UNIVERSIDAD POLITÉCNICA DE MADRID

E.T.S. DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

PROYECTO FIN DE GRADO

GRADO EN INGENIERÍA DEL SOFTWARE

Implementación de un Chatbot para Mejorar la Experiencia Estudiantil en la Universidad

Autor: Joseph David Torres Troya

Director: Cristian Ramírez Atencia

Madrid, 7 de julio de 2025

Implementación de un Chatbot para Mejorar la Experiencia Estudiantil en la Universidad
Proyecto Fin de Grado, 7 de julio de 2025
Autor: Joseph David Torres Troya
Director: Cristian Ramírez Atencia

E.T.S. de Ingeniería de Sistemas Informáticos
Campus Sur UPM, Carretera de Valencia (A-3), km. 7
28031, Madrid, España

Si deseas citar este trabajo, la entrada completa en BibTeX es la siguiente:

```
@mastersthesis{citekey,
title = {Implementación de un Chatbot para Mejorar la
Experiencia Estudiantil en la Universidad},
author = {Torres Troya, J.D. \& Ramírez Atencia, C.} school =
{E.T.S. de Ingeniería de Sistemas Informáticos},
year = {2025},
month = {7},
type = {Proyecto Fin de Grado}
}
```

Agradecimientos

Le quiero agradecer a mi familia y a todos mis amigos que han hecho posible que llegue hasta aquí, al punto de finalizar mi carrera. También quiero agradecer a mi tutor Cristian por ayudarme con este trabajo y estar interesado en ser mi tutor para llevar a cabo este proyecto, además de mi amiga Carolina por darme la idea de este PFG.

Resumen

Resumen

Este *Proyecto de Fin de Grado* (PFG) trata sobre el desarrollo de un *chatbot* conversacional pensado para ayudar a los estudiantes de la Escuela Técnica Superior de Ingeniería de Sistemas Informáticos (ETSISI) a resolver dudas habituales sobre la universidad, como temas de secretaría, becas, matrículas o localización de servicios. El objetivo principal del proyecto es facilitar el acceso a esa información, que ya está disponible en la web oficial, pero que muchas veces cuesta encontrar.

El funcionamiento del sistema se basa en dos partes. La primera consiste en preparar los documentos cargados, dividiéndolos en fragmentos y convirtiéndolos en un formato que el sistema pueda entender, conocido como *embeddings*. Con eso se construye una base de datos semántica, que permite hacer búsquedas por su significado, no solo por palabras clave.

La segunda parte se da cuando el usuario hace una pregunta. El sistema busca los fragmentos más relacionados con esa pregunta y los usa como base para que el *modelo de lenguaje* genere una respuesta clara y coherente. De esta forma, el *chatbot* no inventa información, sino que responde en base a lo que encuentra en los documentos.

Para probar si el sistema funciona correctamente, se han hecho distintos tipos de preguntas: algunas con respuesta directa, otras sin respuesta, y otras más ambiguas. Esto ha servido para comprobar cómo se comporta el *chatbot* en distintos casos. Los resultados han sido positivos y se han cumplido los objetivos planteados al inicio del proyecto.

Finalmente, se incorpora funcionalidad específica para los perfiles encargados de realizar pruebas y mantener el sistema operativo. Entre estas funcionalidades se incluye la generación automatizada de documentos que reco-

pilan las respuestas ofrecidas por los distintos modelos de recuperación, con el fin de facilitar su análisis comparativo y la evaluación del rendimiento del sistema.

Palabras clave: *chatbot*, ETSISI, recuperación semántica, *modelo de lenguaje*, *embeddings*

Abstract

This *Bachelor's Thesis Project* focuses on the development of a conversational *chatbot* designed to assist ETSISI students in resolving common questions related to the university, such as administrative procedures, scholarships, enrollment, or the location of services. The main objective of the project is to facilitate access to this information, which is already available on the official website but is often difficult to find.

The system operates in two main phases. The first involves processing the uploaded documents by dividing them into fragments and converting them into a machine-readable format known as *embeddings*. These are used to build a semantic database that allows for searches based on meaning, rather than simple keywords.

The second phase takes place when a user submits a question. The system searches for the most relevant fragments and uses them as context for the *language model* to generate a clear and coherent response. In this way, the *chatbot* does not invent information, but instead responds based on the content found in the documents.

To evaluate the system's effectiveness, different types of questions have been tested: some with direct answers, others with no answer, and some intentionally ambiguous. This approach has helped assess how the *chatbot* behaves in various scenarios. The results have been positive, and the initial goals of the project have been successfully achieved.

Finally, specific functionalities have been included for those responsible for testing and maintaining the system. These features include the automated generation of documents compiling the responses given by different retrieval models, with the aim of facilitating comparative analysis and evaluating the system's performance.

Keywords: *chatbot*, ETSISI, semantic retrieval, *language model*, *embeddings*

Índice general

Índice de figuras

Índice de tablas

Índice de listados

1. Introducción

Este *Proyecto de Fin de Grado* tiene como objetivo desarrollar un sistema conversacional con inteligencia artificial, concretamente un *chatbot*, pensado para ayudar a los estudiantes de la ETSISI a resolver dudas habituales sobre la universidad.

El sistema está diseñado para responder a preguntas relacionadas con trámites administrativos, becas, matrículas, calendario académico o localización de servicios en el campus. Toda la información se obtiene directamente de los contenidos ya disponibles en la página web oficial de la ETSISI, incluyendo tanto el texto de la web como documentos descargables.

Para lograr esto, se ha usado una arquitectura llamada *Retrieval-Augmented Generation* (RAG). Esta arquitectura combina dos partes: una que busca fragmentos relevantes dentro de los documentos, y otra que genera respuestas basadas en esos fragmentos usando un modelo de lenguaje. Así se consigue que las respuestas estén basadas en información real y no sean inventadas por el sistema.

Este proyecto también permite aplicar conocimientos adquiridos a lo largo del grado, como el uso de modelos de lenguaje, procesamiento de texto, almacenamiento de datos o diseño de interfaces. Además, se evalúa cómo se comporta el *chatbot* ante distintos tipos de preguntas y se analiza si puede ser útil para mejorar la experiencia del estudiante en la universidad.

1.1. Objetivos

El objetivo principal de este proyecto es desarrollar un *chatbot* capaz de responder, de forma útil y comprensible, a las preguntas que realizan los estudiantes sobre distintos aspectos de la ETSISI, utilizando como base la información contenida en los documentos oficiales de la universidad.

Para conseguir este objetivo general, se han definido los siguientes subobjetivos específicos, que guían el desarrollo del sistema y permiten organizar el trabajo de forma estructurada:

Subobjetivos

1. Analizar los requisitos funcionales del sistema, identificando los casos de uso más relevantes para los estudiantes y definiendo cómo debe ser la interacción entre usuario y sistema.
2. Diseñar la arquitectura del sistema conversacional, separando sus componentes principales: carga y extracción de documentos, procesamiento semántico, recuperación de información y generación de respuestas.
3. Implementar un prototipo funcional que incluya:
    - Rastreo y descarga de archivos PDF a partir de enlaces en páginas web.
    - Extracción y tratamiento del contenido textual de los documentos.
    - Generación de respuestas a partir de preguntas formuladas en lenguaje natural.
4. Aplicar técnicas de recuperación semántica de información utilizando *embeddings* y comparar dos métodos de búsqueda: *Similarity* y *Maximum Marginal Relevance* (MMR).
5. Diseñar y realizar pruebas que permitan evaluar el rendimiento del sistema ante diferentes tipos de preguntas: claras, ambiguas y fuera de contexto.
6. Evaluar la calidad de las respuestas generadas, teniendo en cuenta aspectos como relevancia, claridad, coherencia y fundamentación en los documentos.
7. Documentar el desarrollo técnico del sistema, incluyendo las decisiones de diseño, las herramientas utilizadas, las limitaciones detectadas y posibles mejoras futuras.

1.2. Motivación

La idea de este Trabajo de Fin de Grado surgió a partir de una experiencia personal que, seguramente, comparten muchos estudiantes durante su paso por la universidad: la dificultad para encontrar de forma clara y rápida información académica importante, como los trámites de secretaría, becas, segundas matrículas o localización de servicios dentro del campus. A pesar de que esta información está disponible en la web oficial de la ETSISI, no siempre está organizada de forma intuitiva, lo que obliga a dedicar tiempo y esfuerzo en encontrar lo necesario.

Esta situación fue la principal motivación para desarrollar un sistema que facilitara ese acceso mediante una interfaz *conversacional*, en la que el usuario pudiera plantear su duda y recibir una respuesta precisa. Al mismo tiempo, el proyecto me permitía aplicar conocimientos que me interesaban especialmente, como la *inteligencia artificial*, el *procesamiento de lenguaje natural* y la *recuperación semántica de información*.

Además de esta motivación personal, existen también razones técnicas y de actualidad que justifican la elección del tema. En los últimos años, los *chatbots* en entornos educativos se han consolidado como una herramienta útil para ofrecer asistencia a estudiantes y automatizar tareas repetitivas. Estudios como los de Okonkwo y Ade-Ibijola (2021) y Kuhail et al. (2022) destacan su utilidad para responder preguntas frecuentes, mejorar la experiencia académica y optimizar recursos institucionales [1], [2].

Como se observa en la Figura 1.1, el interés por la inteligencia artificial ha experimentado un crecimiento sostenido en los últimos años, tanto en el ámbito académico como en el industrial.

Figura 1.1. Tendencia de búsqueda del término *inteligencia artificial* en Google Trends.

Por todo ello, este proyecto tiene como objetivo resolver un problema real en un contexto que conozco de primera mano, aplicando tecnologías actuales que están en constante evolución. Además, su diseño permite su reutilización y adaptación en otros centros educativos, o su integración futura en plataformas más avanzadas.

1.3. Justificación

Este proyecto resulta relevante tanto por su utilidad práctica para los estudiantes de la ETSISI, como por el valor académico y formativo que representa su desarrollo. La dificultad para acceder de forma rápida y clara a información institucional es un problema real que afecta al día a día del alumnado. Aunque dicha información está disponible en la página web oficial, su localización suele requerir múltiples pasos de navegación, lo que genera confusión y pérdida de tiempo.

El uso de un sistema conversacional basado en inteligencia artificial permite mejorar este proceso mediante una interfaz natural, donde el usuario puede plantear preguntas en lenguaje cotidiano y obtener respuestas relevantes sin necesidad de conocer la estructura interna de la web. Este tipo de soluciones ya ha demostrado su eficacia en otros entornos educativos y administrativos, por lo que su aplicación a nivel de centro universitario es perfectamente viable.

Desde el punto de vista técnico, este proyecto permite poner en práctica múltiples conocimientos adquiridos a lo largo del grado en Ingeniería del Software, incluyendo áreas como el desarrollo web, el procesamiento de lenguaje natural, la estructuración de datos y el diseño de prototipos. Además, se utilizan herramientas modernas como modelos de lenguaje de gran escala, técnicas de recuperación semántica con *embeddings*, y metodologías ágiles iterativas de desarrollo.

También se han aplicado conceptos vistos en asignaturas específicas como *Fundamentos de Ingeniería del Software* (FIS), relacionados con la definición de requisitos, análisis funcional y diseño de arquitecturas, así como en *Gestión de Proyectos y Riesgos* (GPR), en lo referente a la planificación, documen-

tación y evaluación de un sistema completo.

Por último, el sistema desarrollado tiene potencial para ser reutilizado o adaptado a otros grados o instituciones, o incluso integrarse con plataformas más amplias de gestión universitaria. Esto lo convierte en una propuesta escalable, con impacto real y posibilidad de evolución futura.

Además, el sistema ha sido complementado con una interfaz web sencilla, accesible desde el navegador, que permite a los estudiantes interactuar con el *chatbot* sin necesidad de conocimientos técnicos.

2. Marco teórico

Primero se presentará en este capítulo los fundamentos conceptuales y tecnológicos en los que se apoya el *Chatbot*.

2.1. Sistemas conversacionales y chatbots

Los sistemas conversacionales, comúnmente conocidos como *chatbots*, son programas informáticos diseñados para interactuar con los usuarios utilizando lenguaje natural, ya sea en formato escrito o hablado. Su propósito principal es simular una conversación humana para ofrecer asistencia, resolver dudas, facilitar tareas o simplemente mantener una interacción informativa.

Existen diferentes tipos de *chatbots* según su nivel de complejidad. Los más simples funcionan mediante reglas predefinidas, utilizando flujos estáticos y respuestas cerradas. Estos sistemas son fáciles de implementar, pero limitados en cuanto a flexibilidad, ya que no comprenden el contexto ni pueden adaptarse a preguntas inesperadas [3].

Por el contrario, los chatbots basados en inteligencia artificial y procesamiento del lenguaje natural (PLN) utilizan modelos estadísticos o redes neuronales que interpretan las intenciones del usuario y generan respuestas personalizadas. Este tipo de sistemas se ha extendido rápidamente en sectores como la atención al cliente, la sanidad, el comercio electrónico y, más recientemente, en el ámbito educativo [2]

El uso de *chatbots* en entornos universitarios permite automatizar la atención a estudiantes, responder consultas frecuentes y ofrecer soporte informativo de forma continua. Diversos estudios han demostrado que esta tecnología puede mejorar la experiencia académica al reducir tiempos de espera y ofrecer respuestas precisas [1]. Además, su capacidad para escalar sin intervención humana los hace especialmente útiles en contextos donde el volu-

men de consultas es elevado.

Este proyecto desarrolla un chatbot orientado a resolver dudas sobre información académica y administrativa de la ETSISI, combinando técnicas de recuperación semántica y generación de texto mediante modelos avanzados de lenguaje.

2.2. Modelos de lenguaje e inteligencia artificial

Los modelos de lenguaje son herramientas basadas en inteligencia artificial que permiten procesar, comprender y generar texto en lenguaje natural. Se construyen a partir de algoritmos de aprendizaje automático, en particular redes neuronales profundas, que aprenden patrones lingüísticos tras analizar grandes volúmenes de datos textuales.

Uno de los principales avances en este campo ha sido la aparición de arquitecturas de tipo *Transformer*, introducidas por Vaswani et al. (2017) [4], que permitieron mejorar significativamente la capacidad de los modelos para mantener el contexto y generar texto coherente. Esta arquitectura se ha convertido en la base de numerosos modelos de lenguaje actuales.

Entre los más conocidos destacan los modelos *GPT* (Generative Pre-trained Transformer), desarrollados por OpenAI. Estos modelos han demostrado un alto rendimiento en tareas como respuesta a preguntas, redacción de textos, resumen automático o traducción [5]. Su funcionamiento se basa en un preentrenamiento con grandes corpus de texto seguido de un ajuste fino (fine-tuning) para tareas específicas.

Sin embargo, los modelos de lenguaje de gran escala (Large Language Models, LLMs) presentan limitaciones importantes. Una de las más destacadas es que no disponen de acceso directo a información actualizada o específica fuera de su entrenamiento. Esto puede dar lugar a respuestas incorrectas o inventadas, lo que representa un riesgo especialmente en contextos educativos o institucionales [6].

Para mitigar este problema, en este proyecto se ha optado por una arquitectura que combina modelos generativos con mecanismos de recuperación

documental, lo que se describe en la siguiente sección.

2.3. Recuperación aumentada por generación (RAG)

Una de las limitaciones principales de los modelos de lenguaje de gran escala (LLMs) es que su conocimiento está restringido a los datos con los que fueron entrenados, lo que impide que accedan directamente a información específica o actualizada. Para resolver este problema, se han desarrollado arquitecturas híbridas que combinan generación de texto con mecanismos de recuperación de información. Entre ellas, destaca el enfoque conocido como *Retrieval-Augmented Generation* (RAG) [7].

El modelo RAG consiste en dos componentes principales: un sistema de recuperación documental y un modelo generativo. Primero, ante una pregunta del usuario, el sistema busca fragmentos de texto relevantes en una base documental indexada mediante *embeddings*. A continuación, esos fragmentos se utilizan como contexto para que el modelo de lenguaje genere una respuesta fundamentada en esa información. De esta forma, se evita que el sistema invente contenido, un problema común en los LLMs.

Como se observa en la Figura 2.1, el proceso RAG consta de dos fases principales: la indexación de documentos mediante representaciones vectoriales (almacenadas en una base semántica), y la recuperación de fragmentos relevantes para construir un *prompt* que se envía al modelo de lenguaje.

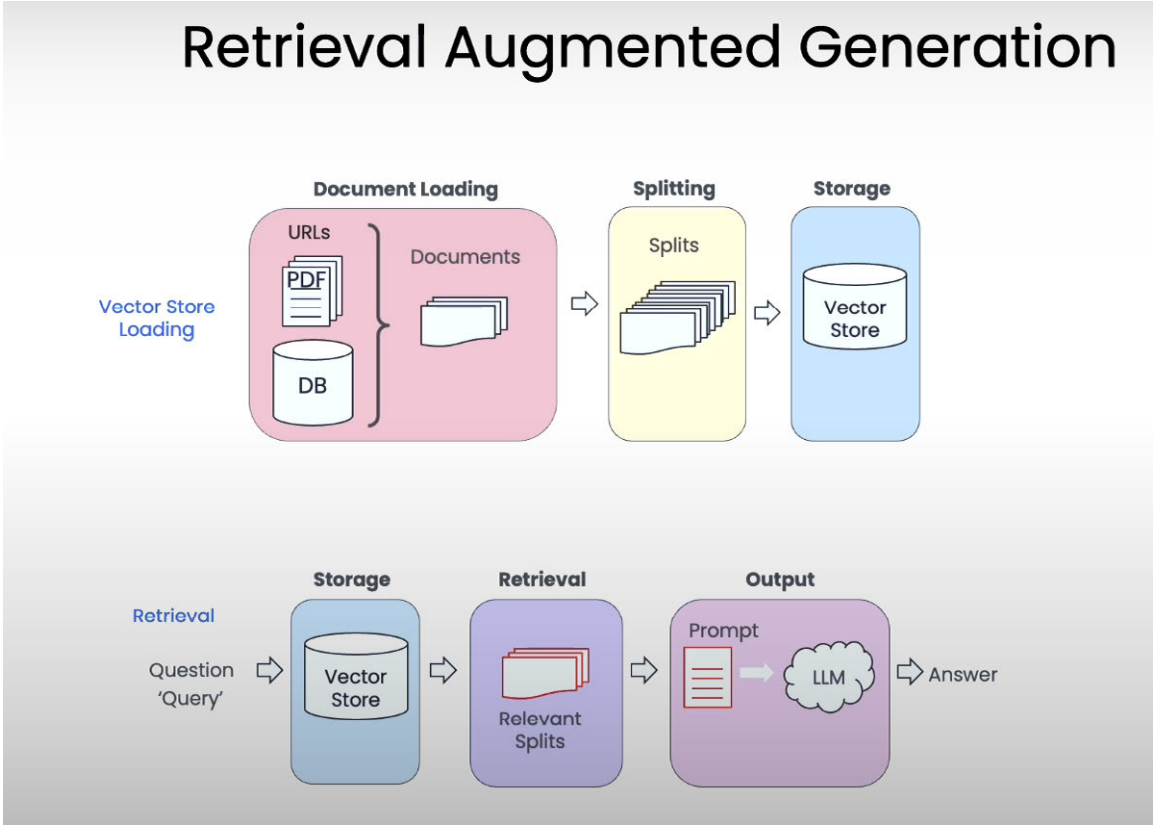


Figura 2.1. Arquitectura general de un sistema RAG. Adaptado de [8].

Este enfoque ha sido especialmente útil en tareas como respuesta a preguntas, asistencia técnica, generación de reportes o ayuda en entornos educativos. Según el curso de DeepLearning.AI [8], RAG permite construir aplicaciones conversacionales robustas sobre conjuntos de datos propios (locales) sin necesidad de reentrenar modelos, es decir, usar modelos ya entrenados sin la necesidad de complicaciones técnicas.

En este proyecto, se ha aplicado RAG para permitir que el *chatbot* acceda a información contenida en documentos oficiales de la ETSISI. Se han analizado dos estrategias de búsqueda:

- **Similarity**: selecciona los fragmentos que más se parecen a la pregunta del usuario, basándose en la distancia vectorial.
- **Maximum Marginal Relevance (MMR)**: prioriza tanto la relevancia como la diversidad de los fragmentos seleccionados, para evitar repeticiones y ampliar la cobertura de la pregunta.

Ambos métodos han sido probados y comparados durante la implementación para analizar su impacto en la calidad de las respuestas generadas.

2.4. Representación semántica con *embeddings*

Para que un modelo pueda recuperar información útil, es necesario representar el significado del texto de forma numérica. Los embeddings cumplen esta función: transforman palabras, frases o documentos en vectores dentro de un espacio de alta dimensión, donde las distancias entre vectores indican su similitud semántica [9].

Como se muestra en la Figura 2.2, un modelo de *embeddings* transforma un conjunto de objetos (por ejemplo, textos o frases) en vectores numéricos en un espacio de alta dimensión, lo que permite compararlos en función de su similitud semántica.

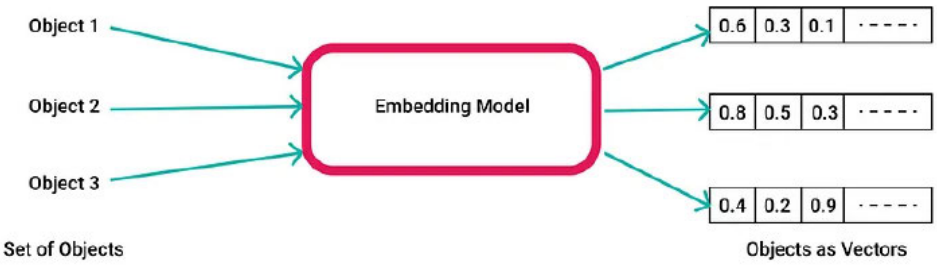


Figura 2.2. Conversión de objetos en representaciones vectoriales mediante un modelo de *embeddings*. Adaptado de [10].

Las técnicas de generación de *embeddings* han evolucionado notablemente en la última década. Modelos como Word2Vec [9] y GloVe [11] permitieron las primeras representaciones eficaces de palabras. Posteriormente, modelos como BERT [12] o Sentence-BERT [13] introdujeron contextos bidireccionales y representaciones a nivel de frase o documento.

En este proyecto, se han utilizado *embeddings* generados por modelos de OpenAI para representar fragmentos de documentos universitarios y preguntas formuladas por los usuarios. Estas representaciones permiten calcular la similitud entre una consulta y los textos disponibles, identificando así aquellos fragmentos más relevantes para generar una respuesta adecuada y lo

más precisa posible.

Este enfoque ha sido utilizado en múltiples aplicaciones, desde sistemas de recomendación hasta motores de búsqueda y asistentes virtuales [6]. Su principal ventaja es que no depende de coincidencias exactas de palabras clave, sino del significado general de las frases o preguntas, lo que permite manejar variaciones lingüísticas y sinónimos.

Como muestra la Figura 2.3, los vectores de embedding proyectados en 2D se agrupan según similitud semántica; las palabras relacionadas aparecen próximas en el espacio visual.

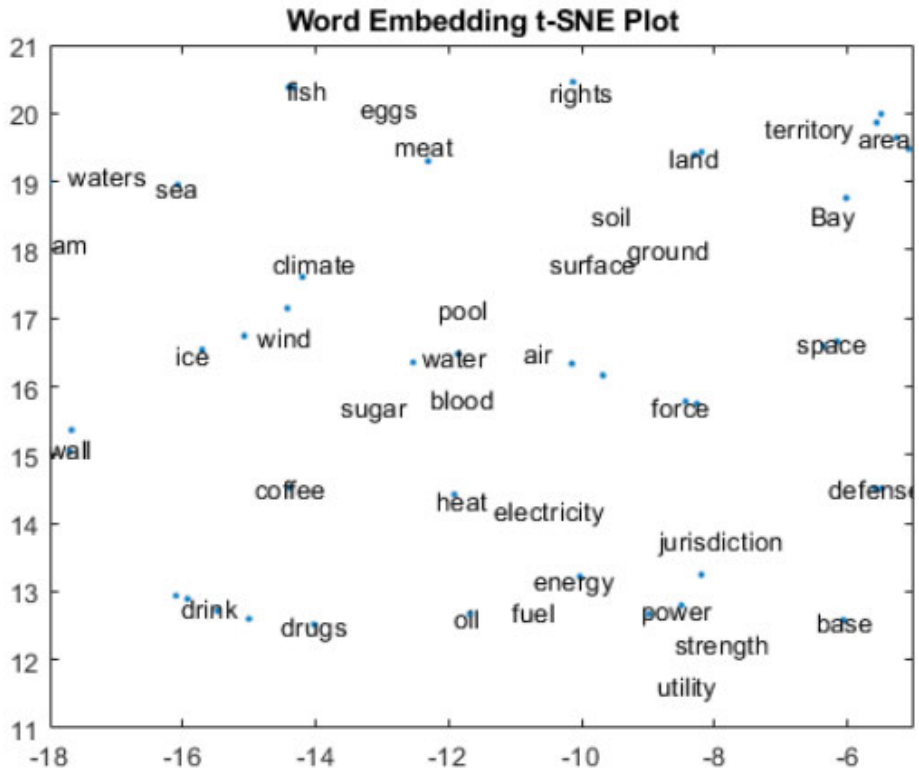


Figura 2.3. Visualización t-SNE de vectores de embedding de palabras, donde clústeres semánticos indican agrupación temática. Adaptado de [14].

La calidad del sistema depende en gran parte de la elección del modelo de *embeddings*, del tamaño de los fragmentos de texto (chunks) y del método de búsqueda empleado (Similarity o MMR, como se vio en la sección anterior).

2.5. Herramientas de desarrollo en sistemas conversacionales

En esta sección se describen las herramientas y tecnologías utilizadas para construir el sistema conversacional propuesto, justificando su elección frente a otras alternativas habituales en el desarrollo de aplicaciones web e inteligencia artificial.

HTML, CSS y JavaScript

La interfaz web del sistema ha sido desarrollada con tecnologías estándar ampliamente adoptadas:

- **HTML** define la estructura de la página.
- **CSS** permite aplicar estilos visuales y un diseño adaptable.
- **JavaScript** gestiona la interacción del usuario, captura la entrada de texto, y se comunica con el backend.

Este enfoque ofrece una solución ligera, accesible desde cualquier navegador y adecuada para prototipos funcionales.

Aunque existen otras librerías y tecnologías que se adaptan muy bien a los *chatbots*, como Gradio, el objetivo de este enfoque ha sido proporcionar un prototipo semejante al *Moodle* de la UPM, simulando su inclusión en la página web de la universidad.

Python, FastAPI y librerías de inteligencia artificial

El backend se ha desarrollado en **Python**, un lenguaje muy extendido en aplicaciones de inteligencia artificial y por la disponibilidad de librerías especializadas.

Se ha utilizado FastAPI como framework para crear la API, gracias a su sencillez y alto rendimiento.

Las principales librerías utilizadas son:

- **LangChain**, para gestionar la carga, segmentación y vectorización de documentos, así como la composición de las cadenas de generación.
- **OpenAI**, para conectar con los modelos GPT mediante su API.
- **ChromaDB**, como base vectorial para la búsqueda semántica eficiente de fragmentos relevantes.
- **dotenv** y **Pydantic**, para la gestión del entorno y validación de datos respectivamente.

Este conjunto de herramientas permite implementar arquitecturas tipo RAG sin necesidad de entrenar modelos desde cero, acelerando el desarrollo y reduciendo los requisitos computacionales.

2.6. Bases de datos vectoriales y almacenamiento semántico

Para el almacenamiento y la indexación de los fragmentos lingüisticos de los documentos, se ha empleado *ChromaDB*, una librería especializada con librerías de tecnologías como *Langchain* y muy usada en el ámbito de APIs GPT.

En lugar de una base de datos relacional, este sistema utiliza un almacenamiento semántico basado en vectores. Cada fragmento de texto se convierte en un embedding y se almacena en ChromaDB, lo que permite compararlo con las consultas del usuario según su significado. Esta técnica ha mostrado buenos resultados en sistemas de preguntas y respuestas [15].

Además, permite actualizar la base documental sin reentrenar el sistema, algo especialmente útil en entornos como el universitario, donde los contenidos cambian con frecuencia.

2.7. *PROMPT* en LLMs

Cuando hablamos de sistemas basados en modelos de lenguaje, como los *chatbots*, uno de los aspectos más importantes para obtener buenas respuestas es el diseño del *prompt*. El *prompt* es, en esencia, el mensaje que se le envía al modelo, y en él se incluye todo lo que el sistema necesita saber para generar una respuesta adecuada.

Aunque a veces el *prompt* puede ser tan simple como una pregunta directa, en muchos casos conviene estructurarlo de forma más detallada, especialmente si queremos que el modelo se comporte de una manera concreta o utilice cierta información como base. A esto se le conoce como *prompt engineering*, o ingeniería de *prompt*.

En esta práctica se suelen combinar varios elementos: instrucciones claras sobre qué debe hacer el modelo, contexto o fragmentos relevantes, la pregunta del usuario y, en ocasiones, alguna indicación sobre cómo debe devolverse la respuesta (formato, idioma, tono, etc.). Todo ello se junta en un único mensaje que se envía al modelo, y del que depende en gran medida la calidad de la respuesta generada.

Como se muestra en la Figura 2.4, el *prompt* incluye instrucciones, fragmentos relevantes y la pregunta del usuario, presentados de forma clara para guiar al modelo.

```
qa_prompt = PromptTemplate(
    input_variables=["context", "question"],
    template="""
Usando únicamente el siguiente contexto, responde la pregunta de forma concisa.
Si la respuesta no se encuentra en el contexto, responde: "No tengo información suficiente en los documentos proporcionados."

Contexto: {context}

Pregunta: {question}

Respuesta:
"""
)
```

Figura 2.4. Ejemplo de *prompt* utilizado en el sistema desarrollado.

Existen distintas estrategias que se pueden seguir al diseñar un *prompt*. Una de las más conocidas es *Chain-of-Thought prompting* (CoT), que consiste en inducir al modelo a razonar paso a paso antes de dar la respuesta final [16]. Es-

to es especialmente útil en tareas que requieren cierto razonamiento lógico. También está el enfoque *few-shot*, donde se incluyen varios ejemplos para que el modelo entienda mejor lo que se espera de él, o el *zero-shot*, donde simplemente se le dan instrucciones sin ejemplos previos [17].

Otro aspecto importante es el contexto. En sistemas como el desarrollado en este proyecto, ese contexto no es estático, sino que se construye dinámicamente a partir de fragmentos de documentos que se han cargado previamente. Así, el *prompt* no solo incluye la pregunta del usuario, sino también los textos más relacionados que el sistema ha recuperado, de forma que la respuesta se base en esa información y no en conocimientos generales del modelo.

También hay que tener en cuenta limitaciones técnicas, como el número máximo de *tokens* que puede procesar el modelo. Por ejemplo, GPT-3.5 puede manejar hasta 4096 tokens entre entrada y salida, mientras que GPT-4 puede llegar a los 8192 o incluso más en algunas versiones [18]. Esto obliga a ser cuidadoso con la cantidad de texto que se introduce en cada llamada al modelo.

Por último, la generación de texto también está influida por parámetros como la *temperatura*, que determina el grado de aleatoriedad en las respuestas. Una temperatura baja genera respuestas más precisas y conservadoras, mientras que una alta permite resultados más creativos [19].

3. Definición del proyecto

3.1. Metodología de trabajo

El desarrollo del sistema se ha organizado en torno a tres grandes bloques funcionales, que se abordaron de forma secuencial. Aunque no se siguió una metodología formal como Scrum o Kanban, el enfoque adoptado se basó en principios ágiles: avanzar de forma incremental, validar resultados a medida que se construía el sistema y adaptar el trabajo según las necesidades que iban surgiendo.

Cada bloque se desarrolló y probó de forma independiente, sin comenzar el siguiente hasta que el anterior estaba completado y ofrecía un comportamiento estable. Esto permitió mantener el control del desarrollo en todo momento y detectar errores a tiempo.

Las fases del trabajo fueron las siguientes:

1. En primer lugar, se implementó el sistema de descarga y procesamiento de documentos, que permitió construir una base de documentos realista con la que alimentar el sistema.
2. A continuación, se desarrolló un entorno de pruebas automáticas que toma directamente los archivos de test desde una carpeta específica y evalúa el comportamiento del sistema utilizando ambas estrategias de recuperación (MMR y Similarity), generando los resultados de forma organizada en otra carpeta para su posterior análisis.
3. Finalmente, se integraron todos los componentes en una aplicación funcional completa, incorporando una interfaz web que permite la interacción directa del usuario desde el navegador.

Durante cada una de estas etapas, se adoptó un enfoque iterativo: se imple-

mentaba una funcionalidad, se probaba, y si el comportamiento no era el esperado, se revisaba el código y se hacían ajustes hasta alcanzar un resultado satisfactorio. Esta forma de trabajar, centrada en bloques con validación interna, resultó especialmente útil para un proyecto de carácter individual, donde era importante avanzar con seguridad y de forma estructurada.

3.2. Descripción general del sistema

El sistema desarrollado es una aplicación conversacional basada en inteligencia artificial que permite a estudiantes universitarios realizar preguntas sobre procedimientos académicos y administrativos utilizando lenguaje natural. Su principal objetivo es facilitar el acceso rápido y preciso a la información contenida en documentos institucionales, como normativas, guías docentes o calendarios, mediante el uso de técnicas de recuperación semántica y modelos de lenguaje.

A diferencia de un buscador convencional o un sistema de navegación por menús, este asistente permite interactuar de forma directa a través de una interfaz web accesible desde cualquier navegador moderno. El usuario puede introducir preguntas libres, y el sistema se encarga de clasificar su claridad, recuperar la información más relevante desde su base documental y generar una respuesta coherente y fundamentada.

El sistema se apoya en una arquitectura tipo RAG (Recuperación Aumentada por Generación), lo que significa que las respuestas no son inventadas por el modelo, sino construidas sobre fragmentos reales extraídos de documentos previamente procesados y vectorizados. Esto garantiza una mayor fiabilidad y trazabilidad de la información.

Se ha diseñado como una herramienta modular y extensible, pensada para funcionar en entorno local sin requerimientos complejos, y con capacidad de actualizar sus datos sin necesidad de reentrenamiento. En conjunto, el sistema constituye un asistente de apoyo que mejora la accesibilidad a la información académica y sirve como base para futuras mejoras en el ámbito educativo.

3.3. Alcance del proyecto

El objetivo principal de este proyecto ha sido desarrollar un asistente conversacional capaz de responder preguntas sobre documentos académicos utilizando lenguaje natural. La idea era facilitar a los estudiantes el acceso a información relevante sin necesidad de navegar por múltiples páginas o documentos, todo ello a través de una interfaz sencilla accesible desde el navegador.

Desde el principio, se definieron algunas funcionalidades clave que el sistema debía cumplir: permitir que el usuario pudiera escribir preguntas de forma libre, clasificar automáticamente si eran claras, ambiguas o fuera de contexto, recuperar fragmentos relevantes desde los documentos y generar respuestas fundamentadas. También se incluyó una pequeña memoria que permitiera al sistema entender mejor preguntas ambiguas, además de un sistema para descargar y preparar automáticamente los documentos que servirían como base de conocimiento.

Eso sí, al tratarse de un proyecto individual y con tiempo limitado, el sistema tiene algunas limitaciones. Por ejemplo, solo guarda una interacción previa en memoria, no incluye control de usuarios ni autenticación, y está pensado para funcionar en local, sin despliegue en la nube. Además, su rendimiento depende de que los documentos cargados estén bien estructurados y sean coherentes.

Algo importante a tener en cuenta es que el alcance del proyecto no se fijó de forma cerrada desde el principio. Durante el desarrollo se fueron celebrando reuniones de seguimiento, aproximadamente una vez al mes, para revisar cómo avanzaba el sistema, resolver dudas y ajustar los objetivos si era necesario. Esto permitió adaptar el trabajo a las necesidades reales que iban surgiendo y tomar decisiones más acertadas en cada fase.

Aunque el sistema ya es plenamente funcional, se ha planteado como una base sobre la que se podrían incorporar mejoras en el futuro. Algunas de ellas podrían ser: ampliar la memoria conversacional, integrar otros modelos de lenguaje, desplegar el sistema en la nube, mejorar la interfaz gráfica o incluir un sistema de análisis automático de la calidad de las respuestas.

4. Análisis del sistema

El objetivo de este capítulo es analizar el sistema desarrollado desde una perspectiva funcional, para entender mejor cómo se estructura, qué necesidades busca cubrir y qué elementos son clave para su correcto funcionamiento. A través de este análisis se podrá evaluar si el sistema está bien alineado con los objetivos iniciales del proyecto y si cumple con los criterios necesarios para ser una herramienta útil en el entorno universitario y poder continuar con el desarrollo.

4.1. Stakeholders del sistema

Para comprender el alcance del sistema y su impacto, es necesario identificar a los actores o partes interesadas, los *stakeholders*, que interactúan directa o indirectamente con él. A continuación, se describen los principales perfiles involucrados en el uso, desarrollo o mantenimiento del sistema.

Usuarios finales

El usuario final principal es el estudiante universitario. Este perfil busca una vía rápida y accesible para obtener información relevante sobre aspectos académicos y administrativos, como normativas, fechas importantes, asignaturas, planes de estudio o procedimientos institucionales.

El sistema está diseñado para que pueda ser utilizado por cualquier alumno, mediante una interfaz web sencilla, accesible desde cualquier navegador. El usuario introduce su consulta en lenguaje natural y recibe una respuesta generada a partir de los documentos oficiales previamente cargados.

Desarrollador del sistema

Quién además actúa como responsable del mantenimiento. Es responsable de:

- Actualizar la base documental mediante el script de descarga.
- Realizar ajustes o mejoras técnicas sobre el sistema.
- Realizar tests de diferentes modelos y generar *CSV* para su análisis
- Verificar que las respuestas generadas mantengan la coherencia y relevancia.

Aunque actualmente el rol de desarrollador y mantenedor coincide con el autor del proyecto, a futuro se permite que en futuras versiones estos roles puedan ser asumidos por otros perfiles técnicos.

Institución académica (stakeholder indirecto)

Si bien no interactúa directamente con el sistema, la universidad (en este caso, la ETSISI) es un actor relevante, ya que los documentos que alimentan el sistema provienen de su web oficial y están destinados a sus estudiantes.

La utilidad del sistema puede repercutir positivamente en la institución al mejorar el acceso a la información y reducir la carga de atención a estudiantes por vías tradicionales (correo, secretaría, etc.).

4.2. Requisitos del sistema

Tras un análisis de las necesidades del sistema, se definieron los siguientes requisitos funcionales y no funcionales. Estos requisitos permiten asegurar que el sistema desarrollado cumpla con los objetivos planteados y pueda ofrecer una experiencia útil y robusta tanto para el usuario final como para el desarrollador encargado de su mantenimiento.

Requisitos funcionales

- **RF1**: El sistema debe permitir que cualquier usuario introduzca preguntas en lenguaje natural desde la interfaz web.
- **RF2**: El sistema debe clasificar automáticamente la pregunta como clara, ambigua o fuera de contexto utilizando un modelo de lenguaje.
- **RF3**: Si la pregunta es clara, el sistema debe recuperar fragmentos relevantes desde la base de vectores.
- **RF4**: El sistema debe generar una respuesta basada en el contenido recuperado, utilizando un modelo de lenguaje (LLM).
- **RF5**: Si la pregunta es ambigua y existe un contexto anterior válido, el sistema debe reformularla y procesarla automáticamente.
- **RF6**: El sistema debe mantener una memoria temporal con la última interacción válida para mejorar la comprensión de preguntas ambiguas.
- **RF7**: El sistema debe mostrar la respuesta generada directamente en la interfaz web.
- **RF8**: El sistema debe permitir al desarrollador cargar documentos PDF o TXT desde una carpeta local, que serán procesados automáticamente.
- **RF9**: El sistema debe generar y almacenar representaciones vectoriales del contenido textual para su posterior recuperación semántica.
- **RF10**: El sistema debe ejecutar pruebas automáticas tomando los archivos de test desde una carpeta específica y guardar los resultados obtenidos en formato CSV.
- **RF11**: El sistema debe permitir comparar automáticamente dos estrategias de recuperación semántica (*Similarity* y *MMR*) en cada ejecución de prueba, procesando los resultados por separado.
- **RF12**: El sistema debe conectarse a la API de OpenAI para realizar tareas de clasificación y generación de respuestas.
- **RF13**: El sistema debe permitir al desarrollador descargar automáticamente documentos PDF a partir de una lista de enlaces definidos previamente.

Requisitos no funcionales

- **RNF1**: El sistema debe estar disponible en entorno local sin necesidad de instalación compleja.
- **RNF2**: El tiempo de respuesta del sistema no debe superar los 5 segundos en condiciones normales.
- **RNF3**: La interfaz debe ser sencilla, responsiva y accesible desde navegadores modernos.
- **RNF4**: El sistema debe permitir añadir nuevos documentos fácilmente sin necesidad de modificar el código.
- **RNF5**: La información utilizada para generar respuestas debe estar basada exclusivamente en los documentos cargados.

5. Diseño del sistema

Este capítulo describe cómo está organizado internamente el sistema y cuál es el flujo funcional que sigue para responder preguntas del usuario. Se presentan los distintos módulos que lo componen y el recorrido que realiza una consulta desde que se introduce hasta que se devuelve una respuesta.

5.1. Fases funcionales principales

El funcionamiento general del sistema se divide en cuatro grandes bloques:

- *Carga y preparación de documentos*: un script automatizado recorre enlaces de la web de la ETSISI, descarga los documentos en formato PDF, verifica su validez y los almacena localmente para su posterior procesamiento.
- *Vectorización e indexado*: los documentos se fragmentan y se transforman en vectores mediante técnicas de *embedding*. Estos vectores, junto con sus metadatos, se almacenan en una base vectorial basada en ChromaDB.
- *Interacción con el usuario*: el sistema clasifica la pregunta recibida, recupera los fragmentos relevantes si procede, construye el *prompt* y genera una respuesta a través de un modelo de lenguaje. La respuesta final se presenta al usuario mediante una interfaz sencilla.
- *Generación y evaluación de pruebas*: el sistema permite ejecutar pruebas automáticas a partir de ficheros de texto con preguntas almacenados en una carpeta específica. Para cada test, se generan archivos de resultados en formato CSV, que recogen las respuestas generadas por distintos modelos o estrategias. Estos archivos se guardan automáticamente en una carpeta, lo que permite comparar de forma sistemática

el comportamiento del sistema en diferentes escenarios sin necesidad de procesar manualmente los datos.

5.2. Componentes del sistema

A continuación se describen los módulos principales del sistema desde el punto de vista funcional.

Controlador del sistema

Este componente actúa como núcleo coordinador de la arquitectura. Centraliza la lógica del flujo de preguntas, tanto en la interacción con el usuario como en las pruebas automatizadas.

Este controlador está implementado en el backend de la aplicación y constituye el punto central de orquestación de los distintos módulos del sistema.

Clasificador de preguntas

Categoriza las consultas del usuario en tres tipos: claras, ambiguas o fuera de contexto. Esta clasificación determina si se procederá a la recuperación de información o si se generará una respuesta alternativa.

Recuperación de respuestas

Se encarga de buscar los fragmentos relevantes en ChromaDB en función de la similitud semántica con la consulta del usuario.

Generador de respuestas

Construye un *prompt* con la pregunta y los fragmentos recuperados, y lo envía al modelo *LLM*, que genera una respuesta basada en ese contexto.

Módulo de descarga de documentos

Automatiza la obtención de documentos desde fuentes institucionales. Recorre los enlaces definidos en un archivo de texto, descarga los archivos PDF y guarda el contenido textual extraído de las páginas web asociadas. Los documentos se organizan en carpetas temáticas para su posterior tratamiento.

Procesador de documentos

Se encarga de leer los documentos descargados, dividirlos en fragmentos textuales y generar sus correspondientes representaciones vectoriales (*embeddings*) mediante un modelo de OpenAI. Estos vectores, junto con los metadatos asociados (tema y fuente), se almacenan en ChromaDB para su posterior recuperación. Este proceso constituye la base del sistema de recuperación semántica.

Módulo de Tests automático

Permite lanzar pruebas automáticas a partir de bancos de preguntas almacenados en una carpeta específica. Para cada grupo de test, ejecuta el sistema con diferentes estrategias de recuperación y guarda las respuestas generadas en archivos CSV, dentro de la carpeta `Resultados/`. Este módulo ha sido clave para comparar objetivamente el comportamiento del sistema en diferentes escenarios y ajustar su rendimiento.

Interfaz Web

Este componente representa la capa de presentación del sistema, accesible desde cualquier navegador moderno. Permite al usuario introducir preguntas en lenguaje natural y visualizar las respuestas generadas por el modelo. Está desarrollado con tecnologías estándar como HTML, CSS y JavaScript, y se comunica con el backend del sistema mediante peticiones HTTP. Su función es servir de intermediario entre el usuario y los componentes internos del sistema, facilitando una experiencia conversacional accesible y sin necesidad de conocimientos técnicos.

La Figura 5.1 resume visualmente los principales módulos funcionales del sistema y las relaciones entre ellos. En el esquema se identifican tanto los componentes internos como los servicios externos utilizados, así como los flujos de comunicación entre ellos. También se representa la interacción del usuario final y del entorno de pruebas.

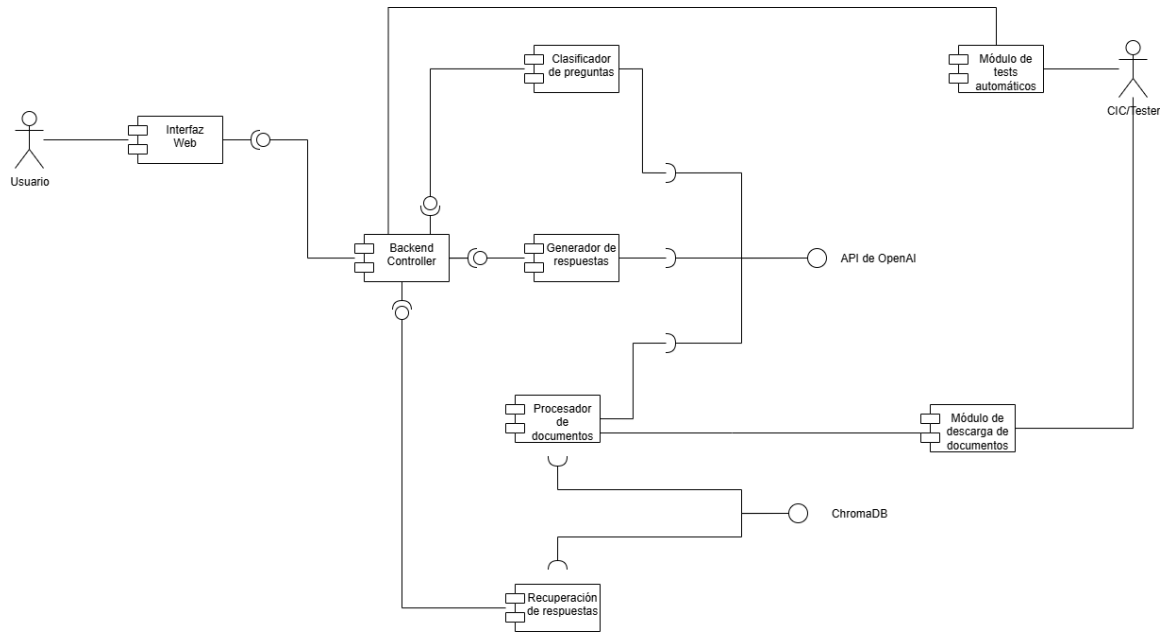


Figura 5.1. Diagrama de componentes del sistema desarrollado.

5.3. Selección del modelo de lenguaje

Durante el diseño del sistema se evaluaron distintas opciones para la generación de respuestas y la clasificación de preguntas. Esta decisión se basó tanto en comparativas académicas como en pruebas prácticas con modelos reales.

Finalmente se optó por utilizar dos modelos distintos de la familia GPT, desarrollados por OpenAI, cada uno orientado a una tarea concreta:

- **GPT-4o-mini**: se emplea para generar las respuestas del sistema. Ofrece un buen equilibrio entre velocidad de ejecución, coste y calidad en tareas de redacción y respuesta basada en contexto.
- **GPT-4**: se utiliza específicamente para la clasificación de preguntas (clara, ambigua, fuera de contexto), ya que ofrece un mayor rendimiento en tareas de comprensión semántica y razonamiento contextual [20].

Se consideraron otras alternativas, como LLaMA (Meta), Claude (Anthropic) o Cohere, pero presentaban ciertas limitaciones. Por ejemplo, los modelos de código abierto requerían una infraestructura más compleja para su despliegue; Claude no ofrecía una integración directa con las herramientas usadas (como LangChain); y Cohere presentaba menor precisión en tareas de recuperación semántica [6], [21].

La elección de los modelos de OpenAI responde a una combinación de factores técnicos, facilidad de integración y fiabilidad contrastada en estudios recientes. Dado que el objetivo principal del proyecto era desarrollar un prototipo funcional que ofreciera respuestas precisas y útiles desde el primer momento, optar por una solución ya entrenada como GPT resultó clave. El uso de su API permite centrar los esfuerzos en el diseño y evaluación de la arquitectura, sin necesidad de entrenar modelos propios ni gestionar infraestructura adicional, lo que encaja perfectamente con los recursos y tiempos para la realización de este PFG.

5.4. Estructura de datos

El sistema utiliza una base vectorial (ChromaDB) para almacenar los fragmentos textuales procesados. Cada documento descargado es dividido mediante un algoritmo de fragmentación recursiva, generando bloques de hasta 4000 caracteres con solapamiento configurable.

Cada fragmento se convierte en un vector semántico mediante un modelo de *embedding* proporcionado por OpenAI. Estos vectores, junto con los metadatos asociados, son almacenados de forma persistente en un directorio local.

Aunque los documentos están organizados físicamente en carpetas según su temática (por ejemplo, *becas*, *matrícula*, *créditos*, etc.), en esta implementación se ha optado por utilizar una única colección en ChromaDB. Esto se ha hecho para simplificar la arquitectura del sistema y reducir la complejidad en el desarrollo y las pruebas, ya que todos los fragmentos quedan igualmente identificados mediante metadatos. Esto permite mantener cierto nivel de organización sin necesidad de separar los datos en colecciones distintas.

Los metadatos incluidos con cada fragmento son:

- `source`: ruta del documento original.
- `tematica`: nombre de la carpeta temática correspondiente.

Esta organización permite al sistema recuperar fragmentos relacionados semánticamente con la consulta del usuario, manteniendo la trazabilidad de su origen. La base se reconstruye en cada ejecución, asegurando la consistencia con el conjunto actual de documentos.

Aunque los metadatos `source` y `tematica` no intervienen directamente en la generación de respuestas, su presencia resulta útil para mantener la trazabilidad del origen de cada fragmento y facilitar análisis posteriores. Durante las pruebas automáticas (ver sección 7), el sistema utiliza el metadato `source` para registrar en los resultados el documento original del que procede el primer fragmento recuperado. El metadato `tematica`, si bien no se ha empleado en

esta versión, podría incorporarse en futuras evaluaciones para analizar el rendimiento del sistema por categorías temáticas o identificar patrones de error asociados a áreas específicas del conocimiento.

Un ejemplo completo del formato en que se almacena cada fragmento, incluyendo el contenido textual, la representación vectorial y los metadatos, puede consultarse en el Listado 5.1.

Listado 5.1. Ejemplo de fragmento almacenado en ChromaDB

```
{
  "id": "frag_017",
  "page_content": "La convocatoria de becas estará abierta hasta el
     15 de julio...",
  "embedding": [0.124, -0.056, ..., 0.981],
  "metadata": {
    "source": "docs/becas/ConvocatoriaBecasGrado2023.pdf",
    "tematica": "becas"
  }
}
```

5.5. Criterios de diseño adoptados

Durante el desarrollo se siguieron los siguientes principios de diseño:

- Modularidad: cada funcionalidad está implementada de forma independiente.
- Escalabilidad: se prevé la incorporación de nuevas fuentes documentales o modelos con facilidad.
- Facilidad de pruebas: la arquitectura permite validar cada módulo por separado.
- Eficiencia: se optimiza el acceso mediante almacenamiento local y consultas semánticas.

5.6. Arquitectura del Sistema

La arquitectura completa del sistema desarrollado se muestra en la Figura 5.2, donde se representan los componentes principales y sus relaciones funcionales. El esquema distingue claramente dos vías de interacción diferenciadas:

- **Acceso del usuario final**: se realiza a través del navegador web, mediante una interfaz que permite enviar preguntas. Esta interfaz se comunica con el backend, que clasifica la consulta, recupera información relevante y genera una respuesta con ayuda de modelos de lenguaje.
- **Acceso del tester o evaluador (CIC)**: se efectúa directamente mediante scripts que interactúan con los módulos internos. Esta vía permite lanzar pruebas automáticas, comparar respuestas y analizar el comportamiento del sistema bajo distintos escenarios.

El esquema también incluye el proceso de mantenimiento documental, compuesto por la descarga de fuentes institucionales, su procesamiento y la construcción de la base vectorial. A su vez, se detalla el recorrido completo de una consulta: desde su entrada hasta la generación final de la respuesta.

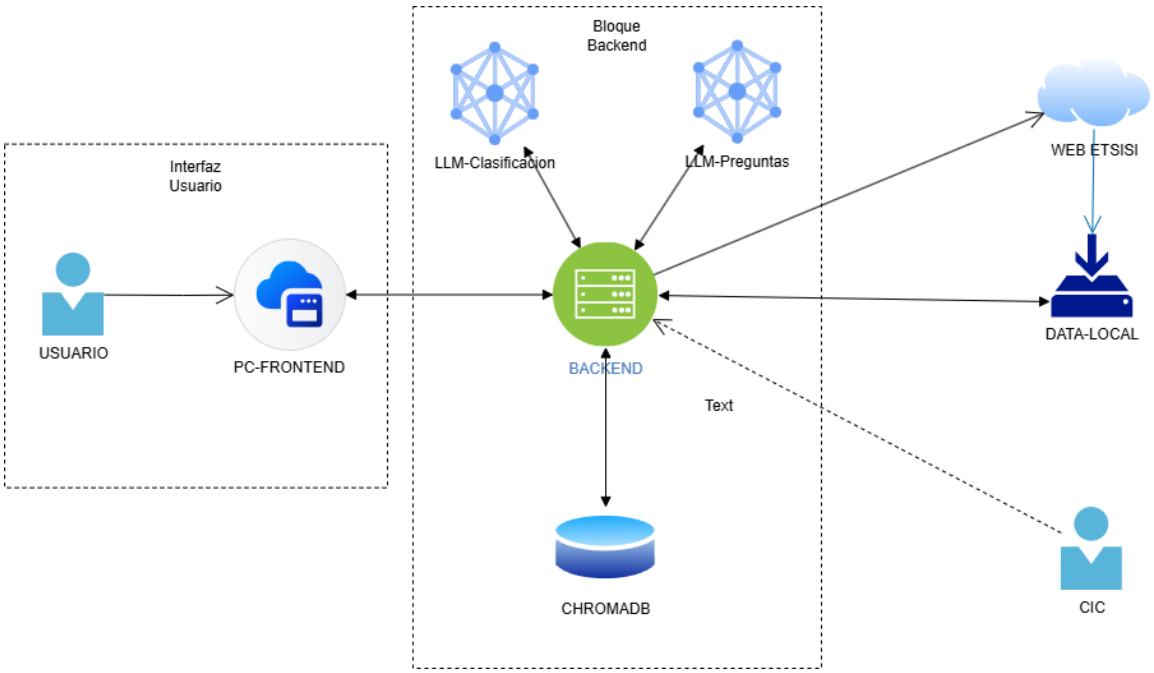


Figura 5.2. Diagrama de arquitectura general del sistema desarrollado.

5.7. Casos contemplados en el diseño y diagramas de secuencia

El sistema contempla tres flujos principales de operación, que han sido modelados mediante diagramas de secuencia para facilitar su análisis y comprensión:

- **Consulta respondible**: el sistema procesa una pregunta clara, recupera fragmentos relevantes y genera una respuesta útil.
- **Consulta ambigua o fuera de contexto**: se detecta que la pregunta no puede ser respondida directamente y se devuelve un mensaje neutro.
- **Carga e indexación de documentos**: se actualiza la base documental mediante la descarga y vectorización de nuevas fuentes.

A continuación se describen estos casos con el apoyo de los diagramas de secuencia diseñados para el proyecto.

La Figura 5.3 muestra el flujo completo de una consulta respondible. El usuario introduce su pregunta a través de la interfaz web. El backend la clasifica como clara, consulta la base vectorial ChromaDB para obtener fragmentos relevantes y construye un *prompt*. Finalmente, el modelo genera una respuesta que se presenta al usuario.

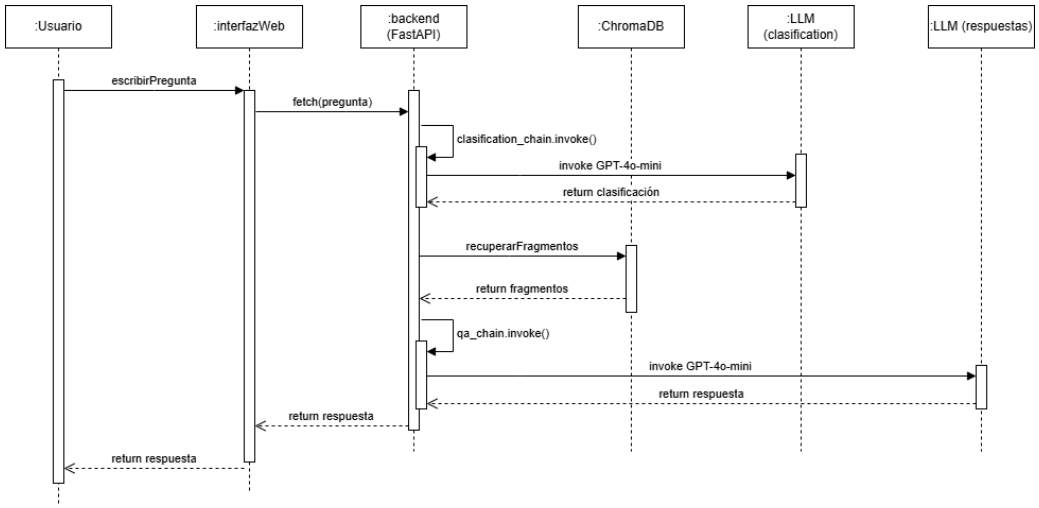


Figura 5.3. Diagrama de secuencia para una pregunta respondible.

La Figura 5.4 ilustra el comportamiento del sistema ante preguntas que no pueden resolverse. Cuando se detecta que la consulta es ambigua o ajena al dominio cubierto por la documentación, el sistema no intenta recuperar información ni generar una respuesta basada en contexto. En su lugar, devuelve un mensaje neutro informando al usuario de la imposibilidad de atender su solicitud.

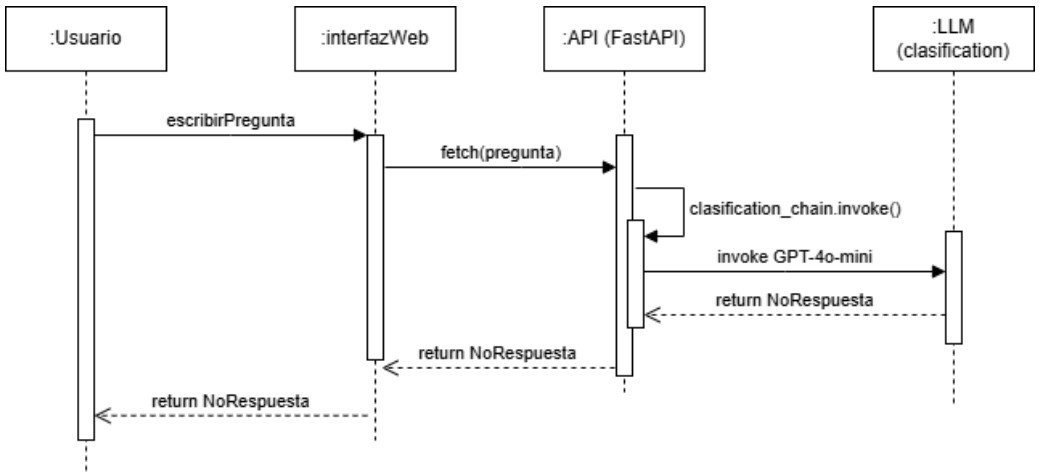


Figura 5.4. Diagrama de secuencia para una pregunta ambigua o fuera de contexto.

La Figura 5.5 representa el proceso automatizado de mantenimiento de la base documental. Este flujo accede a enlaces institucionales, descarga los documentos en PDF, los divide en fragmentos, genera sus embeddings semánticos y los almacena en ChromaDB junto con los metadatos necesarios. Esta tarea resulta fundamental para mantener la capacidad del sistema de recuperar información relevante.

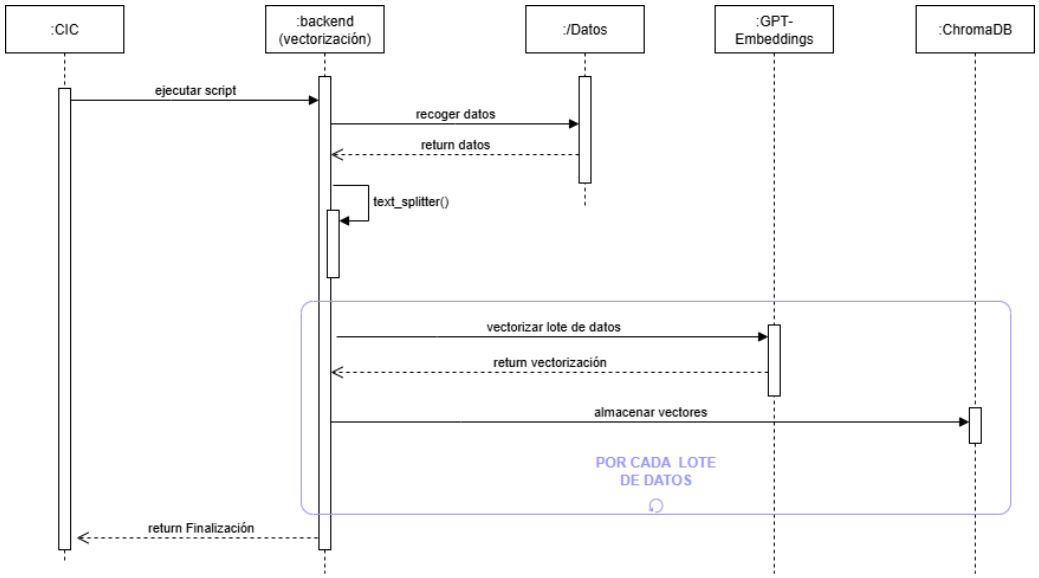


Figura 5.5. Diagrama de secuencia del proceso de descarga e indexación semántica.

La Figura 5.6 muestra el flujo seguido por el módulo de evaluación automática. Este script recorre todos los archivos de prueba, clasifica cada pregunta y genera respuestas utilizando distintas estrategias de recuperación. Los resultados se almacenan automáticamente en archivos CSV, lo que facilita el análisis comparativo del rendimiento del sistema.

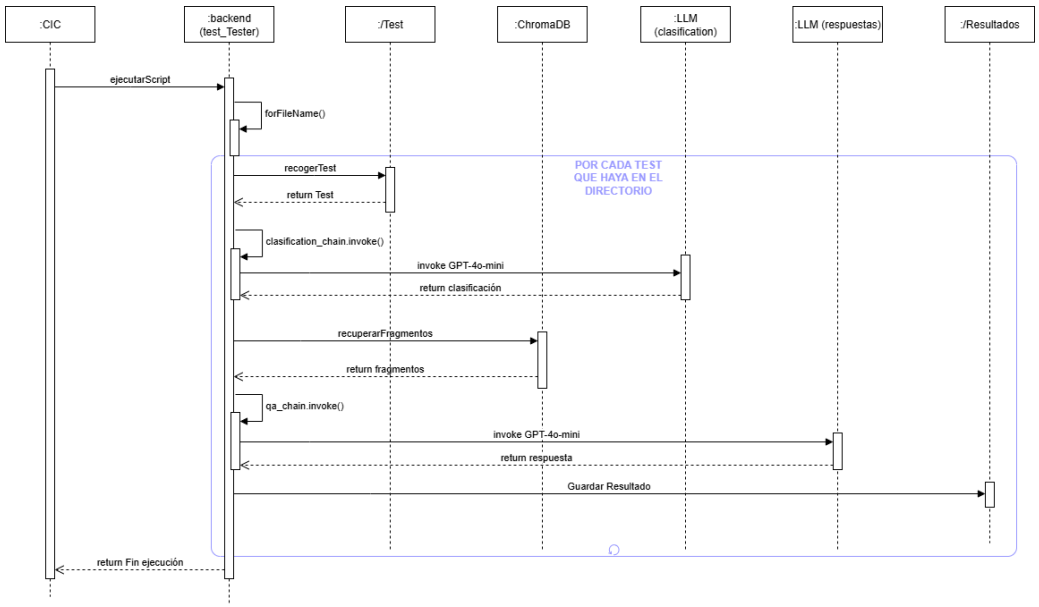


Figura 5.6. Diagrama de secuencia del script de pruebas automáticas.

Estos diagramas no solo documentan el diseño del sistema, sino que también permiten entender cómo se organiza internamente, cómo se conectan

sus módulos y qué decisiones técnicas se han adoptado para cumplir los requisitos funcionales y de eficiencia.

6. Desarrollo del sistema

En este capítulo se describe el desarrollo técnico del sistema conversacional, organizado en torno a cuatro bloques funcionales: descarga de documentos, procesamiento e indexación semántica, y clasificación de la pregunta y generación de respuestas, y el entorno de pruebas automatizadas. Cada módulo se ha desarrollado de forma independiente y validado antes de su integración completa.

6.1. Descarga de documentos

El primer bloque funcional se encarga de automatizar la obtención de información desde fuentes institucionales. Esta parte está implementada en el script *Load_PDFs.py*, que gestiona tanto la descarga de documentos PDF como la extracción del texto visible de las páginas HTML.

El sistema recorre una lista de URLs definidas en un archivo de texto llamado enlaces.txt y, para cada página, realiza las siguientes tareas:

- Accede a la web correspondiente y guarda su contenido textual en un archivo plano.
- Detecta enlaces a archivos PDF y los descarga localmente en carpetas organizadas por temática.

El script ha sido diseñado con mecanismos de tolerancia a fallos. En caso de error SSL, se reintenta la descarga sin verificación de certificados. También se capturan errores generales de red, se gestionan los archivos duplicados y se informa del estado de cada operación sin interrumpir el proceso completo. Este enfoque robusto permite ejecutar la descarga de forma masiva sin necesidad de intervención manual, incluso en presencia de enlaces incorrectos o páginas inaccesibles.

6.2. Procesamiento e indexación semántica

Una vez descargados, los documentos se procesan para ser utilizados por el sistema de recuperación semántica. Este procesamiento no se realiza en el mismo script que la descarga, sino en los módulos posteriores de backend.

El flujo de procesamiento incluye:

- Lectura y carga de los textos extraídos y los PDF descargados.
- Segmentación en fragmentos mediante la clase *RecursiveCharacterTextSplitter* de
- Segmentación en fragmentos mediante la clase *RecursiveCharacterTextSplitter* de *LangChain*, con un tamaño de 2000 caracteres y un solapamiento de 600. Esta configuración fue seleccionada tras realizar pruebas comparativas con distintos valores, buscando un equilibrio entre coherencia contextual y eficiencia en la recuperación (ver capítulo 7)
- Generación de vectores semánticos usando el modelo *text-embedding-ada-002* a través de *OpenAIEmbeddings*.
- Inserción de los fragmentos en una base vectorial persistente utilizando *Chroma*.

Para mejorar la eficiencia del proceso, los fragmentos se insertan en la base en bloques de 100 elementos mediante la función *add_texts*, lo que reduce el número de operaciones individuales y mejora el rendimiento.

Cada fragmento se guarda junto con metadatos que permiten mantener trazabilidad, como la fuente original y la temática. Antes de cada nueva carga, la base se reconstruye por completo para garantizar la coherencia con los documentos actuales.

6.3. Clasificación de preguntas y generación de respuestas

El sistema emplea dos *prompts* bien diferenciados para tratar cada consulta: uno para la clasificación inicial y otro para la generación de respuesta. Esta lógica se utiliza tanto en el funcionamiento del sistema interactivo como en el entorno de pruebas.

- El **prompt de clasificación** (ver Figura 6.1) proporciona instrucciones explícitas para que el modelo categorice las preguntas según su grado de claridad. Se define el significado de cada etiqueta ("CLARA", "AMBIGUA", "FUERA DE CONTEXTO" o "NO TENGO INFORMACIÓN SUFICIENTE") con ejemplos implícitos, y se pide al modelo que devuelva exclusivamente una de esas tres palabras, en mayúsculas y sin explicaciones adicionales. Esto permite aplicar una clasificación robusta que luego condiciona la lógica de recuperación o respuesta.
- El **prompt de generación de respuestas** (ver Figura 6.2) impone una restricción fundamental: el modelo debe responder únicamente utilizando el contexto proporcionado. Si no encuentra la información necesaria, debe devolver una respuesta fija: "No tengo información suficiente en los documentos proporcionados." Esta formulación evita que el modelo genere contenido inventado ("alucinaciones"), garantizando que toda respuesta esté anclada en fragmentos documentales reales.

  Adicionalmente, en caso de ser ambigua, se le indica como instrucción explícita que solo responda a la última pregunta que se le hizo, recapitulando y usando la pregunta anterior como contexto.

```
clasificacion_prompt = PromptTemplate(
    input_variables=["pregunta"],
    template="""
Eres un asistente que clasifica preguntas realizadas por estudiantes dentro del ámbito universitario.

Clasifica la siguiente pregunta en una de estas tres categorías, siguiendo estas instrucciones:
- CLARA: si la pregunta se entiende por sí sola y puede responderse directamente en el contexto de una universidad
- AMBIGUA: si la pregunta no está clara por sí sola, tiene múltiples posibles significados, depende de una convers
- FUERA DE CONTEXTO: si no está relacionada con la universidad, estudios, clases o procedimientos académicos (por

Pregunta: "{pregunta}"

Devuelve solo una de estas palabras en mayúsculas: CLARA, AMBIGUA o FUERA DE CONTEXTO.
"""
```

Figura 6.1. Prompt utilizado para clasificar las preguntas según su claridad.

```
qa_prompt = PromptTemplate(
    input_variables=["context", "question"],
    template="""
Usando únicamente el siguiente contexto, responde la pregunta de forma concisa.
Si la respuesta no se encuentra en el contexto, responde: "No tengo información suficiente en los documentos proporcionados."

Contexto: {context}

Pregunta: {question}

Respuesta:
"""
)
```

Figura 6.2. Prompt utilizado para generar respuestas a partir del contexto recuperado.

El comportamiento ante preguntas ambiguas se gestiona mediante una lógica condicional basada en la existencia o no de memoria conversacional previa. En concreto, si una pregunta es clasificada como ambigua y existe una conversación anterior válida (es decir, una pregunta anterior clara con una respuesta válida), el sistema concatena ambas preguntas y las trata como una sola, intentando predecir el significado completo a partir del contexto anterior. Si la respuesta obtenida no contiene información suficiente, la memoria se borra para evitar conversaciones incorrectas en futuras respuestas.

En caso de que la pregunta ambigua no venga precedida por una conversación clara (es decir, si no hay memoria disponible), el sistema informa al usuario de que su pregunta es demasiado imprecisa y solicita que la reformule. Por otro lado, si la pregunta es clasificada como fuera de contexto (por ejemplo, sobre un tema no relacionado con los documentos académicos), el sistema lo comunica de forma directa y descarta cualquier posible memoria.

Además, cuando una pregunta clara obtiene una respuesta válida, el sistema guarda tanto la pregunta como la respuesta en memoria. Si, por el contrario, no se recupera información útil, se considera que no se dispone de contexto válido y la memoria se elimina.

Se puede ver la lógica de la conversación en el Anexo A.

6.4. Entorno de pruebas automatizadas

Este bloque funcional corresponde a la fase de pruebas y evaluación del sistema, desarrollada en el script `test_Tester.py`. Este módulo permite simular interacciones con el asistente, procesar preguntas clasificadas y registrar resultados para su análisis posterior.

El script explora automáticamente todos los archivos presentes en una carpeta llamada `Test/`, ejecutando los experimentos con ambas estrategias de recuperación (MMR y Similarity) para cada conjunto de preguntas.

El proceso general que sigue el entorno de pruebas es el siguiente:

- Clasifica cada pregunta como clara, ambigua o fuera de contexto mediante el *prompt* de clasificación.
- Si la pregunta es clara o ambigua con contexto anterior válido, se realiza la recuperación de fragmentos desde ChromaDB.
- Se genera la respuesta utilizando el contexto y la pregunta.
- El resultado se guarda en un archivo `.csv` dentro de la carpeta `Resultados/`, incluyendo la estrategia utilizada, la clasificación, los fragmentos recuperados y la respuesta generada.

Durante el desarrollo se probaron dos métodos distintos de recuperación de fragmentos: por similitud semántica directa (*similarity*) y mediante diversidad con relevancia marginal máxima (*Maximum Marginal Relevance*, MMR). El sistema evalúa ambos métodos de recuperación de fragmentos, generando un reporte para su futuro análisis. Tras analizar la calidad de los fragmentos obtenidos y la coherencia de las respuestas generadas, se optó por dejar

MMR como estrategia por defecto en la versión final del sistema, al ofrecer resultados más diversos y relevantes.

Los resultados de cada ejecución se almacenan en la carpeta `Resultados/` en formato `.csv`, diferenciando los ficheros por estrategia utilizada. Esto permite comparar el rendimiento de forma directa sin necesidad de procesar manualmente los datos. Las preguntas de entrada se leen desde ficheros organizados por tipo en la carpeta `Test/`, lo que permite mantener una estructura clara y reutilizable de los bancos de pruebas.

6.5. Funcionamiento práctico del sistema

El sistema ha sido diseñado para ejecutarse en entorno local y facilitar su uso sin necesidad de modificar el código fuente. Para ello, se asume una estructura mínima de carpetas situada en el ESCRITORIO del usuario, donde deben colocarse los elementos principales del proyecto.

Esta organización permite que cualquier usuario pueda ejecutar el sistema en su propio equipo sin preocuparse por rutas absolutas o configuraciones específicas, siempre que mantenga la misma estructura. Todo el flujo se realiza de forma local, sin conexión a una base remota ni despliegue en la nub.

6.5.1. Funcionamiento de la página web

Ante todo esto, se trata de un prototipo pensado para su inclusión y colaboración con los servicios encargados de manejar y mantener la página Moodle de la UPM.

La interfaz web del sistema ha sido diseñada para resultar cómoda y fácil de usar desde cualquier navegador. Al cargar la página de Moodle e iniciar sesión, el usuario ve un pequeño icono con forma de asistente en la esquina inferior derecha (ver Figura 6.3). Este icono funciona como botón de activación: al hacer clic sobre él, se abre la ventana de conversación.

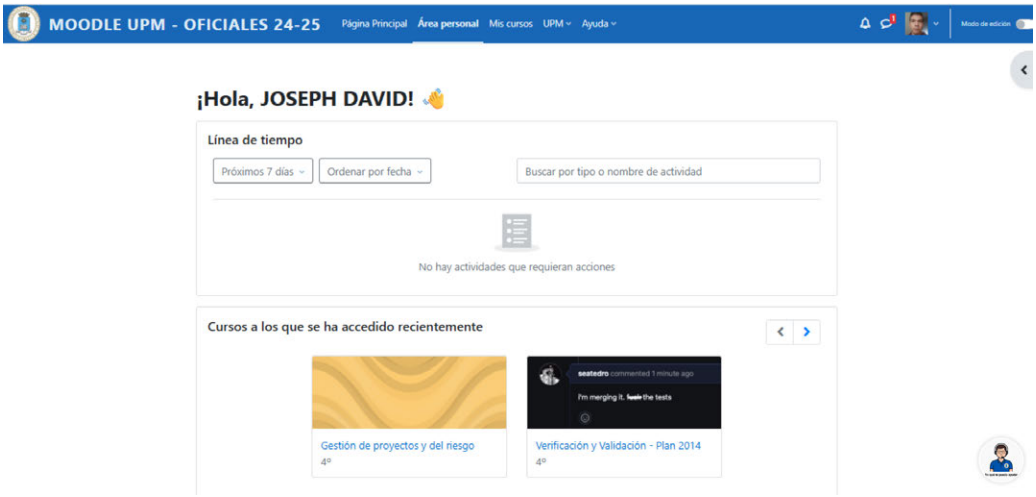


Figura 6.3. Vista inicial: icono del asistente antes de comenzar la conversación.

Una vez desplegada la ventana, el usuario puede escribir su pregunta en el cuadro de texto y pulsar el botón Enviar (ver Figura 6.4). En ese momento, el sistema procesa la consulta de forma automática: clasifica el tipo de pregunta, recupera la información relevante (si procede) y genera una respuesta.

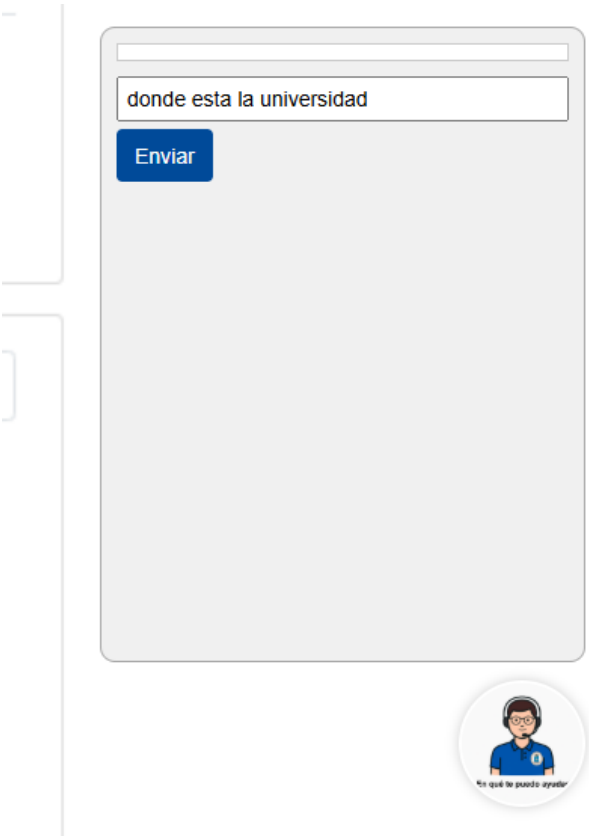


Figura 6.4. Interfaz de conversación con campo de entrada para preguntas.

La respuesta aparece justo debajo, en la misma ventana, manteniendo la estructura de conversación (ver Figura 6.5). El usuario puede seguir preguntando de forma natural, y el sistema recuerda la última interacción para poder aclarar dudas ambiguas cuando sea necesario.

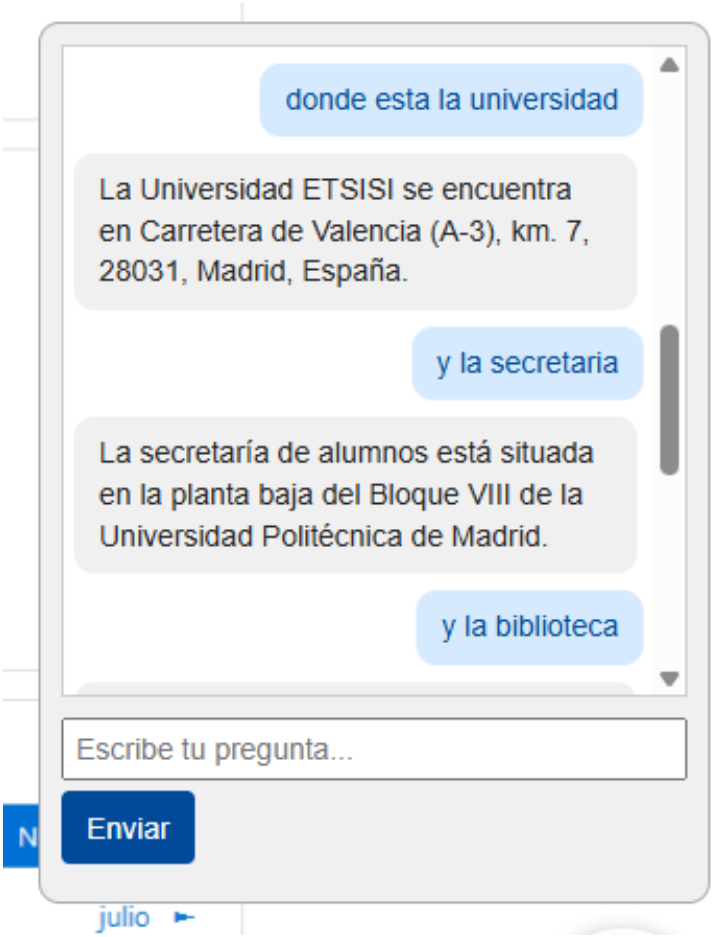


Figura 6.5. Ejemplo de conversación con varias preguntas seguidas.

Este diseño permite que cualquier estudiante pueda interactuar con el asistente sin necesidad de conocimientos técnicos, y obtener respuestas rápidas sobre los procedimientos académicos de la universidad.

6.5.2. Funcionamiento del script tester

El script `test_Tester.py` automatiza la ejecución de pruebas sobre el sistema utilizando distintos bancos de preguntas almacenados en archivos de texto. Su funcionamiento se basa en dos directorios:

- **Test**: carpeta que contiene archivos `.txt` con las preguntas de cada prueba. Cada archivo representa un bloque de test temático, como se muestra en la Figura 6.6. Esta carpeta puede ser modificada libremente por el evaluador del sistema: es posible añadir nuevos ficheros con conjuntos de preguntas personalizadas o eliminar aquellos que no se deseen ejecutar. De este modo, el script se adapta fácilmente a distintas necesidades de análisis sin requerir cambios en el código.
- **Resultados**: carpeta donde se guardan automáticamente los resultados generados, en formato CSV, diferenciando entre las estrategias `similarity` y `mmr` (ver Figura 6.7).

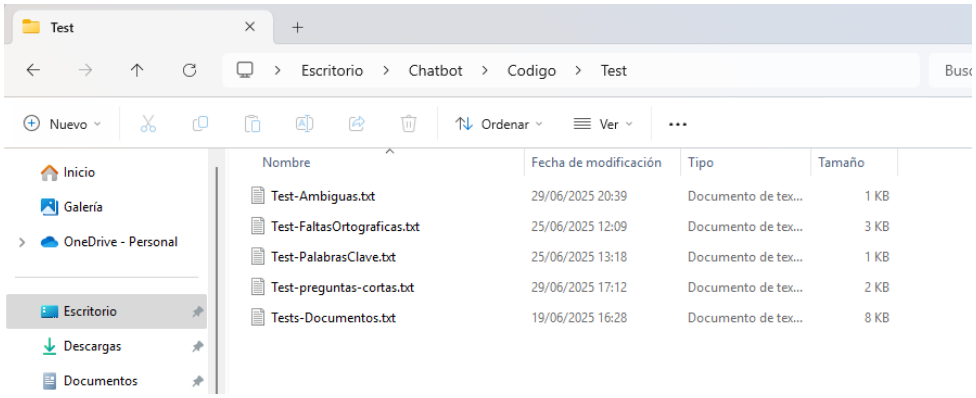


Figura 6.6. Carpeta `Test/` con los archivos de entrada utilizados en las pruebas.

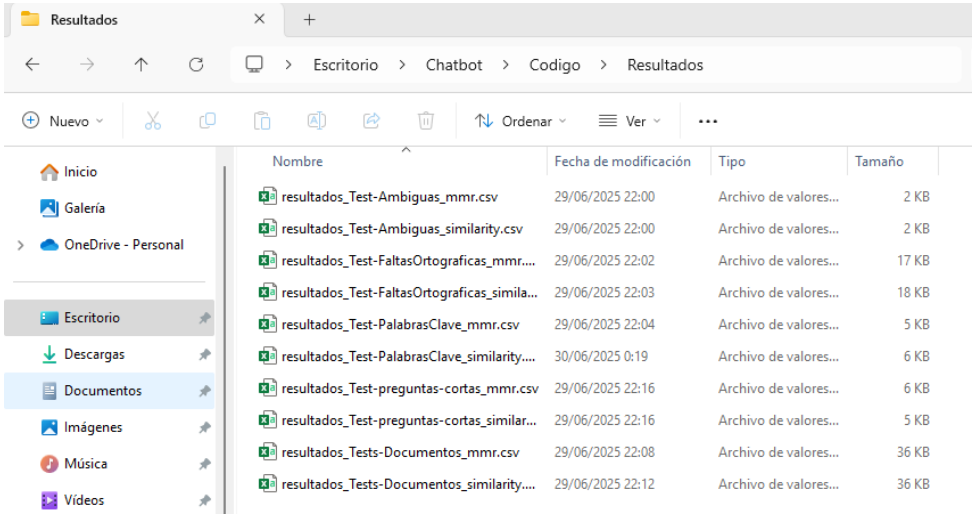


Figura 6.7. Carpeta `Resultados/` con los archivos CSV generados automáticamente por el sistema.

Para cada archivo de prueba en la carpeta `Test/`, el script lanza dos ejecuciones: una con estrategia `mmr` y otra con `similarity`. Los resultados se guardan en `Resultados/` bajo un nombre identificativo, lo que permite compararlos fácilmente y analizarlos posteriormente sin intervención manual.

Este flujo automatizado ha sido clave para evaluar de manera sistemática el comportamiento del sistema en distintos escenarios, incluyendo preguntas claras, ambiguas, con errores ortográficos o con palabras clave aisladas.

[1]

[1] El repositorio completo del sistema se encuentra disponible en GitHub: https://github.com/JosephDavidTorres/Chatbot/tree/master

7. Pruebas y análisis de resultados

Este capítulo recoge las pruebas realizadas para evaluar el funcionamiento del sistema,específicamente la generación de respuestas. Se ha prestado especial atención a comparar las dos estrategias de búsqueda semántica implementadas: *Similarity* y *Maximum Marginal Relevance* (MMR).

7.1. Objetivo y enfoque de las pruebas

El propósito de las pruebas ha sido comprobar la capacidad del sistema para:

- Recuperar los fragmentos más relevantes ante consultas reales.
- Generar respuestas coherentes y fundamentadas.
- Clasificar correctamente preguntas ambiguas o fuera de contexto.
- Evaluar el impacto de distintos parámetros de fragmentación del texto.
- Comparar el rendimiento entre las estrategias *MMR* y *Similarity*.

Para ello se han diseñado distintos bloques de test, cada uno orientado a un escenario representativo del uso real del sistema: preguntas temáticas y evaluación de fragmentación, preguntas con errores ortográficos, entradas ambiguas o consultas formadas solo por palabras clave. Cada bloque incluye un conjunto de preguntas adaptado a su objetivo concreto.

Antes de realizar estas pruebas temáticas, se llevó a cabo una prueba inicial para analizar el comportamiento del sistema frente a distintas configuraciones de fragmentación (*chunk size* y *overlap*), así como con los dos métodos de recuperación implementados. A partir de los resultados obtenidos

en dicha prueba, se decidió adoptar la combinación de **2000 caracteres de fragmento y 600 de solapamiento** como configuración base para el resto de tests.

La justificación detallada de esta decisión se recoge en la Sección 7.2, donde se comparan cuantitativamente las diferentes configuraciones evaluadas.

7.2. Evaluación de configuraciones de fragmentación y modelo con un banco de preguntas temáticas

Este bloque recoge la prueba inicial realizada para determinar qué combinación de parámetros de fragmentación y estrategia de recuperación ofrecía mejores resultados globales. A partir de esta evaluación, se estableció la configuración base utilizada en el resto de pruebas temáticas del sistema.

Resultados obtenidos. Se probaron tres configuraciones distintas de fragmentación del texto:

- Fragmentos de 1000 caracteres con un solapamiento de 300.
- Fragmentos de 2000 caracteres con un solapamiento de 600.
- Fragmentos de 4000 caracteres con un solapamiento de 1000.

Cada una se evaluó utilizando las dos estrategias de recuperación implementadas: *Similarity* y *Maximum Marginal Relevance* (MMR). Todas las combinaciones fueron probadas sobre el mismo conjunto de 78 preguntas temáticas, diseñadas para simular consultas reales del entorno universitario. Las preguntas utilizadas en esta prueba están recogidas en el Anexo C.

Tabla 7.1. Comparativa global de configuraciones probadas

Método	Chunk / Overlap	Aciertos	Errores	Precisión
MMR	1000 / 300	63	15	80,8 %
MMR	2000 / 600	66	12	84,6 %
MMR	4000 / 1000	66	12	84,6 %
Similarity	1000 / 300	63	15	80,8 %
Similarity	2000 / 600	67	11	85,9 %
Similarity	4000 / 1000	62	16	79,5 %

Como muestra la Tabla 7.1, la configuración con fragmentos de 2000 caracteres y solapamiento de 600 ofreció el mejor rendimiento global. Esta combinación permitió mantener suficiente contexto para generar respuestas útiles, sin introducir demasiada información irrelevante, como sucede con fragmentos más largos.

Durante el análisis se observó que, independientemente de los parámetros utilizados, los errores tienden a concentrarse en ciertas temáticas: especialmente **cambios de titulación**, **reconocimiento de créditos** y **calendario académico**. Estas áreas suelen contener información dispersa, normativa ambigua o mal estructurada, lo que dificulta su recuperación precisa.

Como conclusión de esta prueba, se adopta la configuración **2000 / 600** como base para todas las pruebas temáticas posteriores, así como para la versión final del sistema.

Además,ambas estrategias, *MMR* y *Similarity*, mostraron un rendimiento muy similar, con solo una o dos respuestas de diferencia entre sus mejores configuraciones. En concreto, *Similarity* fue la que alcanzó el mayor número de aciertos (67 de 78), pero la ventaja frente a *MMR* no fue significativa.

A pesar de sus diferencias, ambas estrategias fallaron en preguntas similares, las mismas mencionadas anteriormente.

En resumen, ambas estrategias son válidas y funcionales. Aunque en esta prueba concreta *Similarity* obtuvo un resultado ligeramente superior, la diferencia no es lo suficientemente clara como para descartar por completo el uso de *MMR* en otros contextos.

7.3. Evaluación con preguntas con faltas ortográficas

Se ha evaluado la capacidad del sistema para manejar consultas que contienen errores ortográficos. Para ello, se formularon 20 pares de preguntas: una con errores de escritura y otra corregida. Las preguntas utilizadas en esta prueba están recogidas en el Anexo C. El objetivo fue comprobar si ambas versiones obtenían respuestas equivalentes, utilizando las dos estrategias de recuperación implementadas: *Similarity* y *Maximum Marginal Relevance* (MMR).

Cada respuesta con faltas fue comparada con su versión correcta y clasificada en tres niveles:

- **Idéntica**: la respuesta es exactamente la misma en ambos casos.
- **Parecida**: la respuesta con errores transmite la misma información de forma aproximada.
- **Diferente**: la respuesta con errores es incorrecta o indica falta de información.

Tabla 7.2. Resultados obtenidos en la prueba con errores ortográficos

Tipo de coincidencia	MMR	Similarity
Idéntica	14 (70 %)	12 (60 %)
Parecida	3 (15 %)	5 (25 %)
Diferente	3 (15 %)	3 (15 %)
Total	20	20

Los resultados se resumen en la Tabla, ofreciendo el porcentaje y el índice de éxito que ha tenido cada estrategia 7.2, y dando lugar al siguiente análisis:

MMR mostró una buena capacidad de generalización, con un 85 % de respuestas útiles (idénticas o parecidas). Fue la única estrategia que ofreció un mayor número de coincidencias exactas, aunque también falló en tres ocasiones al no recuperar información adecuada.

Similarity, aunque no generó tantas coincidencias exactas, demostró una notable tolerancia a las faltas. Su rendimiento fue más uniforme, con menor sensibilidad al deterioro de la entrada, lo que la convierte en una opción más robusta ante errores tipográficos.

Ambas estrategias lograron un 85 % de respuestas útiles, pero con perfiles distintos: *MMR* se muestra más precisa, mientras que *Similarity* es más flexible semánticamente. En este escenario se prefiere MMR, ya que proporciona más garantías de respuesta incluso con consultas mal escritas.

7.4. Evaluación con preguntas ambiguas

Este bloque de pruebas tiene como objetivo comprobar si el sistema es capaz de identificar preguntas ambiguas, es decir, consultas que no pueden responderse directamente por falta de contexto o porque pueden tener múltiples interpretaciones. Para ello, se diseñaron 10 preguntas intencionadamente poco precisas o demasiado generales. Las preguntas utilizadas en esta prueba están recogidas en el Anexo C.

Este tipo de pruebas van dedicadas al sistema de clasificación, sin embargo, se utilizarán estrategias de recuperación en el caso de que generen una respuesta.

Las pruebas se realizaron utilizando ambas estrategias de recuperación: *Similarity* y *Maximum Marginal Relevance* (MMR). El sistema debe clasificar adecuadamente estas consultas y evitar generar respuestas inventadas o sin fundamento.

Tabla 7.3. Preguntas ambiguas correctamente identificadas

Estrategia	Total de preguntas	Preguntas clasificadas como ambiguas
MMR	10	10
Similarity	10	10

La Tabla 7.3 recoge el número de preguntas detectadas correctamente como ambiguas por cada estrategia, indicando que ambas identificaron correcta-

mente las 10 preguntas como ambiguas. En todos los casos, el sistema respondió con un mensaje que sugiere reformular la pregunta o proporcionar más información. Esto confirma que el clasificador de preguntas está funcionando adecuadamente y no se ve afectado por el método de recuperación utilizado.

7.5. Evaluación con preguntas de solo palabras clave

Este bloque de pruebas evalúa si el sistema es capaz de clasificar correctamente preguntas ambiguas cuando el usuario introduce únicamente una o varias palabras clave sin formular una consulta completa. Este tipo de entradas son frecuentes en asistentes conversacionales, especialmente cuando el usuario no sabe cómo expresar con claridad su duda. Las preguntas utilizadas en esta prueba están recogidas en el Anexo C.

El experimento se realizó ejecutando dos iteraciones completas del sistema, una con la estrategia de recuperación *MMR* y otra con *Similarity*, manteniendo la misma configuración técnica y los mismos ficheros de entrada. Aunque el objetivo principal era observar la clasificación, también se analizaron las diferencias en la respuesta generada cuando el sistema decidía no marcar una pregunta como ambigua.

Tabla 7.4. Variabilidad en la clasificación de preguntas por palabras clave

Iteración	Total de preguntas	Clasificadas como no ambiguas
MMR – Iteración 1	35	4
MMR – Iteración 2	35	6

La tabla 7.4 muestra que, aunque la mayoría de las preguntas fueron correctamente clasificadas como ambiguas, se produjeron ligeras variaciones entre las ejecuciones, a pesar de usar la misma configuración temperature=0. Esto se debe a la naturaleza estocástica de los modelos generativos, que pueden interpretar de forma diferente preguntas muy breves o poco informativas.

Estas pequeñas fluctuaciones no comprometen la estabilidad general del sistema, pero conviene tenerlas en cuenta en entornos de producción, ya que

pueden impactar en el uso de recursos al activar innecesariamente la recuperación de contexto. No obstante, el sistema se comporta de forma razonablemente estable y predecible, incluso ante entradas mínimas.

7.6. Conclusión general

Tras ejecutar los distintos bloques de pruebas y analizar sus resultados, se ha optado por establecer *Maximum Marginal Relevance (MMR)* como la estrategia de recuperación principal del sistema.

Aunque *Similarity* ha mostrado un rendimiento competente y una cierta tolerancia ante errores de entrada, MMR ha ofrecido mejores resultados en aspectos clave: mayor número de respuestas idénticas en pruebas con errores ortográficos, fragmentos más diversos y relevantes, y una respuesta más precisa en preguntas complejas.

Si bien en algunas pruebas se observaron pequeñas variaciones entre ejecuciones (como en las preguntas de palabras clave), esta inestabilidad no es exclusiva de un método, sino inherente al funcionamiento de los modelos generativos. No obstante, optar por una estrategia más precisa reduce el riesgo de respuestas incompletas o inconsistentes.

Por todo ello, MMR ha sido seleccionada como la configuración base del sistema final, al proporcionar mejores garantías de calidad, utilidad y fiabilidad en un entorno conversacional realista.

Finalmente siguiendo otro tema, en cuanto a los documentos cargados, al observar los errores cometidos en los distintos bloques temáticos, se aprecia una mayor dificultad en las categorías de calendario académico y cambios de titulación, donde la información está dispersa o poco estructurada. Esto sugiere que el sistema podría beneficiarse de un preprocesamiento adicional en estas áreas, o de un refuerzo en la recuperación semántica.

Cabe destacar que todos los resultados obtenidos durante la fase de pruebas se encuentran almacenados en la carpeta Resultados, organizada según el nivel de fragmentación empleado (4000, 2000 y 1000 caracteres). Dado que la configuración final seleccionada fue la de 2000 caracteres con 600 de so-

lapamiento, los resultados completos de los distintos bloques de pruebas se encuentran en dicha subcarpeta. Por su parte, las carpetas correspondientes a las fragmentaciones de 4000 y 1000 caracteres contienen únicamente los resultados utilizados en la fase comparativa inicial para determinar la configuración óptima del sistema y de las pruebas.

8. Rendimiento y coste del sistema

Este capítulo recoge una estimación del rendimiento técnico del sistema y su coste económico aproximado durante su ejecución. Aunque el sistema ha sido desarrollado como un prototipo funcional en entorno local, el uso de modelos de lenguaje mediante la API de OpenAI implica cierto consumo computacional y económico, lo cual resulta relevante para evaluar su viabilidad a medio plazo.

8.1. Tiempos de ejecución

El comportamiento del sistema ha sido fluido durante su uso, tanto en pruebas automatizadas como en interacción directa desde la interfaz web. A modo de resumen:

- El arranque inicial de la página web tarda aproximadamente **1 minuto**, incluyendo la carga de modelos, base de datos y configuración del entorno.
- Una consulta individual desde el navegador tarda entre **2 y 4 segundos** en resolverse, en función de si necesita clasificación, recuperación y generación.
- El conjunto completo de pruebas realizadas (incluyendo tests temáticos, preguntas con errores ortográficos, ambiguas y por palabras clave) se ejecutó en unos **11 minutos**.

Este rendimiento es adecuado para un sistema orientado a asistencia pun-

tual en un entorno académico, y permite su uso continuado sin esperas excesivas para el usuario.

8.2. Coste económico estimado

Durante las pruebas, el sistema hizo uso de dos modelos de lenguaje distintos mediante la API de OpenAI:

- **GPT-4**: utilizado para la clasificación de preguntas (clara, ambigua o fuera de contexto).
- **GPT-4o-mini**: utilizado para la generación de respuestas con contexto documental.

La ejecución de todos los bloques de pruebas supuso un coste total de aproximadamente **0,26 €**. Esta cifra incluye tanto las tareas de clasificación como las de generación.

En un uso cotidiano, como el ofrecido por la interfaz web, se ha observado que el coste es muy bajo. De media, el sistema puede generar hasta **tres respuestas por menos de un céntimo de euro**, lo que lo convierte en una solución aceptable y a ser considerada; sin embargo, para un uso a futuro o reducción de costes, es viable el estudio de otros modelos como Llama.

8.3. Coste total del proyecto

Durante todo el desarrollo del Trabajo de Fin de Grado, se habilitó un crédito de 10 € para el uso de la API de OpenAI. De ese importe, se ha utilizado un total de **6,05 €**, lo que cubre todas las pruebas, interacciones, ajustes de prompts, clasificación y generación realizadas a lo largo del proyecto.

Este gasto reducido refuerza la viabilidad del sistema como prototipo funcional, con bajo coste operativo incluso durante procesos intensivos de pruebas.

El estado final del saldo puede consultarse en el Anexo B.

9. Impacto social y medioambiental

El desarrollo de este sistema conversacional no solo responde a un reto técnico, sino que también tiene implicaciones directas en el ámbito social y medioambiental. Este tipo de soluciones digitales, además de mejorar el acceso a la información, pueden contribuir positivamente al bienestar de la comunidad universitaria y a la sostenibilidad de los procesos internos. Todo ello encaja con los objetivos que recoge el *Plan de Sostenibilidad Ambiental de la Universidad Politécnica de Madrid (UPM) 2021–2025* [22].

Impacto social

Desde el punto de vista social, el sistema facilita que cualquier estudiante pueda resolver dudas académicas sin necesidad de recurrir siempre al personal de administración o buscar entre documentos complejos. Esto tiene varios beneficios:

- Mejora el acceso a la información, especialmente para estudiantes de nuevo ingreso o en situaciones en las que no pueden acudir presencialmente a la universidad.
- Reduce la carga de trabajo en los servicios administrativos, al automatizar respuestas a preguntas frecuentes.
- Favorece la equidad, ya que cualquier persona, desde cualquier dispositivo y en cualquier momento, puede obtener la misma información de forma clara.

Además, se ha tenido especial cuidado en evitar respuestas incorrectas o in-

ventadas, limitando la generación del modelo únicamente al contenido extraído de los documentos oficiales. Esto refuerza la confianza y fiabilidad del sistema.

Impacto medioambiental

Aunque el impacto ambiental directo de este tipo de aplicaciones es reducido, sí puede aportar mejoras en sostenibilidad a medio plazo:

- Al facilitar el acceso digital a la información, se reduce la necesidad de impresiones en papel y visitas físicas a la secretaría.
- El sistema funciona de forma local, sin servidores desplegados de forma continua ni servicios en la nube activos permanentemente, lo que reduce el consumo energético.
- Al estar diseñado de forma modular, se puede reutilizar en otras facultades o servicios sin necesidad de desarrollar nuevos sistemas desde cero.

Este tipo de soluciones digitales están alineadas con las estrategias institucionales que promueven la eficiencia, la digitalización y el respeto al entorno, tal como recoge el plan de sostenibilidad de la universidad [22].

Reflexión final

Proyectos como este demuestran que la tecnología puede tener un impacto más allá del rendimiento técnico. Facilitar el acceso a la información, reducir tiempos y recursos, y mejorar la experiencia del estudiante son pequeños avances que, en conjunto, aportan valor real a la universidad. Integrar esta perspectiva social y medioambiental desde el diseño es un paso necesario hacia una universidad más accesible, eficiente y comprometida con su entorno.

10. Conclusiones

Este *Trabajo de Fin de Grado* ha sido una experiencia muy enriquecedora tanto a nivel académico como personal. El objetivo principal era construir un sistema conversacional que ayudara a los estudiantes de la ETSISI a resolver dudas habituales sobre la universidad, y puedo decir que se ha logrado desarrollar un prototipo funcional que cumple con esa misión.

A lo largo del proyecto he tenido la oportunidad de aplicar muchos de los conocimientos aprendidos durante la carrera: desde el procesamiento de lenguaje natural y el uso de modelos de lenguaje, hasta el diseño modular de sistemas, el trabajo con bases de datos vectoriales y la implementación de pruebas automatizadas. Todo ello ha sido posible utilizando herramientas actuales y accesibles, como las APIs de OpenAI, LangChain o ChromaDB.

Durante las pruebas se ha comprobado que el sistema es capaz de generar respuestas útiles y coherentes, identificar preguntas ambiguas y adaptarse a diferentes tipos de consultas. Comparando distintas estrategias de recuperación, como *MMR* y *Similarity*, se ha llegado a conclusiones prácticas que han servido para mejorar la calidad del sistema final.

Aunque se trata de un prototipo, el resultado es una herramienta funcional que podría ser útil en un entorno real, y que abre la puerta a futuras mejoras e integraciones más completas dentro de la universidad. Me quedo especialmente satisfecho con haber conseguido que todo el sistema funcione en conjunto: desde la descarga y organización de documentos, hasta la interacción conversacional desde el navegador.

La inteligencia artificial es, sin duda, una herramienta muy potente que puede facilitar muchas tareas del día a día. Usada correctamente, no solo puede mejorar la eficiencia de los procesos, sino también permitir nuevas formas de acceder a la información y tomar decisiones más informadas. En particular, los modelos de lenguaje como los utilizados en este proyecto permiten procesar grandes volúmenes de texto de forma casi instantánea, algo que

manualmente requeriría mucho más tiempo y esfuerzo.

Más allá de generar respuestas, también permiten transformar la forma en la que nos relacionamos con los datos, evaluando la calidad de las respuestas, detectando errores o guiando al usuario para mejorar su propia consulta. Esto convierte a los *chatbots* no solo en asistentes, sino en herramientas activas de mejora continua.

Este trabajo es solo una pequeña muestra de lo que pueden aportar estas tecnologías en el ámbito académico. Aunque aún queda camino por recorrer, estoy convencido de que soluciones como esta pueden tener un gran impacto, no solo en universidades, sino también en empresas u organizaciones que quieran hacer más accesible su información interna.

Y lo más importante: no hay un único camino. En inteligencia artificial, cada decisión —desde la elección del modelo hasta el diseño de los *prompts*— abre nuevas posibilidades. Por eso, más allá del resultado concreto, lo valioso ha sido el proceso de exploración y aprendizaje que me ha permitido construir una solución propia desde cero.

11. Trabajo futuro

El sistema desarrollado ha alcanzado los objetivos definidos para este *Proyecto de Fin de Grado*, y ha permitido construir un prototipo funcional y útil. Sin embargo, debido a las limitaciones de tiempo, recursos y acceso a ciertas herramientas, algunas funcionalidades quedaron fuera del alcance del desarrollo actual. En esta sección se describen posibles líneas de mejora y ampliación que podrían abordarse en versiones futuras del sistema.

11.1. Acceso personalizado e integración con Moodle

Una de las ideas iniciales era que el sistema estuviera conectado con la plataforma Moodle de la UPM, permitiendo que cada estudiante pudiera acceder mediante su cuenta universitaria. Sin embargo, no se pudo implementar un sistema de autenticación real, por lo que la interfaz mostrada en este proyecto es una simulación visual del entorno Moodle.

De cara al futuro, sería muy valioso integrar un sistema de *login* vinculado al entorno institucional, lo que permitiría ofrecer respuestas personalizadas basadas en los datos del alumno (como su matrícula, asignaturas, horario, etc.). Para lograr esto sería necesaria una colaboración directa con los servicios informáticos de la universidad, así como el acceso controlado a los datos académicos internos de los estudiantes.

11.2. Mejora de la interfaz web

Aunque se ha desarrollado una interfaz funcional y clara, esta se mantiene como un prototipo básico pensado para pruebas. En futuras versiones sería recomendable mejorar el diseño visual, adaptarlo a dispositivos móviles, e in-

cluir funcionalidades como historial de preguntas, valoración de respuestas o incluso sugerencias automáticas.

11.3. Exploración de alternativas a GPT

Para este proyecto se ha utilizado la API de OpenAI (modelos GPT), principalmente por su facilidad de uso, la gran comunidad que lo respalda y su integración directa con librerías como LangChain. Esta decisión permitió centrar los esfuerzos en la arquitectura del sistema y evitar la necesidad de entrenar modelos propios.

No obstante, en un escenario a largo plazo —especialmente si se busca reducir costes o mantener el sistema de forma independiente— sería interesante explorar alternativas más económicas o de código abierto, como LLaMA (Meta), Mistral, Claude (Anthropic) o Cohere. Estas opciones podrían ofrecer un buen rendimiento en tareas específicas si se ajustan adecuadamente, aunque también requerirían más recursos técnicos para su despliegue.

11.4. Mejoras en la gestión y escalabilidad de la base de datos

El sistema actual está pensado para funcionar en local, con una base vectorial simple que puede procesar un volumen moderado de documentos. Sin embargo, si se desea escalar el sistema para funcionar en una universidad completa, sería necesario rediseñar la estructura de almacenamiento semántico.

Esto implicaría tanto mejorar la arquitectura (por ejemplo, separando las colecciones por facultad o tipo de documento), como introducir estrategias de particionado, filtrado o priorización para asegurar que el *chatbot* pueda responder de forma eficiente, incluso cuando trabaje con decenas de miles de fragmentos.

11.5. Ampliación de la memoria conversacional

En esta versión, la memoria del sistema solo guarda una interacción previa para mejorar la interpretación de preguntas ambiguas. En el futuro, sería útil implementar una memoria más completa que permita mantener el contexto durante varias rondas de conversación, lo cual mejoraría la fluidez y naturalidad del diálogo.

11.6. Automatización completa y despliegue

Finalmente, otra línea de trabajo sería automatizar el ciclo completo del sistema: desde la actualización de documentos institucionales hasta la ejecución periódica de pruebas y la monitorización del uso. Además, se podría estudiar su despliegue en la nube para que cualquier estudiante pudiera acceder al sistema sin necesidad de ejecutarlo en local.

Resumen

En conjunto, el sistema desarrollado constituye una base sólida sobre la que construir futuras versiones más potentes y adaptadas al entorno universitario real. Las mejoras propuestas no solo ampliarían su funcionalidad, sino que también lo harían más útil y accesible para los estudiantes, consolidándolo como una herramienta de apoyo académico de largo recorrido.

A. Anexo - Fragmentos de código relevantes

A continuación se incluyen imágenes de fragmentos de código fuente que ilustran partes clave del comportamiento del sistema. Estas capturas complementan las explicaciones ofrecidas en el capítulo de desarrollo y ayudan a visualizar la lógica aplicada en distintos módulos.

```python
def preguntar(req: PreguntaRequest):
    pregunta = req.pregunta.strip()
    categoria = clasificacion_chain.invoke(pregunta).content.strip().upper()
    if categoria == "CLARA":
        output = qa_chain.invoke({"query": pregunta})
        respuesta = output["result"]
        if "No tengo información suficiente" not in respuesta:
            memoria["ultima_pregunta"] = pregunta
            memoria["ultima_respuesta"] = respuesta
        else:
            memoria["ultima_pregunta"] = None
            memoria["ultima_respuesta"] = None

    elif categoria == "AMBIGUA" and memoria["ultima_pregunta"] and memoria["ultima_respuesta"]:
        pregunta_con_contexto = f"{memoria['ultima_pregunta']} -> {pregunta}"
        output = qa_chain.invoke({"query": pregunta_con_contexto})
        respuesta = output["result"]
        if "No tengo información suficiente" in respuesta:
            memoria["ultima_pregunta"] = None
            memoria["ultima_respuesta"] = None
    elif categoria == "AMBIGUA":
        respuesta = "Tu pregunta es ambigua. Por favor, intenta especificarla mejor."
        memoria["ultima_pregunta"] = None
        memoria["ultima_respuesta"] = None
    else:
        respuesta = "Esta pregunta no está relacionada con los documentos universitarios proporcionados."
        memoria["ultima_pregunta"] = None
        memoria["ultima_respuesta"] = None

    return {"respuesta": respuesta, "categoria": categoria}
```

Figura A.1. Fragmento de código que implementa la lógica de clasificación y gestión del contexto conversacional.

```python
def preguntar(req: PreguntaRequest):
    pregunta = req.pregunta.strip()
    categoria = clasificacion_chain.invoke(pregunta).content.strip().upper()
    if categoria == "CLARA":
        output = qa_chain.invoke({"query": pregunta})
        respuesta = output["result"]
        if "No tengo información suficiente" not in respuesta:
            memoria["ultima_pregunta"] = pregunta
            memoria["ultima_respuesta"] = respuesta
        else:
            memoria["ultima_pregunta"] = None
            memoria["ultima_respuesta"] = None

    elif categoria == "AMBIGUA" and memoria["ultima_pregunta"] and memoria["ultima_respuesta"]:
        pregunta_con_contexto = (
            f"Pregunta actual: {pregunta}\n"
            f"Pregunta anterior: {memoria['ultima_pregunta']}"
        )
        output = qa_chain.invoke({"query": pregunta_con_contexto})
        respuesta = output["result"]
        if "No tengo información suficiente" in respuesta:
            memoria["ultima_pregunta"] = None
            memoria["ultima_respuesta"] = None
    elif categoria == "AMBIGUA":
        respuesta = "Tu pregunta es ambigua. Por favor, intenta especificarla mejor."
        memoria["ultima_pregunta"] = None
        memoria["ultima_respuesta"] = None
    else:
        respuesta = "Esta pregunta no está relacionada con los documentos universitarios proporcionados."
        memoria["ultima_pregunta"] = None
        memoria["ultima_respuesta"] = None

    return {"respuesta": respuesta, "categoria": categoria}
```

Figura A.2. Diagrama lógico que muestra cómo el sistema decide si puede o no responder a una pregunta, según su clasificación y el contexto disponible.

B. Anexo - Coste de uso de la API de OpenAI

En la Figura B.1 se muestra el saldo final de la cuenta utilizada para el desarrollo del proyecto, tras realizar todas las pruebas y consultas necesarias.

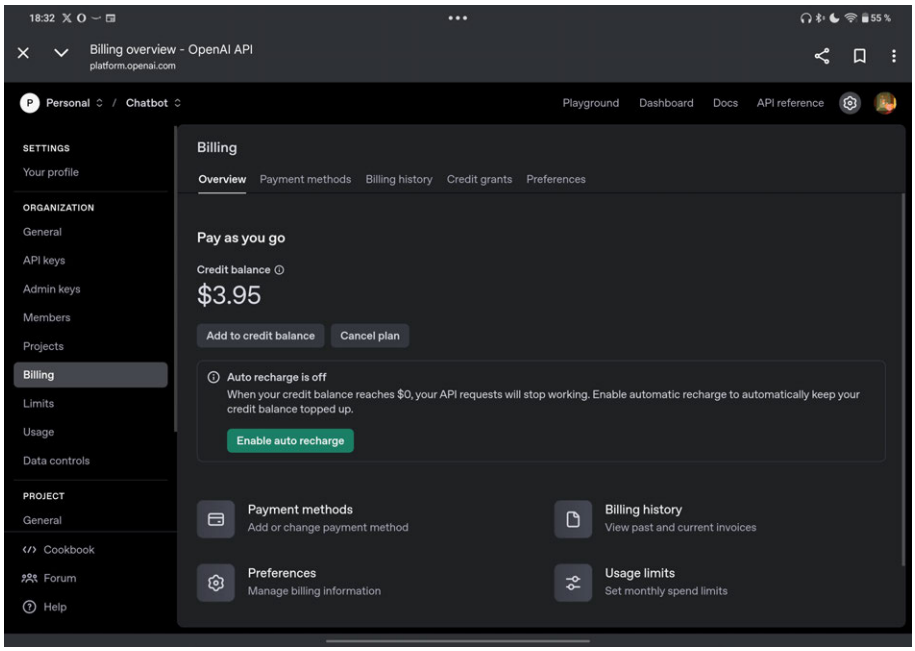


Figura B.1. Saldo restante en la cuenta de OpenAI tras completar el desarrollo del sistema con 10€ de presupuesto.

C. Anexo - Preguntas utilizadas en los tests automáticos

En esta sección se recogen los distintos conjuntos de preguntas utilizados durante la ejecución de pruebas automáticas, agrupados por tipo y origen. Cada bloque de preguntas se corresponde con un archivo de test en formato `.csv` incluido en el sistema.

Preguntas Ambiguas

Archivo: Test-Ambiguas.txt

Descripción: Este conjunto incluye preguntas deliberadamente formuladas de manera ambigua o poco específica, con el fin de evaluar la capacidad del sistema para detectar la falta de contexto suficiente y responder de forma adecuada.

#	Pregunta
1	¿Qué ocurrió con los ingresos ese año?
2	¿Está relacionado con la rentabilidad?
3	¿Se aplicó esa metodología también aquí?
4	¿Eso afecta al resultado final del análisis?
5	¿Hizo lo mismo?
6	¿Cuándo empieza ese periodo?
7	¿Esa decisión fue correcta en el contexto?
8	¿Cómo se interpreta ese cambio?
9	¿Hay alguna diferencia destacable?
10	¿Eso también se aplicó?

Tabla C.1. Listado de preguntas ambiguas utilizadas en el test `ambiguas.csv`.

Faltas Ortográficas

Archivo: Test-FaltasOrtograficas.txt

Descripción: Este conjunto de preguntas contiene errores ortográficos de forma deliberada. Se utiliza para evaluar la robustez del sistema ante entradas mal redactadas o con errores frecuentes cometidos por los usuarios.

#	Pregunta
1	¿Cuántos credtios europes pueden reconoseerse por partisipar en actividades universitarias?
2	Qué tipo de actvidades se incluyn en el catalgo de actividades acreditables?
3	Cual es el proedimiento para q una activdad sea aceptada en el Catálog General?
4	Q artículos normativos reconocen las actiidades culturales y deportivaz?
5	Cual es el max nmero de credts por ser representante estudiantil?
6	Cúal es el prmer dia lectibo del curso 2024-2025 en la upm?
7	Que dias festibos hay en deciembr 2023 segun el calendrio?
8	En qe fechas cae la convkatoria ordnaria de eneros?
9	Cuándo comienz el 2o semestre del 2025-26?
10	Q día se da el ezamen de nivelasion de inglés en 2024?
11	Cuales son las fechaz para la matriculacion ordnaria 2024?
12	Como puedo camviar asignaturas en mi matrcula?
13	Q pasa si no pago la matriula en tiempo?
14	K documentos hay q entregar si soy alumn nuevo?
15	Q formas de pago ay en la UPM?
16	Q servicios ofrece la tarjeta universitaria vitual?
17	Donde se encuentr la Secretaria de Alumns?
18	Ke normativa rige la pernanencia en la UPM?
19	Como se puede acceder al repositrio de software?
20	Q pasos hay que seguir para reconoser créditos si estudie fuera?

Tabla C.2. Listado de preguntas con faltas ortográficas utilizadas en el test `faltas_ortograficas.csv`.

Palabaras Clave

Archivo: Test-PalabrasClave.txt

Descripción: Este conjunto de pruebas está compuesto por términos o ex-

presiones clave de una sola palabra (o breves sintagmas), sin estructura de pregunta. El objetivo es evaluar si el sistema es capaz de identificar correctamente el contexto relevante en base a la presencia de conceptos importantes, incluso cuando no se formula una pregunta completa.

#	Entrada
1	matrícula
2	asignaturas
3	créditos ECTS
4	expediente
5	calendario académico
6	convocatoria
7	actas
8	segunda matrícula
9	reconocimiento de créditos
10	secretaría
11	carné universitario
12	certificado
13	devolución de tasas
14	permanencia
15	traslados
16	becas
17	normativa
18	Politécnica Virtual
19	repositorio de software
20	VPN
21	correo institucional
22	biblioteca
23	aula virtual
24	delegado
25	Consejo de Estudiantes
26	Delegación
27	actividades acreditables
28	representación estudiantil
29	comedor universitario
30	Secretaría de Alumnos
31	Bloque VIII
32	Edificio Prisma
33	Aula 1001
34	pabellón deportivo
35	cafetería

Tabla C.3. Listado de entradas de prueba basadas en palabras clave sueltas (`palabras_clave.csv`).

Banco de Preguntas

Archivo: Tests-Documentos.txt

Descripción: Conjunto extenso de preguntas reales y simuladas agrupadas por categorías temáticas frecuentes en la documentación universitaria. Este bloque se utilizó para evaluar el rendimiento del sistema sobre documentos concretos subidos en la página web de la universidad.

Categoría: Actividades acreditables y representación estudiantil

#	Pregunta
1	¿Cuántos créditos europeos pueden reconocerse por participar en actividades universitarias acreditables?
2	¿Qué tipo de actividades se incluyen en el Catálogo General de Actividades Universitarias Acreditables?
3	¿Cuál es el procedimiento para que una actividad sea incluida en el Catálogo General de Actividades Acreditables?
4	¿Qué artículos de la normativa regulan el reconocimiento de actividades culturales y deportivas?
5	¿Cuál es el número máximo de créditos que pueden reconocerse por actividades de representación estudiantil?
6	¿Qué criterios se usan para determinar si una actividad es reconocible con créditos ECTS?
7	¿Cuántos créditos pueden obtenerse como delegado de titulación según la normativa?
8	¿Qué se requiere para solicitar el reconocimiento de créditos por asistencia a sesiones de órganos colegiados?
9	¿Qué organismo aprueba el Catálogo General de Actividades Universitarias Acreditables?
10	¿Qué normativa regula el reconocimiento y transferencia de créditos en la UPM desde 2013?

Categoría: Calendario académico

#	Pregunta
1	¿Cuál es el primer día lectivo del curso académico 2024-2025 en la UPM?
2	¿Qué días festivos hay en diciembre de 2023 según el calendario académico?
3	¿En qué fechas se celebra la convocatoria ordinaria de exámenes del primer semestre en enero de 2024?
4	¿Qué actividades permiten ampliación restringida de matrícula y cuántos ECTS se pueden añadir como máximo?
5	¿Cuándo comienza el segundo semestre del curso 2025-2026?
6	¿Qué días festivos se recogen durante el mes de mayo en el calendario académico 2025?
7	¿En qué fechas puede un estudiante de máster universitario formalizar su matrícula en septiembre de 2024?
8	¿Cuándo tienen lugar las vacaciones de Semana Santa en el curso 2023-2024?
9	¿Qué día se celebra el examen de nivelación de lengua inglesa en diciembre de 2024?
10	¿Cuál es la fecha límite de cierre de actas para la convocatoria extraordinaria del curso 2024-2025?

Categoría: Cambios de titulación y reingresos

#	Pregunta
1	¿Qué requisitos debo cumplir para solicitar un reingreso en la UPM si cursé parcialmente una diplomatura?
2	¿Durante qué fechas se puede solicitar el reingreso en la UPM según el protocolo de 2022?
3	¿Qué consecuencias económicas tiene el reingreso o cambio de titulación en la ETSISI?
4	¿Qué diferencia hay entre un cambio de plan dentro de la misma titulación y un traslado entre titulaciones diferentes?
5	¿Qué información debo incluir en la solicitud online para un reingreso en la UPM?
6	¿Puedo solicitar simultáneamente un reingreso y un cambio de titulación?
7	¿Qué implica el cambio de plan del Grado en Software 2009 al de 2014?
8	¿Qué ocurre si quiero pasar de un grado extinguido a otro que ya también ha sido extinguido?
9	¿Dónde encuentro las tablas de reconocimiento de créditos para cambios de plan?
10	¿Qué proceso deben seguir los estudiantes con estudios universitarios extranjeros para acceder a grados UPM?

Categoría: Carné universitario

#	Pregunta
1	¿Cómo puedo obtener la Tarjeta Universitaria Virtual de la UPM?
2	¿Qué requisitos hay para que me emitan el carné universitario virtual?
3	¿Qué servicios ofrece la tarjeta universitaria virtual de la UPM?

Categoría: Devolución de tasas

#	Pregunta
1	¿Cómo se solicita la devolución de precios públicos en la UPM?
2	¿Qué documentación es necesaria para tramitar la devolución de tasas universitarias?
3	¿Cuál es el correo de contacto del Negociado de Devolución de Precios Públicos de la UPM?

Categoría: Guía del estudiante

#	Pregunta
1	¿Qué grados se imparten actualmente en la ETSI Sistemas Informáticos de la UPM?
2	¿Dónde se encuentra la Secretaría de Alumnos de la ETSISI y qué gestiones se realizan allí?
3	¿Cómo se estructura el proceso de matrícula anual en la UPM?
4	¿Qué servicios ofrece el Centro de Informática y Comunicaciones (CIC)?
5	¿Qué es el procedimiento EVALÚA y para qué sirve?
6	¿Cómo se puede acceder al repositorio de software gratuito de la UPM?
7	¿Qué ayudas ofrece la Delegación de Alumnos, como comedor o publicaciones?
8	¿Qué normativa regula la permanencia de los estudiantes en la UPM?
9	¿Qué es Politécnica Virtual y qué gestiones se pueden hacer desde allí?
10	¿Qué requisitos hay para acreditar el nivel B2 de inglés en la UPM?

Categoría: Matrícula

#	Pregunta
1	¿Cuáles son las fechas del periodo ordinario de matrícula para el curso 2024-2025?
2	¿Qué asignaturas pueden añadirse durante el periodo extraordinario de matrícula?
3	¿Cómo puedo modificar mi matrícula si necesito cambiar asignaturas o modalidad?
4	¿Qué importe tiene el crédito para grados según su nivel de experimentalidad?
5	¿Qué tipos de exenciones existen para el pago de matrícula?
6	¿Cuáles son las formas de pago aceptadas para la matrícula en la UPM?
7	¿Qué sucede si no pago la matrícula en los plazos establecidos?
8	¿Qué documentos debo entregar si soy estudiante de nuevo ingreso?
9	¿Cuál es la normativa de permanencia y cómo afecta a los estudiantes de primer curso?
10	¿Qué precios se aplican a estudiantes visitantes para cursar asignaturas sueltas?

Bibliografía

[1] C. W. Okonkwo y A. Ade-Ibijola, «Chatbots applications in education: A systematic review,» *Computers and Education: Artificial Intelligence*, vol. 2, pág. 100 033, 2021. DOI: 10.1016/j.caeai.2021.100033.

[2] M. A. Kuhail, N. Alturki, S. Alramlawi y K. Alhejori, «Interacting with educational chatbots: A systematic review,» *Education and Information Technologies*, vol. 27, n.º 5, págs. 6811-6839, 2022. DOI: 10.1007/s10639-022-11177-3.

[3] E. Adamopoulou y L. Moussiades, «An overview of chatbot technology,» *Artificial Intelligence Applications and Innovations*, págs. 373-383, 2020.

[4] A. Vaswani, N. Shazeer, N. Parmar et al., «Attention is All You Need,» *Advances in Neural Information Processing Systems*, vol. 30, 2017. dirección: https://arxiv.org/abs/1706.03762.

[5] T. B. Brown, B. Mann, N. Ryder et al., «Language Models are Few-Shot Learners,» *Advances in Neural Information Processing Systems*, vol. 33, págs. 1877-1901, 2020. dirección: https://arxiv.org/abs/2005.14165.

[6] R. Bommasani, D. A. Hudson, E. Adeli et al., «On the Opportunities and Risks of Foundation Models,» *arXiv preprint arXiv:2108.07258*, 2022. dirección: https://arxiv.org/abs/2108.07258.

[7] P. Lewis, E. Perez, A. Piktus et al., «Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,» en *Advances in Neural Information Processing Systems*, vol. 33, 2020, págs. 9459-9474. dirección: https://arxiv.org/abs/2005.11401.

[8] DeepLearning.AI, *LangChain: Chat with Your Data*, Curso en línea disponible en: https://www.deeplearning.ai/short-courses/langchain-chat-with-your-data/, 2023.

[9] T. Mikolov, K. Chen, G. Corrado y J. Dean, «Efficient Estimation of Word Representations in Vector Space,» *arXiv preprint arXiv:1301.3781*, 2013. dirección: https://arxiv.org/abs/1301.3781.

[10] R. Nguyen, *Choosing the Right Embedding Model: A Guide for LLM Applications*, https://medium.com/@ryanntk/choosing-the-right-embedding-model-a-guide-for-llm-applications-7a60180d28e3, Consultado en junio de 2025, 2023.

[11] J. Pennington, R. Socher y C. D. Manning, «GloVe: Global Vectors for Word Representation,» en *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, págs. 1532-1543. dirección: https://aclanthology.org/D14-1162.

[12] J. Devlin, M.-W. Chang, K. Lee y K. Toutanova, «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,» *arXiv preprint arXiv:1810.04805*, 2018. dirección: https://arxiv.org/abs/1810.04805.

[13] N. Reimers e I. Gurevych, «Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,» en *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019. dirección: https://arxiv.org/abs/1908.10084.

[14] MathWorks, *Word Embedding t-SNE Plot*, https://de.mathworks.com/help/textanalytics/ug/visualize-word-embedding-using-text-scatter-plot.html, Adaptado de un tutorial oficial de MathWorks sobre visualización de embeddings con t-SNE, 2020.

[15] ChromaDB contributors, *Chroma: The AI-native open-source embedding database*, https://www.trychroma.com, Accedido en junio de 2025, 2023.

[16] J. Wei, X. Wang, D. Schuurmans et al., «Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,» *arXiv preprint arXiv:2201.11903*, 2022. dirección: https://arxiv.org/abs/2201.11903.

[17] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo e Y. Iwasawa, «Large Language Models are Zero-Shot Reasoners,» *Advances in Neural Information Processing Systems*, vol. 35, págs. 22 199-22 213, 2022. dirección: https://arxiv.org/abs/2205.11916.

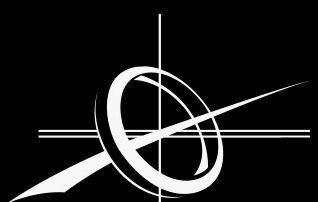
[18] OpenAI, *OpenAI Tokenizer*, https://platform.openai.com/tokenizer, 2023.

[19] OpenAI, *OpenAI API Documentation*, https://platform.openai.com/docs, 2023.

[20] OpenAI, *GPT-4 Technical Report*, https://openai.com/research/gpt-4, 2023.

[21] S. Zheng y otros, «A Survey of Large Language Models,» *arXiv preprint arXiv:2303.18223*, 2023. dirección: https://arxiv.org/abs/2303.18223.

[22] Universidad Politécnica de Madrid, *Plan de Sostenibilidad Ambiental de la UPM 2021–2025*, Disponible en: https://sostenibilidad.upm.es/plan-de-sostenibilidad-ambiental/, 2021.



Universidad Politécnica de Madrid

ETSI **SISTEMAS INFORMÁTICOS**