe sobre el desarrollo de la aplicación.

La aplicación desarrollada es capaz de recuperar la información relevante de una base de datos usando un sistema RAG y generar una respuesta adecuada a partir de esta que resuelva la pregunta del usuario de forma consistente, es decir, es un asistente de búsqueda de información funcional. Además, se puede acceder y usar fácilmente como una aplicación web y la gestión del sistema se ha simplificado automatizando el procesamiento de los documentos.

Sobre la distribución del software, el código Pyhton ha sido compilado en un ejecutable y, junto con la imagen del contenedor de Chroma y el instalador de Ollama, han sido comprimidos en un archivo comprimido ejecutable para facilitar su distribución, aunque requiere la instalación por separado de Docker y Ollama para poder ejecutarlo. Este sistema de distribución es fácil de utilizar y requiere instalación mínima de software.

En este proyecto se han podido aplicar varias habilidades y conocimientos adquiridos

durante la carrera, como la planificación del proyecto, el diseño de la aplicación y el desarrollo de una aplicación web.

A lo largo de este proyecto también he adquirido nuevos conocimientos y he aprendido sobre nuevas formas de aprovechar la Inteligencia Artificial. También me ha permitido planificar el desarrollo de un producto software, aunque no sea para uso comercial, y las complicaciones que este tipo de proyectos puede conllevar.

Líneas futuras

Este sistema ha sido desarrollado con el propósito de ser una prueba de concepto y para base de aplicaciones futuras, y no para su distribución directa en su estado actual. Por ello, algunas de las vías de mejora consisten en la continuación del desarrollo de la aplicación, como las siguientes:

- Si se desea utilizar el sistema RAG desarrollado en este proyecto en un entorno real, es recomendable desarrollar una interfaz y sistemas de soporte más apropiados para un uso orientado al público, añadiendo gestión de usuarios con distintos permisos para limitar acceso a ciertos documentos o facilitar la integración en páginas web, por ejemplo.
- Se propone establecer un sistema que permita a los usuarios subir sus propios documentos al servidor de forma temporal y privada para añadirlo al espacio de búsqueda, aumentando así las capacidades del sistema.
- El gestor de la base de datos, aunque funcione en su estado actual, podría hacerse más fácil de usar añadiendo una interfaz gráfica para interactuar con las colecciones. Podría, incluso, accederse directamente desde la página web de la aplicación y restringir su acceso para administradores.
- Sobre la mejora del sistema, podrían repetirse las pruebas de rendimiento de los componentes no utilizados en máquinas más potentes y con mejores modelos, lo cual podría llevar a resultados distintos a los alcanzados en este proyecto.

Parte II

Apéndices

Apéndice A

Anexos

A.1. Información complementaria

A.1.1. Guion de estudio de calidad

Este es el guion seguido para realizar las entrevistas del estudio de calidad con profesores, del que se muestran los resultados en el capítulo 8 Evaluación del Sistema, en las Pruebas de Calidad.

Buenos días, soy [entrevistador]. Lo primero de todo, muchas gracias por prestarme tu tiempo y aceptar participar en este estudio.

El propósito de este es probar la efectividad de un asistente desarrollado que funciona como ChatGPT que posiblemente utilices por internet, pero capaz de citar sus fuentes y dar información exacta y confiable.

Para esto, usa la pregunta que le hagas para buscar información entre varios documentos guardados y la usa como fuente para crear la respuesta. Además, también te dice en qué documentos ha encontrado la información y te permite comprobarlo por ti misma.

Ahora te voy a presentar la interfaz de la aplicación y tendrás que realizar tres preguntas al asistente. No te puedo explicar cómo funciona, pero tendrás algunos ejemplos de cómo debes realizar las preguntas aunque tu puedes elegir sobre qué quieres preguntar. La única condición sobre las preguntas es que deben tratar sobre la educación primaria o secundaria en Castilla y León.

Después de cada pregunta, te haré algunas cuestiones simples sobre la respuesta que recibas, y cuando hayas terminado todas las preguntas puedes decirme lo que piensas del asistente.

[Ahora se realizan las preguntas. No se explica cómo funciona ni se responden preguntas sobre esto. Tras cada pregunta y respuesta se hacen las preguntas indicadas en la sección de Pruebas de Calidad.]

[Tras las tres preguntas.]

Ya casi hemos terminado el estudio. No tienes que hacer nada más, pero si tienes

alguna opinión sobre el asistente o la aplicación web, sería muy útil si la compartieses ahora. Puede ser sobre cualquier aspecto que te haya llamado la atención o cualquier opinión al respecto.

[Ahora se recogen las opiniones del usuario, si las da.]

Con eso ya hemos terminado entonces. Muchas gracias por tu colaboración.

A.2. Diagramas y tablas

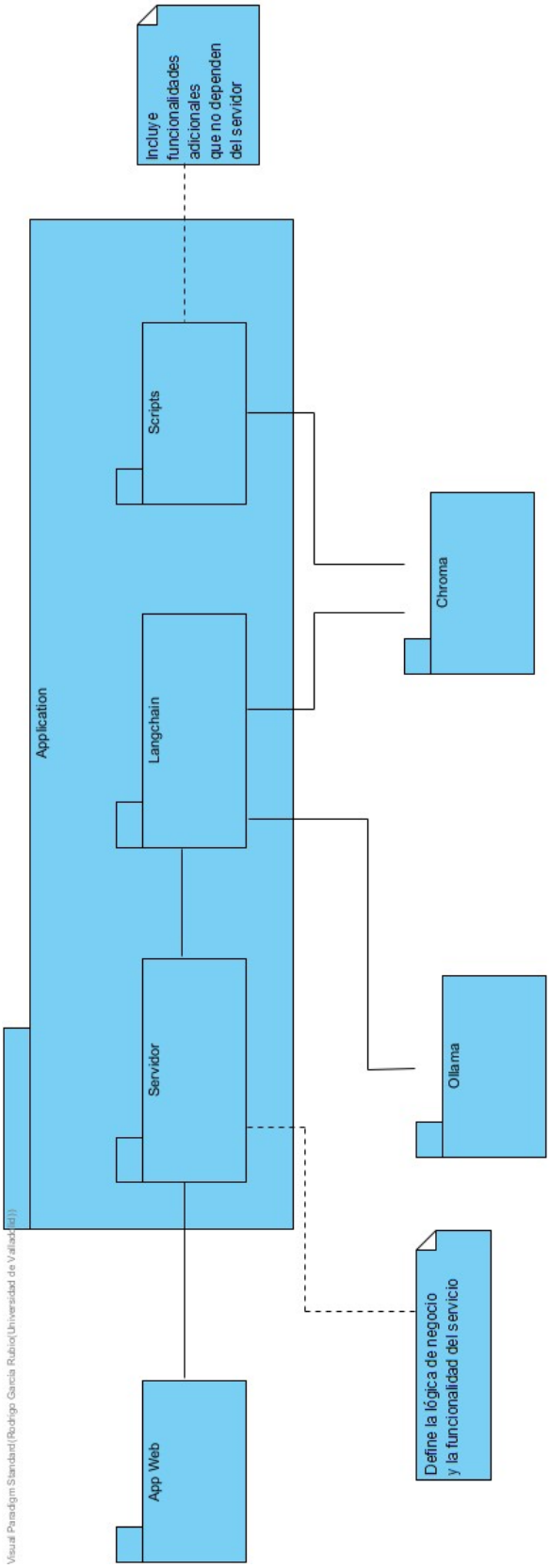


Figura A.2: Diagrama de paquetes del sistema

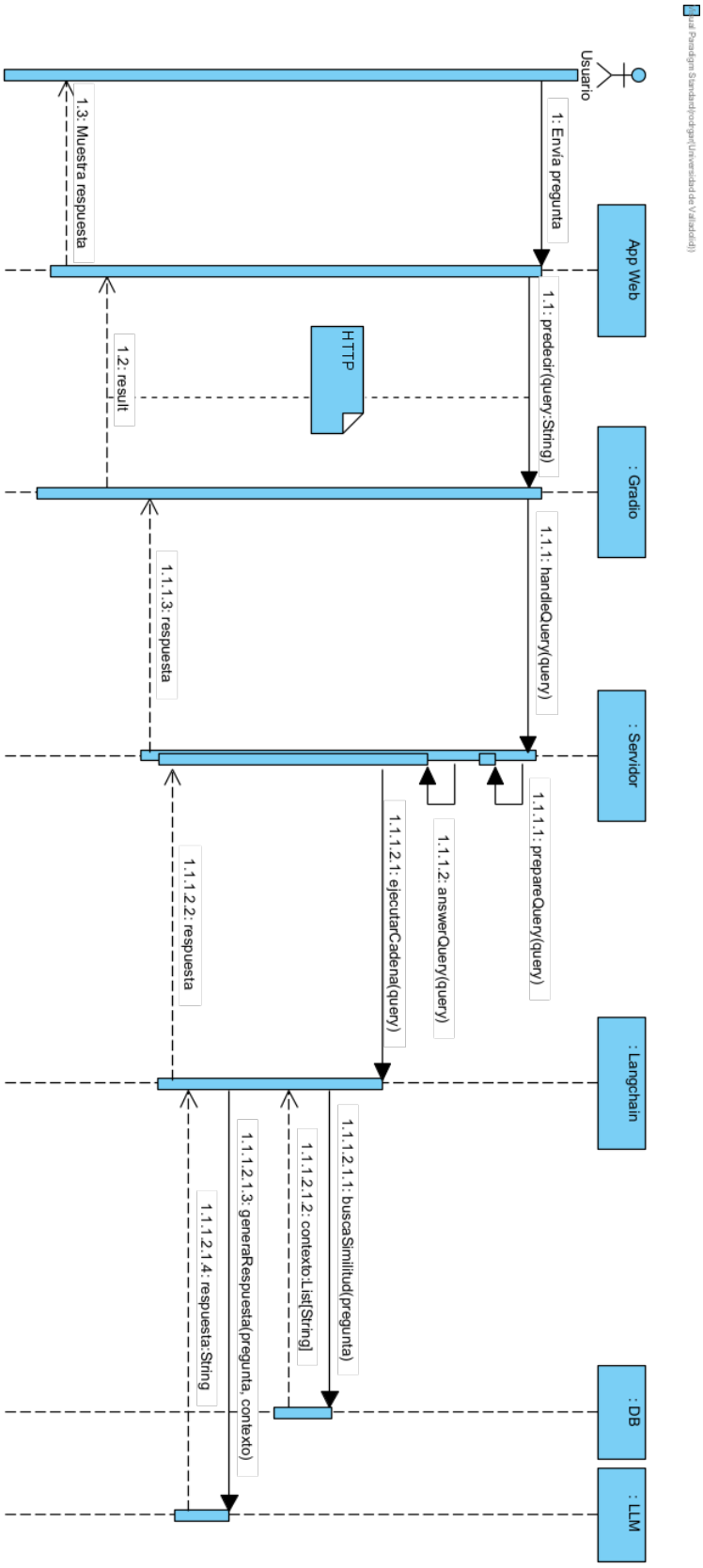


Figura A.3: Secuencia de acciones de Caso de Uso 1: Realizar Pregunta

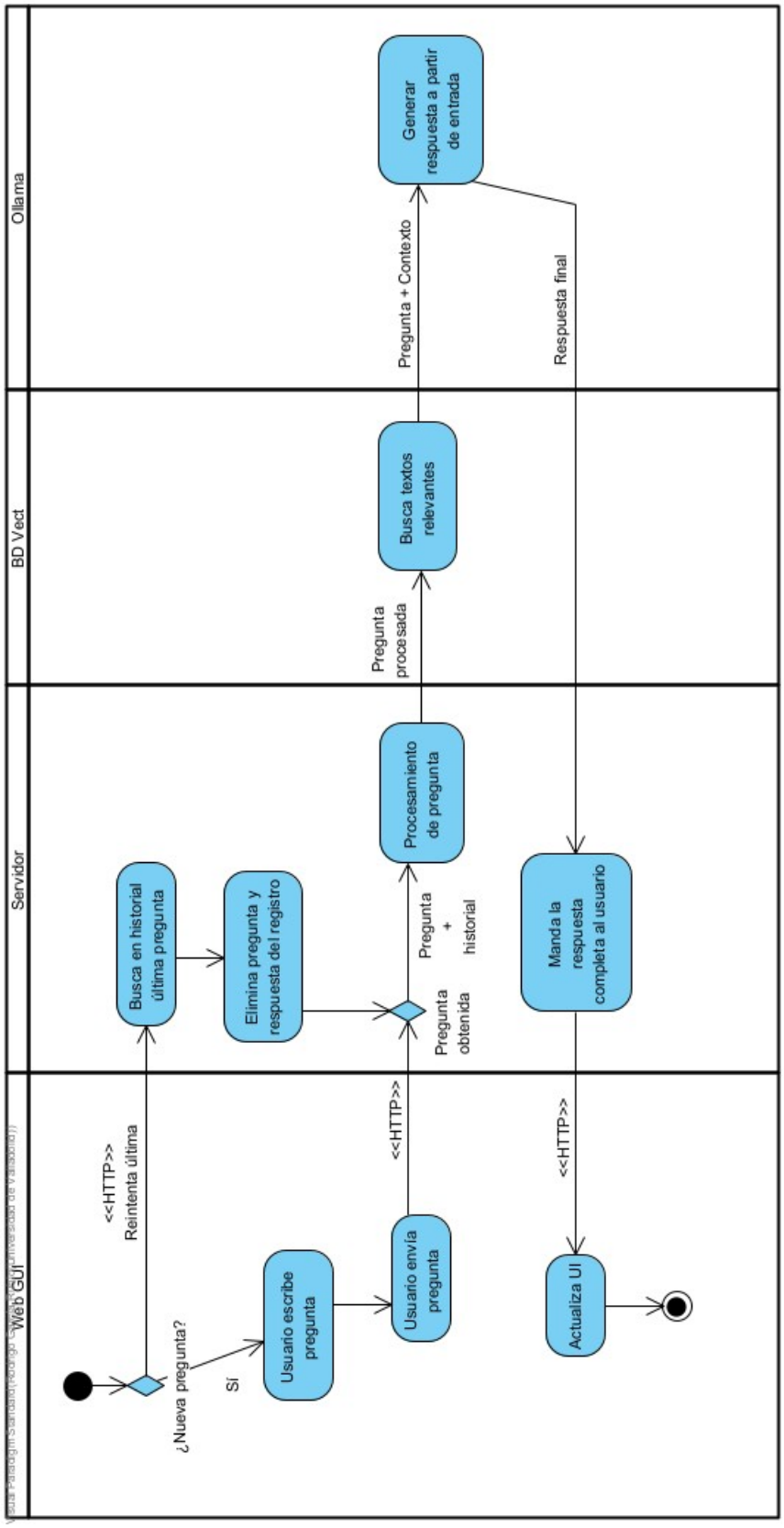


Figura A.4: Diagrama de actividades del CU-1: Realizar una pregunta

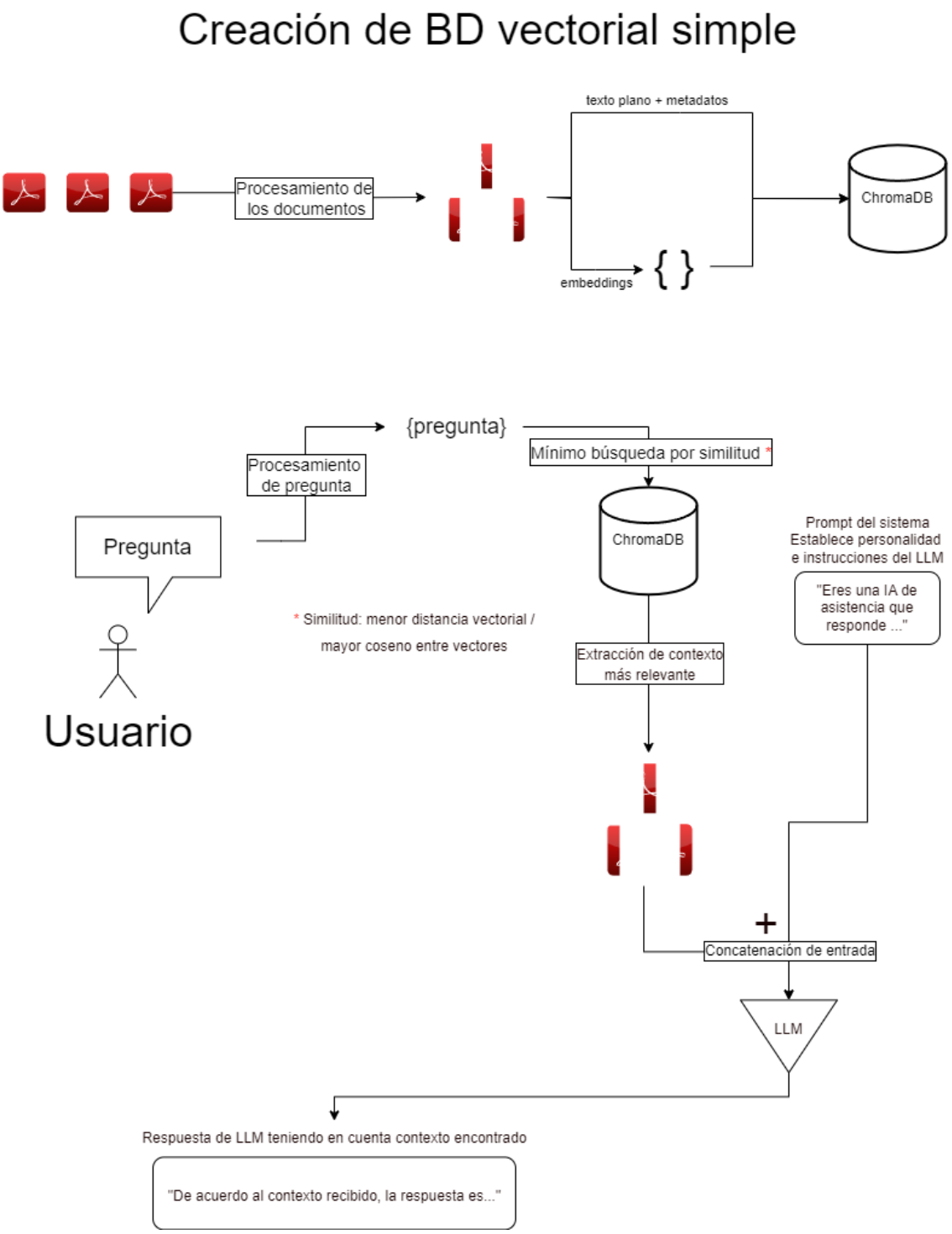


Figura A.5: Arquitectura de aplicación RAG

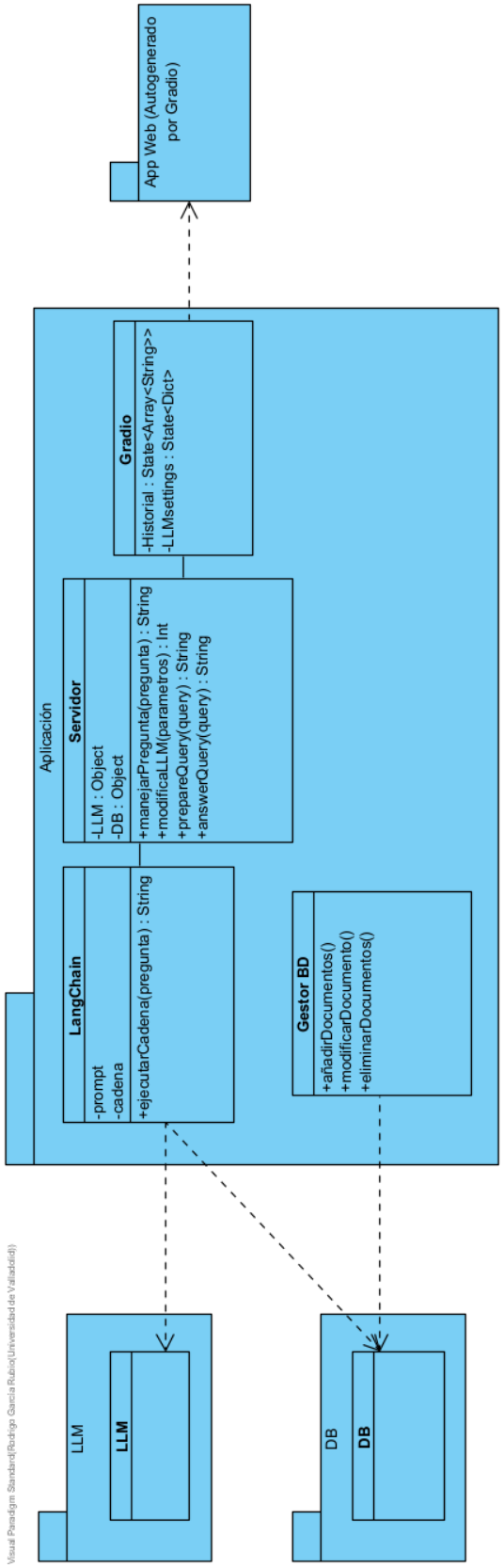


Figura A.6: Diagrama de clases de la aplicación

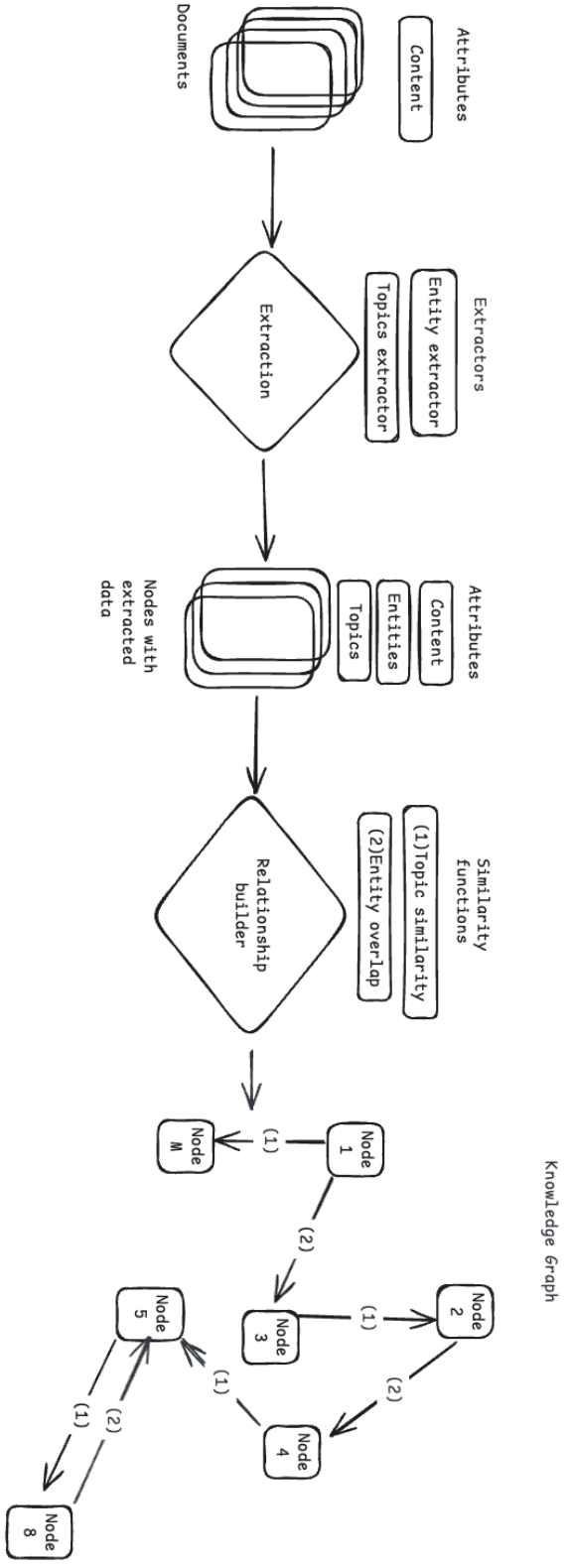


Figura A.7: Grafo de conocimiento generado por Ragas para crear ejemplos artificiales [27]

Etiquetas de fila	Fidelidad	Recuperación	Relevancia	Tiempo
Algoritmo de búsqueda por similitud de contextos				
Distancia euclidiana al cuadrado (por defecto)	0,356	0,667	0,671	133
Producto escalar	0,337	0,635	0,665	132,03
Similitud por coseno	0,361	0,693	0,677	134,19
Estrategia de fragmentación y almacenamiento de documentos				
Directa	0,356	0,667	0,671	133
Padre-Hijo	0,502	0,683	0,752	174,15
Padre-Hijo + Resúmen	0,5171	0,6983	0,8225	175,42
Fragmentador de documentos				
NLTKTextSplitter	0,364	0,599	0,595	247,39
RecursiveCharacterTextSplitter (por defecto)	0,356	0,667	0,671	133
SentenceTransformersTextSplitter	0,355	0,658	0,632	231,84
Modelo de generación de respuesta				
Llama 3.2:3b-instruct-q4_K_M	0,285	0,668	0,539	38,69
Llama3.1:8b-instruct-q4_K_M (por defecto)	0,356	0,667	0,671	133
Mistral-Nemo:latest (12b)	0,519	0,665	0,782	221,71
Modelo de vectorización de documentos				
all-MiniLM-L6-v2 (por defecto)	0,356	0,667	0,671	133
all-mpnet-base-v2 (más grande)	0,423	0,728	0,684	192,38
multi-qa-mpnet-base-dot-v1 (para búsquedas)	0,427	0,743	0,69	180,4
Monitorización y mejora de resultados				
Evaluación y repetición de búsqueda	0,436	0,715	0,722	421,59
Ninguna	0,356	0,667	0,671	133
Preprocesamiento de documentos				
Documentos originales	0,356	0,667	0,671	133
Eliminación de formatos e imágenes	0,355	0,722	0,734	132,38
Preprocesamiento de preguntas				
Combinación	0,504	0,843	0,753	811,02
Extracción de filtros	0,384	0,672	0,677	412,21
Generación de preguntas paralelas + Rerank	0,505	0,84	0,752	394,81
Ninguna	0,356	0,667	0,671	133

Tabla A.1: Resultados de escenarios de experimentación

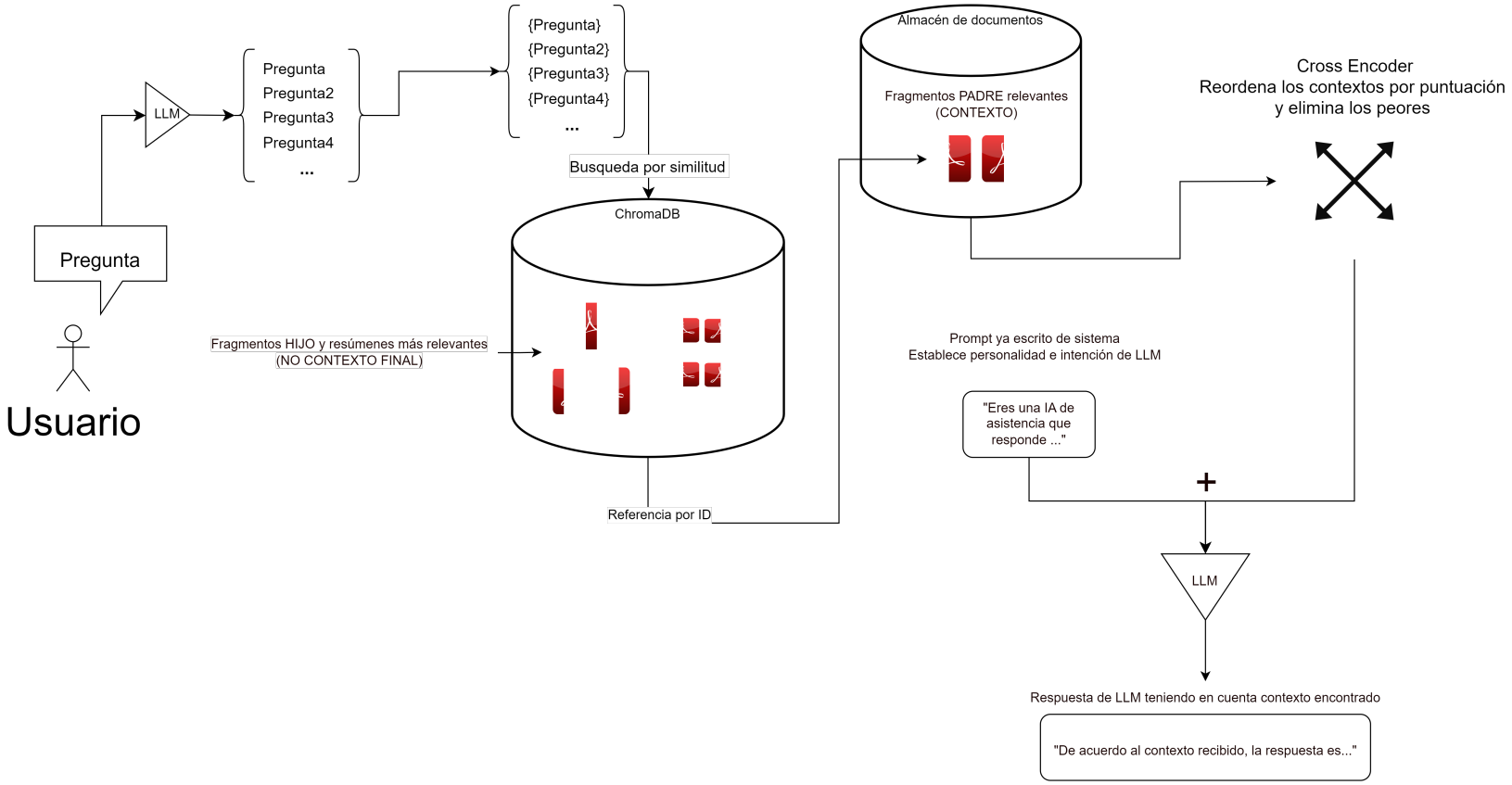


Figura A.8: Proceso de búsqueda de contextos en un sistema RAG

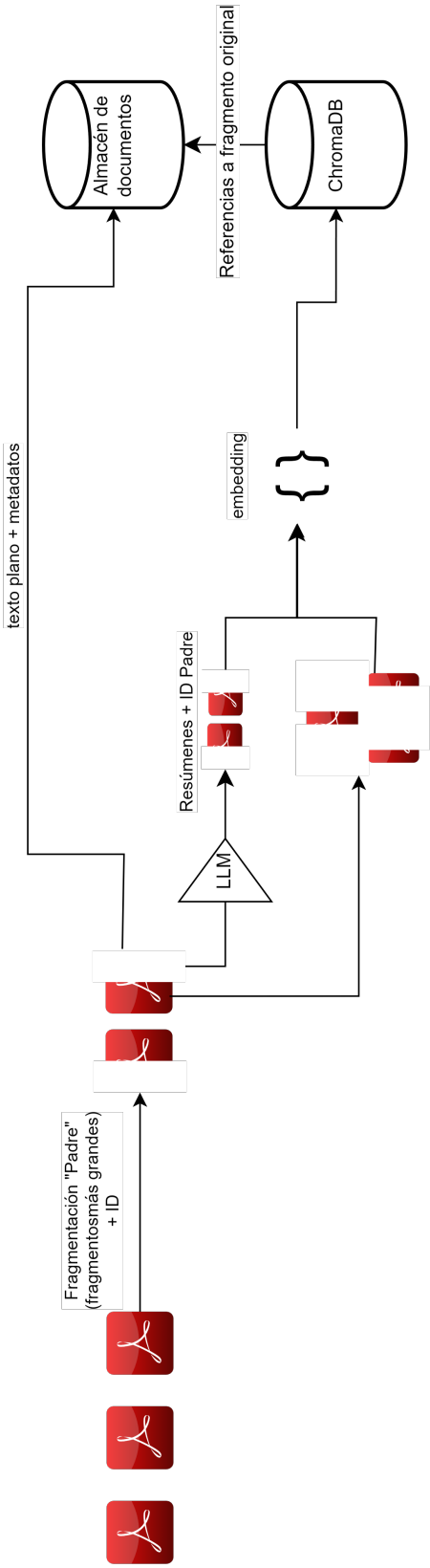


Figura A.9: Procesamiento de los documentos para introducirlos en la base de conocimiento

A.3. Manual de Instalación

No es necesario instalar la aplicación en sí, pero sí necesita otros programas que es necesario instalar.

- **Ollama**: el instalador de Ollama está incluido junto con los ejecutables y contiene la versión correcta. Una vez instalado, funciona como un servicio de Windows, por lo que se inicia automáticamente junto con el sistema. Este no incluye los modelos LLM a usar, solo es la plataforma para ejecutarlos.
- **Chroma**: la base de datos se usa como un contenedor de Docker. La imagen correcta es `chromadb/chroma:0.6.2.dev10` y es necesario indicar un directorio como volumen para la permanencia de la base de datos.

Una vez instalados y configurados los programas anteriores, es necesario editar el archivo `config.ini` para que el sistema encuentre todos los servicios. También será necesario disponer de uno o más modelos LLM para usar en el sistema, que se pueden descargar y usar directamente desde Ollama o descargando un modelo de HuggingFace y ejecutándolo mediante Ollama.

A.4. Manual de Uso

A.4.1. Manual de Usuario

Los usuarios del sistema solo usarán la aplicación web, que ya deberá estar puesta en ejecución por los administradores.

La interfaz se muestra en la siguiente imagen.

Los elementos de la interfaz marcados en la figura A.10 son los siguientes:

1. **Cuadro de chat**: aquí se mostrará la conversación con el chatbot, incluyendo las preguntas del usuario y las respuestas del asistente.
2. **Preguntas de ejemplo**: al cargar la página el asistente ofrece al usuario estas tres preguntas de ejemplo de estructura de las preguntas a realizar.
3. **Cuadro de texto**: permite introducir texto al hacer click sobre este.
4. **Botón de envío**: envía la pregunta escrita en el cuadro de texto al asistente. Si el cuadro de texto está vacío, no hace nada.
5. **Búsqueda mejorada**: al activar esta opción, ofrece mejores resultados pero tarda más en ofrecer una respuesta.
6. **Respuesta corta**: al activarla, reduce la longitud máxima de las respuestas.

Figura A.10: Interfaz de la aplicación

7. **Textualidad**: indica cómo de cercana y textual debe ser la respuesta del asistente al texto original de las fuentes usadas.

En las figuras A.11 y A.12 se muestra el cuadro de chat tras realizar una pregunta.

Figura A.11: Mensaje de ejemplo 1

Lo elementos marcados son los siguientes:

1. **Borrar**: elimina todas las preguntas y sus respuestas, vaciando el cuadro de chat.
2. **Pregunta realizada**: muestra la pregunta realizada al asistente.

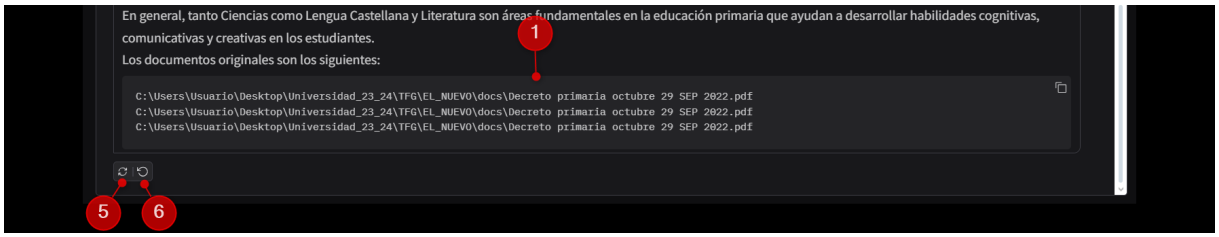


Figura A.12: Mensaje de ejemplo 2

3. **Respuesta**: muestra la respuesta del asistente.
4. **Fuentes**: muestra las fuentes originales de la información de la respuesta y ofrece un enlace para acceder al documento original.
5. **Reintentar**: elimina la respuesta obtenida y vuelve a buscar una respuesta nueva.
6. **Deshacer**: elimina la última pregunta y respuesta del cuadro de chat.

A.4.2. Manual de Administración

Configuración del sistema

Para establecer los parámetros globales del sistema, se debe editar el archivo "config.ini" en la carpeta de la aplicación, estableciendo las direcciones URL y puertos para cada uno de los servicios, así como los modelos a utilizar en el programa principal.

Gestión de la base de datos

Para gestionar la base de conocimiento del sistema, los administradores disponen de una herramienta para manipular la base de datos y los documentos que contiene.

Esta herramienta consiste en un ejecutable "gestor_bd.exe" que se ejecuta y maneja desde la terminal, como se puede ver en la Figura A.13, que muestra el menú inicial del gestor.

Al ejecutarla, se conecta automáticamente al servidor Chroma especificado en el archivo "config.ini" y muestra todas las colecciones disponibles. Para seleccionar una, basta con introducir su número, que aparece a la izquierda, y presionar Enter. Para crear una colección, seleccionar la última opción.

Administrar colección

Al elegir una de las colecciones, podemos añadir archivos de un directorio, eliminar toda la colección o ver qué documentos hay en la colección mediante el mismo método de elección, como se puede ver en la Figura A.14. Si se elige añadir archivos dará a elegir también si se quieren producir resúmenes adicionales para añadir a la base de

Figura A.13: Menú inicial del gestor de BD

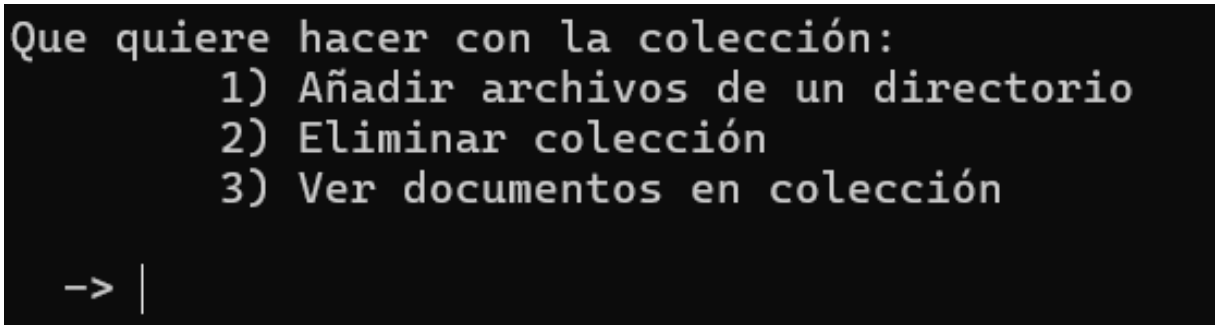


Figura A.14: Opciones de manipulación de colección

datos para mejorar las búsquedas, aunque esta opción añadirá un tiempo considerable al procesamiento de los documentos al añadirlos a la base de datos.

Por último, pedirá la ubicación de los documentos a cargar. Actualmente, solo soporta documentos en formato PDF y archivos de texto simples. Dada una dirección, introducirá todos los documentos disponibles en la base de datos, es decir, solo los documentos accesibles, que puedan ser leídos correctamente sin errores y que tengan el formato de documento correcto.

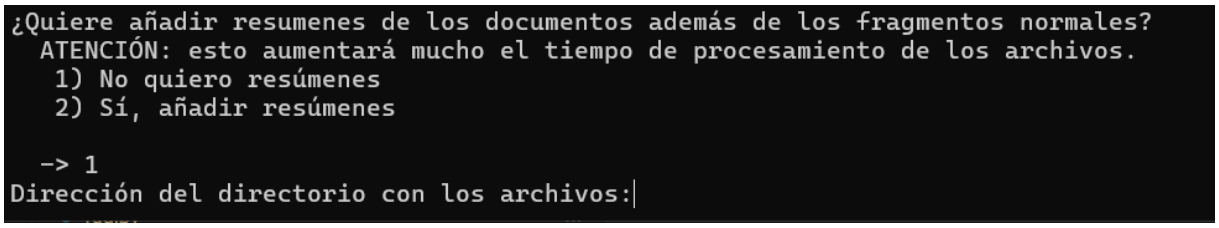


Figura A.15: Pasos para añadir documentos a la base de conocimiento

Una vez terminado, el gestor se reiniciará y volverá al menú inicial de la Figura A.13. No mantiene la conexión con la colección abierta tras cada operación para mejorar la estabilidad y evitar modificaciones inesperadas sobre la base de datos.

Crear colección

Tras elegir crear una colección, se pedirá el nombre de esta, como se muestra en la Figura A.16. Después, ofrecerá las mismas opciones que al elegir una colección preexistente.

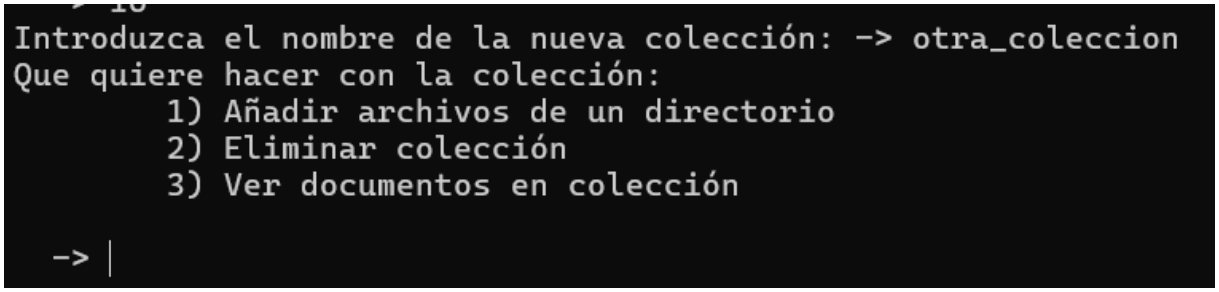


Figura A.16: Pasos para crear una colección

Ejecución del programa principal

Para ejecutar el sistema principal, se necesita:

- El servicio Ollama en funcionamiento
- El contenedor Chroma ejecutado y funcional

Después, solo es necesario ejecutar el archivo "asistente.exe", preferiblemente desde una terminal, para lanzar la aplicación.

Bibliografía

[1] Mistral AI. *mistralai/Mistral-Nemo-Instruct-2407 · Hugging Face*. URL: https://huggingface.co/mistralai/Mistral-Nemo-Instruct-2407 (visitado 20-03-2025).

[2] *AWS Deployment | Chroma Docs*. URL: https://docs.trychroma.com/deployment/aws#docker (visitado 25-08-2024).

[3] Junta de Castilla Y León. *DECRETO 37/2022, de 29 de septiembre, por el que se establece la ordenación y el currículo de la educación infantil en la Comunidad de Castilla y León - Portal de Educación de la Junta de Castilla y León*. URL: https://www.educa.jcyl.es/es/resumenbocyl/decreto-37-2022-29-septiembre-establece-ordenacion-curricul (visitado 05-04-2025).

[4] Junta de Castilla Y León. *DECRETO 38/2022, de 29 de septiembre, por el que se establece la ordenación y el currículo de la educación primaria en la Comunidad de Castilla y León - Portal de Educación de la Junta de Castilla y León*. URL: https://www.educa.jcyl.es/es/resumenbocyl/decreto-38-2022-29-septiembre-establece-ordenacion-curricul (visitado 05-04-2025).

[5] Junta de Castilla Y León. *DECRETO 39/2022, de 29 de septiembre, por el que se establece la ordenación y el currículo de la educación secundaria obligatoria en la Comunidad de Castilla y León - Portal de Educación de la Junta de Castilla y León*. URL: https://www.educa.jcyl.es/es/resumenbocyl/decreto-39-2022-29-septiembre-establece-ordenacion-curricul (visitado 05-04-2025).

[6] Junta de Castilla Y León. *ORDEN EDU/423/2024, de 9 de mayo, por la que se desarrolla la evaluación y la promoción en la Educación Primaria en la Comunidad de Castilla y León - Portal de Educación de la Junta de Castilla y León*. URL: https://www.educa.jcyl.es/es/resumenbocyl/orden-edu-423-2024-9-mayo-desarrolla-evaluacion-promocion-e (visitado 05-04-2025).

[7] Jianlv Chen et al. *BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation*. 2024. arXiv: 2402.03216 [cs.CL].

[8] *Chroma*. URL: https://www.trychroma.com/ (visitado 25-08-2024).

[9] Aaron Grattafiori, Abhimanyu Dubey et al. *The Llama 3 Herd of Models*. 2024. arXiv: 2407.21783 [cs.AI]. URL: https://arxiv.org/abs/2407.21783.

[10] Simone Kresevic et al. "Optimization of hepatological clinical guidelines interpretation by large language models: a retrieval augmented generation-based framework". En: *npj Digital Medicine* 7.1 (23 de abr. de 2024), pág. 102. ISSN: 2398-6352. DOI: 10.1038/s41746-024-01091-y. URL: https://www.nature.com/articles/s41746-024-01091-y (visitado 22-09-2024).

[11] *LangChain*. URL: https://www.langchain.com/langchain (visitado 25-08-2024).

[12] Patrick Lewis et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". En: ().

[13] Chaofan Li et al. *Making Large Language Models A Better Foundation For Dense Retrieval*. 2023. arXiv: 2312.15503 [cs.CL].

[14] Meta LLama. *meta-llama/Llama-3.2-1B-Instruct · Hugging Face*. Dic. de 2024. URL: https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct (visitado 20-03-2025).

[15] Jing Miao et al. "Integrating Retrieval-Augmented Generation with Large Language Models in Nephrology: Advancing Practical Applications". En: *Medicina* 60.3 (8 de mar. de 2024), pág. 445. ISSN: 1648-9144. DOI: 10.3390/medicina60030445. URL: https://www.mdpi.com/1648-9144/60/3/445 (visitado 22-09-2024).

[16] *NotebookLM | Note Taking & Research Assistant Powered by AI*. URL: https://notebooklm.google/ (visitado 22-09-2024).

[17] *Ollama*. URL: https://ollama.com (visitado 25-08-2024).

[18] Nils Reimers e Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". En: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, nov. de 2019. URL: https://arxiv.org/abs/1908.10084.

[19] *Running Chroma - ChromaDB Cookbook | The Unofficial Guide to ChromaDB*. URL: https://cookbook.chromadb.dev/running/running-chroma/#chroma-cli (visitado 31-08-2024).

[20] BarD Software s.r.o. *GanttProject: free project management tool for Windows, macOS and Linux*. GanttProject. URL: https://www.ganttproject.biz (visitado 06-09-2024).

[21] Gradio Team. *Gradio*. URL: https://gradio.app (visitado 25-08-2024).

[22] Tino Max Thayil, Da Chen y Yongtae Hwang. *Context Recall - Ragas*. URL: https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/context_recall/ (visitado 20-06-2025).

[23] Tino Max Thayil, Da Chen y Yongtae Hwang. *Faithfulness - Ragas*. URL: https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/faithfulness/ (visitado 20-06-2025).

[24] Tino Max Thayil, Da Chen y Yongtae Hwang. *Generate Synthetic Testset for RAG - Ragas*. URL: https://docs.ragas.io/en/stable/getstarted/rag_testset_generation/ (visitado 01-04-2025).

[25] Tino Max Thayil, Da Chen y Yongtae Hwang. *Ragas*. URL: https://docs.ragas.io/en/stable/ (visitado 01-04-2025).

[26] Tino Max Thayil, Da Chen y Yongtae Hwang. *Response Relevancy - Ragas*. URL: https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/answer_relevance/ (visitado 20-06-2025).

[27] Tino Max Thayil, Da Chen y Yongtae Hwang. *Testset Generation for RAG - Ragas*. URL: https://docs.ragas.io/en/stable/concepts/test_data_generation/rag/#knowledge-graph-creation (visitado 18-06-2025).

[28] *Tus proyectos*. URL: https://es.overleaf.com/project (visitado 25-08-2024).

[29] Ozan Unlu et al. *Retrieval Augmented Generation Enabled Generative Pre-Trained Transformer 4 (GPT-4) Performance for Clinical Trial Screening*. 8 de feb. de 2024. DOI: 10.1101/2024.02.08.24302376. URL: http://medrxiv.org/lookup/doi/10.1101/2024.02.08.24302376 (visitado 23-09-2024).

[30] *Zotero | Your personal research assistant*. URL: https://www.zotero.org/ (visitado 25-08-2024).