

**UNIVERSIDAD CEU SAN PABLO**

Facultad de Ciencias Económicas y Empresariales

TRABAJO FIN DE GRADO

Grado en Business Intelligence

**Desarrollo de un chatbot para asistencia y resolución de dudas de estudiantes universitarios**

**Development of a chatbot for student assistance and university-related queries**

Autor: Ignacio Amores Acedo

Tutor: Pablo Arés Gastesi

Curso Académico: 2024-2025

Resumen

En este trabajo se ha desarrollado un chatbot pensado para ayudar a los estudiantes de universidad a resolver dudas comunes sobre trámites, servicios y temas académicos. El objetivo principal ha sido facilitar el acceso a la información sin que el alumno tenga que depender siempre del personal administrativo.

Para ello, he creado dos versiones distintas del sistema. Una funciona con spaCy, una herramienta que permite detectar palabras clave y entender lo que el usuario está preguntando. La otra utiliza embeddings, que permiten comparar el significado de la pregunta del usuario con una base de respuestas ya definidas, para ofrecer la más adecuada en cada caso.

Ambas versiones se han validado mediante un dataset enriquecido que recoge preguntas directas, reformulaciones reales y casos de prueba sin solución válida. La evaluación comparativa refleja el rendimiento diferencial de ambos enfoques ante distintos niveles de dificultad, analizando tanto su precisión semántica como su robustez ante consultas fuera de dominio.

Palabras clave: chatbot, lenguaje natural, spaCy, embeddings, dataset

Abstract

This project focuses on the development of a chatbot designed to help university students solve common questions related to procedures, services, and academic topics. The main goal has been to make access to information easier, so students don’t always have to rely on administrative staff.

To achieve this, I built two different versions of the system. One uses spaCy, a tool that helps detect key elements in the user’s message to understand what they’re asking. The other makes use of embeddings, which allow the system to compare the meaning of the user’s question with a set of predefined answers and return the most relevant one.

Both versions were evaluated using an enriched dataset including direct questions, real reformulations and control cases with no valid answer. The comparative analysis highlighted the performance differences of each model depending on question complexity, assessing both semantic accuracy and robustness against out-of-domain queries.

Keywords: chatbot, natural language, spaCy, embeddings, dataset

Índice

[1 Introducción 4](#_Toc201106618)

[1.1 Contexto y motivación del proyecto 5](#_Toc201106619)

[1.2 Detección del problema y necesidad real 6](#_Toc201106620)

[1.3 Objetivos generales y específicos 7](#_Toc201106621)

[1.3.1 Objetivo general 7](#_Toc201106622)

[1.3.2 Objetivos específicos 7](#_Toc201106623)

[1.4 Estructura de la memoria 7](#_Toc201106624)

[2 Marco teórico y estado del arte 9](#_Toc201106625)

[2.1 Chatbots y asistentes conversacionales: evolución y aplicaciones actuales 9](#_Toc201106626)

[2.2 Procesamiento del lenguaje natural en sistemas inteligentes 10](#_Toc201106627)

[2.3 La librería spaCy como herramienta de NLP 11](#_Toc201106628)

[2.4 Embeddings semánticos y búsqueda basada en similitud 11](#_Toc201106629)

[2.5 Comparativa con modelos de lenguaje grandes (LLMs) y motivos de exclusión 12](#_Toc201106630)

[3 Metodología del proyecto 14](#_Toc201106631)

[3.1 Enfoque general de desarrollo 14](#_Toc201106632)

[3.2 Arquitectura general del sistema 15](#_Toc201106633)

[3.3 Implementación de la versión con spaCy 16](#_Toc201106634)

[3.4 Implementación de la versión con embeddings 18](#_Toc201106635)

[3.5 Diseño del dataset y categorización de respuestas 20](#_Toc201106636)

[3.6 Criterios definidos para la evaluación 21](#_Toc201106637)

[3.7 Interfaz y despliegue funcional 22](#_Toc201106638)

[4 Resultados y evaluación comparativa 25](#_Toc201106639)

[4.1 Porcentaje de acierto por tipo de pregunta 25](#_Toc201106640)

[4.2 Matriz de confusión simplificada 26](#_Toc201106641)

[4.3 Tiempos de respuesta 27](#_Toc201106642)

[5 Discusión 29](#_Toc201106643)

[6 Conclusiones finales 31](#_Toc201106644)

[6.1 Conclusiones técnicas 31](#_Toc201106645)

[6.2 Aprendizaje personal del proceso 32](#_Toc201106646)

[6.3 Mejoras futuras y aplicación real 33](#_Toc201106647)

[7 Bibliografía 35](#_Toc201106648)

[8 Anexos 36](#_Toc201106649)

[8.1 Ejemplos de preguntas y respuestas del dataset 36](#_Toc201106650)

[8.2 Fragmentos de código representativos 36](#_Toc201106651)

[8.3 Capturas de la interfaz funcional implementada 36](#_Toc201106652)

# Introducción

El desarrollo de modelos y técnicas de procesamiento del lenguaje natural ha hecho posible que los sistemas conversacionales actuales puedan interpretar preguntas formuladas en lenguaje libre con un nivel de precisión y contexto impensable hace apenas una década. Este avance técnico ha permitido automatizar interacciones en sectores como la banca, la salud o la educación, donde las consultas repetitivas sobre procesos definidos generan una carga operativa innecesaria.

En este trabajo se propone un chatbot orientado a resolver consultas frecuentes de estudiantes universitarios, centrado en temas como procedimientos administrativos, normativa académica, uso de servicios o problemas comunes relacionados con la plataforma institucional. Su diseño responde a una situación concreta y conocida: la saturación de los canales de soporte durante periodos clave del curso, como las fechas de matrícula, las convocatorias de evaluación o la presentación de trabajos de fin de grado.

Se han desarrollado dos versiones funcionales del sistema. La primera se apoya en técnicas de procesamiento lingüístico tradicionales mediante la herramienta spaCy, que permite identificar entidades, estructuras sintácticas y patrones gramaticales. La segunda utiliza representaciones vectoriales a partir de embeddings semánticos, lo que facilita el emparejamiento entre la consulta del usuario y un conjunto de respuestas predefinidas, incluso cuando el mensaje no coincide literal ni estructuralmente con ninguna pregunta almacenada.

Ambas aproximaciones se han implementado sobre un controlador común que gestiona la interpretación del mensaje, la búsqueda de información y la construcción de la respuesta final. La evaluación de cada versión se plantea desde una perspectiva comparativa, atendiendo a criterios como la precisión semántica, la robustez ante reformulaciones y la capacidad de adaptación a nuevos contenidos sin necesidad de reprogramación directa.

Más allá del desarrollo técnico, este trabajo busca demostrar que es posible construir una herramienta útil, modular y adaptable para mejorar el acceso a la información institucional sin recurrir necesariamente a modelos generativos de gran escala. La solución planteada prioriza la claridad, el control del flujo conversacional y la eficiencia computacional, sin renunciar a una experiencia de usuario fluida.

## Contexto y motivación del proyecto

Durante el desarrollo de la vida académica, los estudiantes se enfrentan de forma recurrente a dudas sobre procesos institucionales que no siempre están bien documentados o centralizados. Consultas como “¿cuándo se abre el plazo de matrícula?”, “¿dónde se solicita un duplicado del carné?”, o “¿qué ocurre si no me presento a una convocatoria?” acaban canalizándose a través del correo electrónico o presencialmente en secretaría, generando tiempos de espera innecesarios y una carga constante para el personal administrativo.

Este tipo de preguntas se repiten curso tras curso. No requieren interpretación compleja ni criterios administrativos variables; su resolución es directa, pero exige que el estudiante sepa a quién acudir y cómo formularla. A menudo, esta dependencia de canales formales se traduce en retrasos, falta de respuesta o abandono de trámites sencillos por parte del propio alumno.

El proyecto parte de esa situación vivida en primera persona. La falta de una herramienta accesible, capaz de atender esas cuestiones básicas de forma automática, representa una oportunidad clara de aplicar técnicas de procesamiento del lenguaje natural para mejorar la experiencia universitaria. La decisión de construir un chatbot no nace de una moda tecnológica, sino de un análisis funcional de qué tipo de sistema puede absorber esa demanda sin alterar los procesos ya establecidos.

La motivación personal detrás de este trabajo también se vincula con el interés por aplicar soluciones prácticas en entornos reales, más allá del código experimental o los modelos entrenados con datasets artificiales. La idea no es replicar lo que ya hacen los grandes modelos generativos, sino estudiar si se puede construir un sistema funcional, controlado y útil con herramientas accesibles y bien ajustadas a un dominio concreto como el universitario.

## Detección del problema y necesidad real

El problema no es la falta de información, sino la forma en que se presenta y distribuye. En muchas universidades, los procedimientos más consultados por los estudiantes (matrícula, convocatorias, certificados, documentación o acceso a plataformas), están explicados en documentos PDF, apartados web dispersos o instrucciones mal estructuradas. Incluso cuando la respuesta existe, encontrarla requiere tiempo, experiencia previa o saber buscar en el sitio correcto.

El resultado es una sobrecarga de los canales de soporte. El correo institucional, la atención presencial y las líneas de contacto asumen consultas que podrían resolverse sin intervención humana si existiera un sistema que interpretase correctamente la pregunta y ofreciera una respuesta directa. Muchas de estas consultas no varían con el tiempo ni requieren interpretación administrativa. Son constantes y repetitivas.

Estamos en una etapa donde el acceso a la información debe ser inmediato, sencillo y fiable. El estudiante no busca rellenar formularios para obtener una respuesta, ni consultar manuales extensos. Espera plantear una duda concreta y recibir una solución clara. Esa expectativa, alimentada por los sistemas de atención automática en otros sectores, todavía no está bien cubierta en el entorno universitario.

Este proyecto responde a esa carencia. No se plantea como una solución experimental ni como una aplicación de última tecnología sin propósito. La necesidad existe, es concreta y está bien definida: resolver, de forma automatizada y precisa, las consultas recurrentes que saturan los canales de atención al estudiante. Para ello, se plantea un sistema conversacional capaz de entender el mensaje del usuario, clasificarlo adecuadamente y devolver una respuesta pertinente sin depender de interacción humana.

## Objetivos generales y específicos

### Objetivo general

Diseñar, implementar y validar un sistema conversacional orientado a resolver de forma automática las dudas más frecuentes de los estudiantes universitarios, utilizando técnicas de procesamiento del lenguaje natural que permitan interpretar el mensaje del usuario y devolver respuestas claras, útiles y coherentes con el contenido institucional.

### Objetivos específicos

* Construir un flujo de procesamiento de texto capaz de identificar la intención del usuario y asociarla a una categoría de respuesta predefinida.
* Desarrollar dos versiones funcionales del chatbot: una basada en extracción lingüística mediante spaCy, y otra en búsqueda semántica mediante embeddings.
* Definir una estructura de datos organizada por temas y consultas, que sirva como base de conocimiento del sistema.
* Evaluar la capacidad de ambas versiones para resolver consultas reales formuladas de forma libre, valorando su precisión, robustez ante reformulaciones y facilidad de mantenimiento.
* Comparar los resultados obtenidos y determinar cuál de las dos aproximaciones ofrece mejor rendimiento para el uso previsto.
* Presentar una solución funcional, documentada y replicable, con posibilidad de ampliación futura sin rediseñar el sistema desde cero.

## Estructura de la memoria

El documento se organiza según el orden real de trabajo seguido en el desarrollo.

El primer capítulo define el problema, expone el objetivo que persigue el proyecto y justifica su interés. Después, el capítulo dos recoge los conceptos técnicos que permiten entender los sistemas construidos. Aquí se explican las bases de los chatbots, el funcionamiento del procesamiento de lenguaje natural, los modelos sintácticos como spaCy, los embeddings semánticos y la diferencia con los grandes modelos de lenguaje.

El capítulo tres detalla el desarrollo completo del sistema. Se explica la arquitectura general implementada, las dos versiones construidas (spaCy y embeddings), el diseño del dataset de preguntas, los criterios de evaluación aplicados y la construcción de la interfaz funcional para probar el chatbot.

El capítulo cuatro muestra los resultados obtenidos tras las pruebas realizadas. Se presentan los porcentajes de acierto de cada modelo, la matriz de confusión con los tipos de error detectados y los tiempos de respuesta medidos en los test de rendimiento.

El capítulo cinco analiza los resultados. Se comparan las diferencias reales entre ambos modelos, se identifican sus puntos fuertes y débiles, y se valoran sus comportamientos ante preguntas bien formuladas, reformuladas y fuera de dominio.

El capítulo seis recoge las conclusiones generales extraídas del proyecto, el aprendizaje adquirido durante el desarrollo y varias propuestas de mejora o ampliación que podrían aplicarse si el sistema se implantase en un entorno real.

El capítulo siete contiene las referencias bibliográficas utilizadas.

El capítulo ocho recoge anexos con ejemplos de preguntas reales, fragmentos representativos de código y capturas de la interfaz desarrollada.

# Marco teórico y estado del arte

## Chatbots y asistentes conversacionales: evolución y aplicaciones actuales

El desarrollo de sistemas conversacionales comenzó en las décadas de 1960 y 1970 con proyectos experimentales como ELIZA y PARRY. Estos sistemas emulaban conversaciones limitadas mediante patrones de sustitución de texto sin interpretar el significado real de los mensajes. Aunque rudimentarios, demostraron la posibilidad de simular un diálogo estructurado con reglas muy simples.

Durante años, los chatbots permanecieron ligados a guiones predefinidos y árboles de decisión cerrados. Cada posible pregunta debía ser anticipada por el desarrollador, y cualquier desviación del patrón previsto llevaba a respuestas genéricas o errores. Estos sistemas cumplían funciones muy concretas en entornos donde el margen de variación lingüística era mínimo.

El avance de los algoritmos de procesamiento de lenguaje natural permitió superar esta rigidez inicial. La introducción de técnicas de tokenización, análisis morfosintáctico, reconocimiento de entidades y clasificación de intenciones abrió la posibilidad de gestionar consultas formuladas de manera libre. Los chatbots modernos no dependen únicamente de coincidencias exactas de texto, sino que analizan la estructura de la frase para identificar su propósito.

Hoy los sistemas conversacionales están implantados en sectores diversos, como servicios financieros, comercio electrónico, plataformas de soporte técnico y entornos educativos. Su función principal es absorber consultas recurrentes, ofrecer respuestas inmediatas y mantener la coherencia con los procedimientos internos de la organización. La precisión, el control sobre el conocimiento manejado y la capacidad de adaptación progresiva definen los sistemas operativos actuales.

El chatbot desarrollado en este proyecto sigue esa línea de diseño. Se centra en interpretar correctamente las preguntas recurrentes de los estudiantes, clasificar el contenido y ofrecer respuestas ajustadas a los procedimientos establecidos por la institución, sin depender de respuestas generadas de forma libre.

## Procesamiento del lenguaje natural en sistemas inteligentes

El procesamiento del lenguaje natural permite que un sistema interprete y manipule texto o discurso en lenguaje humano. Estas técnicas convierten un mensaje escrito en una representación estructurada que puede ser analizada por un programa informático. El primer paso consiste habitualmente en dividir el texto en tokens, es decir, en unidades mínimas como palabras o signos de puntuación. Posteriormente, se analizan la morfología, la categoría gramatical de cada palabra y las dependencias sintácticas que existen entre ellas.

Este análisis estructural facilita la detección de patrones en el mensaje, como la identificación de entidades mencionadas, fechas, ubicaciones o nombres propios. Sobre esa estructura, el sistema puede aplicar reglas o algoritmos que permiten clasificar el contenido de la consulta, seleccionar la respuesta adecuada o desencadenar acciones adicionales dentro del sistema.

El procesamiento de lenguaje natural se utiliza hoy de forma estable en múltiples sistemas inteligentes. Los motores de búsqueda utilizan algoritmos de NLP para interpretar la intención detrás de las consultas de los usuarios. Los sistemas de atención telefónica automática aplican reconocimiento de voz y procesamiento sintáctico para gestionar peticiones. Las aplicaciones de asistencia virtual procesan instrucciones escritas o dictadas y generan respuestas ajustadas al contexto.

En sistemas conversacionales como el desarrollado en este trabajo, el PLN permite trabajar con mensajes formulados de forma libre, sin necesidad de que el usuario seleccione previamente una opción entre varias disponibles. El análisis de las estructuras lingüísticas proporciona la flexibilidad necesaria para interpretar preguntas expresadas de múltiples formas y reducir la dependencia de coincidencias exactas de texto.

## La librería spaCy como herramienta de NLP

spaCy es una librería de procesamiento del lenguaje natural optimizada para el análisis de texto en tiempo real. Permite realizar operaciones de tokenización, análisis morfosintáctico, detección de entidades y clasificación de dependencias gramaticales de forma rápida y con un consumo moderado de recursos. A diferencia de otros enfoques que priorizan el entrenamiento de modelos propios, spaCy proporciona modelos preentrenados ajustados a varios idiomas y preparados para ser utilizados directamente sobre tareas comunes de análisis lingüístico.

En el contexto de este proyecto, spaCy se emplea como base de la primera versión del chatbot. A través de sus modelos de análisis sintáctico es posible identificar el núcleo de la oración, los objetos afectados y los modificadores que acompañan a cada mensaje. La detección de entidades permite localizar términos específicos relevantes para el dominio universitario, como fechas de matrícula, códigos de asignatura, departamentos o servicios concretos.

El principal valor de spaCy reside en su capacidad de descomponer el mensaje y proporcionar una representación estructurada sobre la que aplicar reglas específicas del sistema. Estas reglas, adaptadas al contexto académico, permiten mapear el contenido del mensaje a categorías predefinidas de consulta sin depender de coincidencias exactas de texto. Además, spaCy permite extender sus capacidades mediante componentes personalizados, lo que facilita incorporar reglas adicionales cuando el dominio de aplicación lo requiere.

Su rendimiento en tareas de extracción de información sobre textos controlados, como es el caso de las preguntas frecuentes de un entorno universitario, permite desarrollar sistemas conversacionales que ofrecen precisión y estabilidad sin necesidad de entrenar modelos propios de gran escala.

## Embeddings semánticos y búsqueda basada en similitud

Los embeddings permiten transformar frases o documentos completos en vectores numéricos que conservan información semántica sobre el contenido. Esta representación vectorial permite comparar el significado de dos textos incluso cuando no comparten las mismas palabras ni la misma estructura gramatical. A diferencia de las técnicas basadas en reglas o coincidencias de tokens, los embeddings sitúan cada frase en un espacio multidimensional donde la proximidad entre vectores refleja la similitud conceptual entre mensajes.

En el sistema desarrollado, esta técnica se emplea en la segunda versión del chatbot. Cada pregunta registrada en la base de datos de respuestas se codifica previamente en su representación vectorial. Cuando el usuario envía un nuevo mensaje, el sistema genera su embedding correspondiente y compara este vector con todos los almacenados. La respuesta asociada al vector más cercano es seleccionada como la respuesta final.

Este enfoque permite trabajar con entradas mucho más variadas, ya que el sistema no requiere que la pregunta del usuario coincida de forma exacta con ninguna de las preguntas registradas. Frases reformuladas, sinónimos o cambios de orden en las palabras siguen conduciendo al mismo resultado si el contenido semántico global de la consulta es similar al de una de las respuestas existentes.

La principal ventaja de este método es su flexibilidad ante la forma en que los usuarios formulan sus preguntas. Además, facilita ampliar el sistema añadiendo nuevas respuestas sin necesidad de redefinir reglas gramaticales o actualizar manualmente patrones lingüísticos. Sin embargo, su efectividad depende de la calidad de los embeddings utilizados y de la correcta configuración de los umbrales de similitud que determinan qué distancia entre vectores se considera suficientemente cercana.

## Comparativa con modelos de lenguaje grandes (LLMs) y motivos de exclusión

Los modelos de lenguaje grandes como GPT-4, LLaMA o Claude han sido entrenados con enormes volúmenes de datos y disponen de miles de millones de parámetros. Su capacidad para generar texto coherente, interpretar consultas abiertas y mantener el contexto a lo largo de una conversación los convierte en herramientas eficaces para múltiples tareas de generación de lenguaje natural.

Su principal ventaja reside en su versatilidad. Pueden gestionar entradas altamente variadas, adaptarse a conversaciones no estructuradas y ofrecer respuestas elaboradas sin necesidad de programar reglas específicas o definir previamente el conjunto de intenciones posibles. Esta flexibilidad permite su uso en asistentes personales, motores de generación de contenido o sistemas de soporte abierto.

Sin embargo, la naturaleza de este proyecto no exige ese nivel de complejidad. Las consultas que el sistema debe resolver están bien definidas, giran en torno a un conjunto reducido de temas institucionales y no requieren la generación de respuestas libres. La prioridad está en controlar el conocimiento manejado, asegurar la coherencia de las respuestas con la normativa vigente y mantener trazabilidad sobre cada mensaje emitido.

El uso de un modelo generativo introduciría riesgos de respuestas inexactas, improvisadas o fuera de los márgenes establecidos, además de elevar los requisitos computacionales, económicos y de supervisión ética del sistema. En este caso, la aplicación de técnicas más controlables como la extracción lingüística con spaCy y la búsqueda semántica mediante embeddings permite construir un sistema funcional, estable y ajustado al dominio específico para el que ha sido diseñado

# Metodología del proyecto

## Enfoque general de desarrollo

El objetivo técnico del proyecto es construir un sistema conversacional que interprete preguntas formuladas en lenguaje libre por estudiantes universitarios y devuelva respuestas exactas basadas en la normativa y los procedimientos académicos establecidos. Para ello, el desarrollo se ha estructurado en dos versiones funcionales independientes, ambas capaces de gestionar el mismo conjunto de preguntas y respuestas, pero aplicando técnicas distintas de procesamiento del lenguaje.

La primera versión emplea la librería spaCy como motor de análisis lingüístico. Utiliza el modelo preentrenado de español para descomponer el mensaje del usuario, identificar las estructuras gramaticales principales y calcular la similitud semántica con las preguntas almacenadas. El emparejamiento se basa en un sistema de umbrales que define la respuesta más adecuada según el grado de coincidencia detectado.

La segunda versión utiliza representaciones vectoriales mediante embeddings. Cada pregunta del dataset se codifica previamente en forma de vectores numéricos con el modelo all-MiniLM-L6-v2 de Sentence Transformers. Durante la ejecución, las preguntas de los usuarios se convierten también en vectores, y la comparación se realiza calculando la distancia semántica entre los vectores almacenados y el nuevo mensaje recibido. La búsqueda de la respuesta más adecuada se gestiona a través de ChromaDB como motor vectorial local.

Ambas versiones comparten la misma estructura de datos base, el mismo conjunto de preguntas reformuladas y el mismo flujo general de interacción. Las diferencias se encuentran en el núcleo del procesamiento lingüístico aplicado y en la lógica de búsqueda que determina la respuesta seleccionada. El objetivo de este planteamiento dual es comparar el comportamiento de ambos enfoques sobre un dominio controlado y analizar sus diferencias en términos de precisión, adaptabilidad y mantenibilidad.

## Arquitectura general del sistema

El sistema está dividido en dos capas principales: el módulo de procesamiento del lenguaje natural (backend) y la interfaz de usuario (frontend). Ambas versiones del chatbot comparten la misma arquitectura general, diferenciándose únicamente en el núcleo de procesamiento responsable de interpretar la consulta del usuario.

El frontend consiste en una interfaz web sencilla desarrollada en HTML, CSS y JavaScript. El usuario introduce su consulta en un campo de texto, y el mensaje se envía mediante una petición HTTP POST a la API local. La comunicación entre el frontend y el backend se realiza de forma asíncrona a través de JavaScript, utilizando el método fetch sobre la ruta /api/chatbot expuesta por el servidor Flask.

El backend está implementado en Python mediante el microframework Flask, que gestiona las peticiones entrantes, centraliza el flujo de trabajo y organiza el acceso a los diferentes módulos de procesamiento. Al recibir una nueva consulta, el backend normaliza el texto y lo redirige al modelo correspondiente, según la versión activa (spaCy o embeddings).

Ambas versiones trabajan sobre el mismo conjunto de datos, almacenado en un fichero faq\_data.json. Este fichero contiene las preguntas principales, sus reformulaciones y las respuestas asociadas. La estructura permite mantener el control sobre el conocimiento manejado y facilita la ampliación o modificación de los datos sin necesidad de reentrenar modelos.

En la versión spaCy, el backend carga el modelo preentrenado es\_core\_news\_md y calcula la similitud semántica entre el mensaje recibido y las preguntas del dataset. En la versión embeddings, el backend utiliza Sentence Transformers para generar los vectores semánticos de las preguntas y almacenar los embeddings en ChromaDB. Al recibir una nueva consulta, genera su vector correspondiente y recupera el más similar desde la base vectorial.

Ambas versiones devuelven la respuesta seleccionada al usuario a través de la API, que posteriormente es mostrada en la interfaz web. El flujo completo está diseñado para funcionar en entorno local, ejecutándose íntegramente en la máquina de desarrollo.

## Implementación de la versión con spaCy

La primera versión del chatbot utiliza spaCy como núcleo de procesamiento lingüístico. Este enfoque aplica técnicas de análisis morfosintáctico y cálculo de similitud semántica sobre los mensajes introducidos por los usuarios, comparándolos contra un conjunto de preguntas predefinidas.

El sistema emplea el modelo es\_core\_news\_md, que proporciona un equilibrio entre rendimiento y precisión para el idioma español. El modelo se carga al iniciar el servidor Flask y permanece residente durante la ejecución para optimizar el rendimiento en consultas sucesivas.

El flujo de procesamiento comienza con la normalización del texto recibido. Esta operación convierte el mensaje a minúsculas, elimina signos de puntuación innecesarios y normaliza los espacios en blanco, reduciendo el impacto de pequeñas variaciones en la entrada.

def normalizar\_texto(texto):

texto = texto.lower()

texto = texto.replace("¿", "").replace("?", "").replace("¡", "").replace("!", "").strip()

return texto

Posteriormente, el sistema compara el mensaje normalizado con cada una de las preguntas almacenadas en el dataset faq\_data.json, que contiene las preguntas principales, sus reformulaciones y las respuestas asociadas. spaCy genera la representación interna de cada frase y calcula la similitud semántica mediante la función similarity():

doc\_usuario = nlp(normalizar\_texto(pregunta\_usuario))

similitud = doc\_usuario.similarity(doc\_pregunta)

El sistema evalúa la similitud obtenida aplicando un esquema de umbrales:

* Si la similitud es igual o superior a 0.75, la respuesta asociada se devuelve directamente.
* Si la similitud se encuentra entre 0.5 y 0.75, se devuelve la respuesta indicando que la coincidencia es baja.
* Si la similitud es inferior a 0.5, el sistema informa al usuario de que no puede ofrecer una respuesta adecuada.

Este mecanismo permite controlar la fiabilidad de las respuestas ofrecidas y evita que el chatbot proporcione respuestas erróneas cuando la consulta se desvía del conocimiento registrado.

El conjunto de datos gestionado tiene el siguiente formato, donde cada pregunta incluye varias reformulaciones posibles:

{

"pregunta": "¿Cuántos créditos tiene el TFG?",

"reformulaciones": [

"¿Cuántos ECTS tiene el Trabajo Fin de Grado?",

"Crédi tos TFG"

],

"respuesta": "El TFG tiene 6 créditos ECTS."

}

El controlador central de la aplicación se encuentra encapsulado en el archivo api.py, que gestiona las peticiones recibidas desde el frontend web y ejecuta la lógica de procesamiento descrita. La respuesta final se entrega en formato JSON para ser mostrada al usuario en la interfaz web.

## Implementación de la versión con embeddings

La segunda versión del chatbot emplea embeddings semánticos como mecanismo principal para determinar la similitud entre la consulta del usuario y el conjunto de preguntas registradas en la base de conocimiento. Este enfoque permite trabajar con reformulaciones y expresiones alternativas de forma más robusta que el modelo basado en análisis gramatical.

Para codificar las preguntas y las consultas se utiliza el modelo all-MiniLM-L6-v2 de Sentence Transformers, diseñado para generar representaciones vectoriales compactas y eficientes de oraciones completas.

Antes de construir los vectores de las preguntas registradas, el sistema normaliza los textos eliminando mayúsculas, signos de puntuación innecesarios y espacios redundantes, reduciendo así el impacto de variaciones superficiales:

def normalizar\_texto(texto):

texto = texto.lower()

texto = texto.replace("¿", "").replace("?", "").replace("¡", "").replace("!", "").strip()

return texto

Las preguntas normalizadas se transforman en vectores mediante el siguiente proceso:

model = SentenceTransformer('all-MiniLM-L6-v2')

embeddings = model.encode(

[normalizar\_texto(q["pregunta"]) for q in preguntas],

normalize\_embeddings=True

)

Una vez generados, los vectores se almacenan localmente utilizando ChromaDB como motor de búsqueda vectorial. Esta base de datos permite realizar búsquedas de similitud de manera eficiente sin necesidad de recalcular los embeddings en cada consulta:

chroma\_client = chromadb.Client()

collection = chroma\_client.create\_collection(name="faq\_embeddings")

collection.add(

documents=[q["respuesta"] for q in preguntas],

embeddings=embeddings.tolist(),

ids=[str(i) for i in range(len(preguntas))]

)

Cuando el usuario envía una nueva consulta, el sistema la normaliza, genera su vector correspondiente y lo compara contra la base vectorial almacenada:

query\_embedding = model.encode([normalizar\_texto(pregunta\_usuario)], normalize\_embeddings=True)

resultados = collection.query(

query\_embeddings=query\_embedding.tolist(),

n\_results=1

)

Si la distancia obtenida se encuentra por debajo de un umbral de similitud establecido (por defecto 0.6), el sistema devuelve la respuesta asociada. Si no se supera este umbral, informa al usuario de que no puede ofrecer una respuesta fiable.

Este enfoque permite ampliar el sistema añadiendo nuevas preguntas sin necesidad de definir manualmente reglas lingüísticas adicionales, lo que simplifica el mantenimiento a medida que el catálogo de consultas crece.

## Diseño del dataset y categorización de respuestas

El sistema trabaja sobre un conjunto de datos previamente estructurado, que constituye la base común tanto para el procesamiento mediante spaCy como para el enfoque basado en embeddings. Este conjunto de preguntas y respuestas reproduce de forma fiel el tipo de consultas administrativas reales que formulan los estudiantes universitarios.

La construcción del dataset inicial se realizó de forma asistida utilizando modelos de lenguaje de OpenAI. A partir de documentos oficiales —normativa académica, manuales de matrícula, procedimientos de Secretaría Virtual y guías institucionales de la Universidad CEU San Pablo— se generaron automáticamente propuestas de preguntas, reformulaciones y respuestas. Este procedimiento permitió acelerar la recopilación inicial de registros, optimizando el proceso de extracción. Tras esta fase automatizada, el conjunto completo de registros fue revisado, validado y corregido manualmente, asegurando la exactitud normativa de las respuestas y la coherencia de las formulaciones.

Cada entrada del dataset inicial (test\_dataset.json) contiene:

* pregunta: enunciado principal de la consulta.
* reformulaciones: variantes reales de expresión de la misma pregunta.
* respuesta: solución exacta a la consulta planteada.

Sobre este dataset base se aplicó posteriormente un proceso adicional de enriquecimiento orientado a su uso en la fase de evaluación comparativa. A partir del dataset inicial, se generó el fichero dataset\_enriquecido.json, que introduce un campo de categorización (categoria) con tres niveles de dificultad:

* Directa: preguntas que coinciden de forma literal con los procedimientos documentados.
* Reformulada: variantes que reproducen la misma consulta aplicando sinónimos, cambios de estructura o expresiones equivalentes.
* Trampa: consultas irrelevantes, incoherentes o imposibles, formuladas para evaluar la robustez de ambos modelos frente a entradas fuera de dominio.

Esta clasificación permitió analizar de forma segmentada el comportamiento de los modelos durante la fase experimental, evaluando no sólo su precisión global, sino también su capacidad de generalización semántica y su resistencia a errores.

El formato definitivo del dataset mantiene una estructura homogénea, extensible y fácilmente escalable:

{

"pregunta": "...",

"reformulaciones": ["...", "...", "..."],

"respuesta": "...",

"categoria": "directa" / "reformulada" / "trampa"

}

Esta estructura permite incorporar futuras ampliaciones al sistema sin necesidad de modificar la arquitectura general de procesamiento ni las funciones de búsqueda semántica.

## Criterios definidos para la evaluación

El sistema se evaluó mediante un conjunto de métricas cuantitativas diseñadas para comparar el comportamiento diferencial de los dos modelos desarrollados. Para ello, se empleó el dataset\_enriquecido.json, categorizado en preguntas directas, reformuladas y preguntas trampa.

Las pruebas se ejecutaron a través de un script de validación automatizada (tester\_comparativo\_v3.py), diseñado para lanzar todas las consultas de forma secuencial sobre ambas versiones del sistema y registrar los siguientes indicadores:

* Porcentaje de acierto por tipo de pregunta: mide la capacidad de los modelos para resolver correctamente preguntas exactas, reformuladas o fuera de dominio.
* Matriz de confusión simplificada: registra los casos de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos, permitiendo analizar el comportamiento de error de cada modelo.
* Tiempo medio de respuesta por modelo: cuantifica la eficiencia computacional de cada enfoque, midiendo el tiempo de inferencia en cada petición.

Este esquema de evaluación permitió realizar una comparativa completa en condiciones controladas, evaluando simultáneamente la precisión semántica, la robustez frente a variabilidad lingüística y el rendimiento operativo de ambos sistemas.

## Interfaz y despliegue funcional

Además del desarrollo técnico de los modelos de procesamiento, se implementó una interfaz conversacional web funcional, que permite simular el uso real del chatbot desde el punto de vista del usuario final.

El diseño de la interfaz se desarrolló en lenguaje HTML y CSS, siguiendo la línea estética corporativa de la Universidad CEU San Pablo, con el objetivo de mimetizar su aspecto y ofrecer una experiencia de usuario coherente con los portales institucionales reales. El estilo visual, los colores, las tipografías y la estructura general se ajustaron para simular cómo podría integrarse este sistema en la página de Secretaría Virtual o en los servicios de consulta académica del CEU.

La aplicación web permite realizar consultas de forma interactiva, mostrando la respuesta generada por los modelos evaluados a partir de las preguntas formuladas. Cada petición realizada desde el frontend se comunica con el backend desarrollado a través de llamadas HTTP asíncronas, enviando la pregunta al servidor API y recuperando la respuesta correspondiente.

Para facilitar su despliegue local durante el desarrollo, el sistema completo se visualizó mediante la extensión "Go Live Server", que permite lanzar un servidor HTTP local simple desde el propio entorno de desarrollo, simulando el comportamiento de la aplicación en un entorno web real.

Las siguientes imágenes muestran la interfaz final implementada durante el proyecto:

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

**Figura 1. Interfaz inicial del chatbot sin interacción**

**Una captura de pantalla de una red social

El contenido generado por IA puede ser incorrecto.**

**Figura 2. Interfaz del chatbot mostrando una respuesta válida tras una reformulación**

**Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.**

**Figura 3. Interfaz del chatbot ante una consulta fuera de dominio**

# Resultados y evaluación comparativa

## Porcentaje de acierto por tipo de pregunta

Para obtener una evaluación detallada del rendimiento de ambos modelos, se realizó un primer análisis de precisión segmentando los datos según el tipo de pregunta. Esta segmentación se basa en la estructura definida previamente en el dataset\_enriquecido.json, que clasifica cada consulta como directa, reformulada o trampa.

El sistema de evaluación automatizado (tester\_comparativo\_v3.py) lanzó todas las preguntas del dataset sobre ambas versiones del sistema. Durante la ejecución, se registraron los aciertos y fallos de cada modelo, considerando como acierto cualquier respuesta que coincidiera, con un umbral de similitud semántica predefinido, con la respuesta esperada. Las preguntas trampa fueron consideradas correctamente resueltas únicamente si el sistema rechazaba ofrecer una respuesta cuando no existía información válida disponible.

El total de preguntas procesadas en esta primera fase fue de 248, distribuidas de la siguiente forma:

* Preguntas directas: 24
* Reformulaciones: 120
* Preguntas trampa: 4

El objetivo de este primer análisis era observar la capacidad de los modelos para resolver correctamente preguntas formuladas de forma estándar, gestionar reformulaciones habituales entre los estudiantes y filtrar correctamente preguntas fuera de dominio.

Los porcentajes de acierto obtenidos se resumen en la siguiente tabla:

| Categoría | Embeddings | SpaCy |
| --- | --- | --- |
| Directa | 87.50% | 91.67% |
| Reformulada | 100.00% | 100.00% |
| Trampa | 100.00% | 25.00% |

En preguntas directas, ambos modelos mantienen un rendimiento elevado, con diferencias mínimas entre ellos. El resultado es especialmente sólido en las reformulaciones, donde ambos sistemas alcanzan el 100% de acierto, lo que indica que son capaces de gestionar correctamente las variantes reales de expresión incluidas en el dataset.

Sin embargo, las diferencias más relevantes aparecen en la categoría de trampas. El modelo de embeddings demostró capacidad total de detección de preguntas fuera de dominio, evitando generar respuestas erróneas, mientras que spaCy incurrió en fallos significativos, proporcionando respuestas en tres de las cuatro consultas trampa. Este primer análisis evidencia la superioridad de embeddings en la detección de límites de conocimiento, anticipando lo que se observará posteriormente en la matriz de confusión.

## Matriz de confusión simplificada

Más allá del análisis porcentual de aciertos, resulta necesario evaluar la precisión de los modelos considerando el tipo de fallo cometido en cada caso. Para ello se construyó una matriz de confusión simplificada, adaptada al contexto del sistema conversacional, diferenciando:

* Verdaderos positivos (TP): el sistema responde correctamente cuando existe una respuesta válida.
* Falsos positivos (FP): el sistema ofrece una respuesta cuando no existe información válida (preguntas trampa).
* Verdaderos negativos (TN): el sistema identifica correctamente la ausencia de respuesta en preguntas trampa.
* Falsos negativos (FN): el sistema no proporciona respuesta cuando sí existía una respuesta válida.

El volumen total de preguntas procesadas para este análisis fue el mismo que en el apartado anterior (248 preguntas).

Los resultados obtenidos fueron los siguientes:

| **Modelo** | **TP** | **FP** | **TN** | **FN** |
| --- | --- | --- | --- | --- |
| Embeddings | 141 | 0 | 4 | 3 |
| SpaCy | 142 | 3 | 1 | 2 |

El modelo de embeddings alcanzó un comportamiento óptimo en la gestión de preguntas fuera de dominio, evitando generar ningún falso positivo. Esto indica que fue capaz de detectar correctamente los límites de su base de conocimiento, proporcionando únicamente respuestas en los casos en los que existía información válida.

En el caso de spaCy, el comportamiento en los verdaderos positivos es comparable, pero incurrió en tres falsos positivos al ofrecer respuestas inapropiadas en preguntas sin solución registrada. Este comportamiento refleja una menor capacidad para filtrar consultas fuera de dominio, lo que implica un mayor riesgo de error en escenarios reales de despliegue donde pueden aparecer preguntas desconocidas o mal formuladas.

La existencia de falsos negativos en ambos modelos refleja aquellas situaciones en las que, pese a existir respuesta válida, el sistema no logró alcanzar el umbral de similitud necesario para ofrecer una respuesta. Estos casos suponen una oportunidad de mejora para el ajuste fino de los parámetros de umbral y para la incorporación de nuevas reformulaciones en la base de conocimiento.

## Tiempos de respuesta

Además de la precisión en las respuestas, se analizó el tiempo medio de inferencia de cada modelo para cada consulta procesada. Esta métrica permite evaluar la eficiencia operativa de ambos sistemas, especialmente relevante en entornos donde el sistema debe ofrecer respuestas en tiempo real a un volumen elevado de usuarios concurrentes.

Durante la fase de validación, el script tester\_comparativo\_v3.py registró el tiempo de respuesta de cada petición individual en milisegundos, considerando el ciclo completo de procesamiento interno de cada backend.

Los tiempos medios obtenidos fueron los siguientes:

| **Modelo** | **Tiempo medio de respuesta** |
| --- | --- |
| Embeddings | 20.97 ms |
| SpaCy | 721.84 ms |

El modelo basado en embeddings muestra un tiempo de respuesta significativamente inferior. Esta diferencia se explica por el distinto enfoque técnico de cada sistema. Embeddings trabaja directamente sobre representaciones vectoriales precalculadas y optimizadas en ChromaDB, lo que permite recuperar la respuesta más cercana mediante búsquedas rápidas de similitud.

Por el contrario, spaCy ejecuta en cada consulta el análisis morfosintáctico completo del mensaje, generando el árbol de dependencias y recalculando las representaciones vectoriales en tiempo real para cada comparación contra el dataset. Este procesamiento más pesado impacta directamente en el tiempo de respuesta agregado.

En un escenario de despliegue real, esta diferencia de tiempos influiría en la escalabilidad del sistema, el consumo de recursos y la capacidad de atender múltiples peticiones concurrentes sin degradación de servicio.

# Discusión

El análisis comparativo de resultados permite extraer varias conclusiones relevantes acerca del comportamiento diferencial de ambos modelos desarrollados. Aunque los porcentajes globales de acierto son elevados en los dos enfoques, las diferencias de rendimiento se hacen especialmente visibles al observar el comportamiento por tipo de pregunta y la gestión de los casos fuera de dominio.

En las preguntas directas, ambas versiones alcanzan niveles de precisión muy similares. spaCy obtiene un porcentaje ligeramente superior (91,67%) frente al 87,50% de embeddings. Esta pequeña ventaja se explica por la naturaleza de la comparación basada en similitud sintáctica, que favorece coincidencias estructurales directas entre las preguntas formuladas y las almacenadas en el dataset.

En las reformulaciones, ambos sistemas logran un 100% de acierto, reflejando que la ampliación del dataset con variantes reales de expresión ha permitido capturar correctamente las distintas formas en las que los estudiantes pueden formular una misma consulta. Esta categoría confirma que ambos modelos son capaces de gestionar adecuadamente la variabilidad natural del lenguaje cuando se dispone de suficientes ejemplos de reformulación.

La diferencia más significativa aparece en el tratamiento de preguntas trampa. El modelo basado en embeddings alcanza un 100% de acierto en esta categoría, detectando correctamente la falta de coincidencia semántica y evitando ofrecer respuestas inapropiadas. SpaCy, en cambio, incurre en falsos positivos en tres de los cuatro casos, generando respuestas cuando no debía. Este comportamiento refleja una limitación inherente a su método de comparación superficial, que carece de un mecanismo eficaz para establecer cuándo una consulta está fuera del conocimiento predefinido.

Los resultados de la matriz de confusión confirman esta tendencia. Embeddings no presenta falsos positivos, mientras que spaCy acumula errores en esa categoría. Aunque ambos presentan falsos negativos residuales, el control de error en preguntas trampa posiciona a embeddings como un modelo más robusto para entornos donde es prioritario evitar respuestas incorrectas.

La comparación de tiempos de respuesta refuerza las diferencias técnicas entre ambos sistemas. El modelo embeddings muestra una latencia media de 20,97 ms, mientras que spaCy requiere 721,84 ms por consulta. Esta diferencia sustancial se debe al uso de representaciones vectoriales precalculadas en el modelo embeddings, frente al procesamiento morfosintáctico dinámico que ejecuta spaCy en cada consulta.

En conjunto, los resultados obtenidos respaldan la superioridad del modelo embeddings cuando el sistema debe operar en entornos controlados, con dominio cerrado de conocimiento, donde es imprescindible mantener un alto nivel de precisión semántica, minimizar el riesgo de alucinaciones conversacionales y optimizar los recursos computacionales.

# Conclusiones finales

## Conclusiones técnicas

El trabajo desarrollado ha permitido construir y validar un sistema conversacional orientado a resolver consultas administrativas frecuentes de estudiantes universitarios, empleando técnicas de procesamiento de lenguaje natural sobre un dominio cerrado de conocimiento.

Se han implementado dos versiones funcionales del chatbot, diferenciadas por el núcleo de procesamiento lingüístico utilizado: una basada en análisis morfosintáctico mediante spaCy y otra fundamentada en búsqueda semántica con embeddings y vectorización previa de las preguntas almacenadas.

La construcción del dataset inicial fue optimizada mediante el uso de modelos de lenguaje para la generación asistida de preguntas, reformulaciones y respuestas, a partir de la documentación oficial de procedimientos académicos de la Universidad CEU San Pablo. Posteriormente, el dataset fue enriquecido mediante un proceso de categorización automática, lo que permitió realizar una evaluación segmentada del rendimiento de ambos modelos.

Los resultados experimentales obtenidos reflejan un alto nivel de precisión global en ambos enfoques, especialmente en la gestión de reformulaciones, donde ambos modelos alcanzan el 100% de acierto. Sin embargo, las diferencias aparecen en el manejo de casos fuera de dominio, donde el modelo embeddings demuestra una mayor robustez semántica al evitar generar respuestas inapropiadas en preguntas trampa.

El análisis de tiempos de respuesta confirma también la superioridad del modelo embeddings en términos de eficiencia computacional, con un tiempo medio de inferencia significativamente inferior al de spaCy, lo que lo posiciona como una solución más escalable para entornos de atención en tiempo real.

En conjunto, la combinación de precisión semántica, control de errores y eficiencia operativa posiciona al modelo basado en embeddings como una solución más adecuada para el despliegue real de este tipo de sistemas conversacionales en entornos universitarios.

## Aprendizaje personal del proceso

El desarrollo de este proyecto ha supuesto un reto real tanto a nivel técnico como organizativo. Desde el inicio, la elección de un tema basado en procesamiento de lenguaje natural supuso afrontar un área en la que inicialmente no disponía de experiencia previa, lo que obligó a dedicar un tiempo considerable a la investigación teórica inicial para poder comprender los fundamentos de los modelos empleados.

La construcción del sistema ha exigido trabajar tanto en el diseño del backend (desarrollo de las APIs de procesamiento, integración de los modelos spaCy y embeddings, gestión de datos, lógica de control, configuración de servidores locales y manejo de entornos virtuales) como en el frontend (diseño de la interfaz conversacional web, integración de llamadas asíncronas, tratamiento de respuestas del backend y manejo de flujos de usuario).

Una parte central del aprendizaje ha sido el proceso continuo de prueba y error durante la implementación. Cada fase del código ha requerido buscar información adicional de documentación técnica, foros de desarrolladores, papers de investigación y ejemplos prácticos que permitieran resolver problemas específicos. Desde ajustes de similitud semántica, configuración de los servidores de vector store, manejo de JSON en las APIs, hasta el control del flujo conversacional, cada módulo ha sido iterado en múltiples ocasiones hasta alcanzar una versión estable y funcional.

Durante todo el proceso de desarrollo, la propia inteligencia artificial ha sido también una herramienta de apoyo clave. El uso de modelos como ChatGPT ha permitido agilizar tareas de generación de datos, validar estructuras de preguntas, obtener fragmentos de código funcional, aclarar conceptos técnicos complejos y acelerar la curva de aprendizaje en temas inicialmente desconocidos. Este uso asistido de IA refleja precisamente uno de los objetivos planteados en el proyecto: explorar cómo estas tecnologías pueden no solo formar parte de los modelos de respuesta, sino también convertirse en asistentes efectivos en los propios procesos de programación y desarrollo de sistemas inteligentes.

A nivel organizativo, el desarrollo del proyecto ha coincidido con el tramo final de la carrera universitaria y su compatibilización con las actividades académicas y profesionales. La planificación inicial tuvo que adaptarse varias veces para encajar los tiempos de aprendizaje real que requería cada fase. En algunos momentos, el proyecto llegó a parecer un objetivo demasiado ambicioso, pero progresivamente, mediante trabajo constante, reajuste de objetivos intermedios y aprendizaje incremental, ha sido posible completarlo de forma satisfactoria.

Este proceso ha enseñado la importancia de la resiliencia técnica: enfrentarse a errores, buscar soluciones alternativas, ajustar los planteamientos iniciales y adaptar el diseño al funcionamiento real de los modelos. Además de adquirir conocimientos en técnicas de NLP, embeddings, desarrollo de APIs y pruebas automatizadas, el proyecto ha servido para consolidar competencias en análisis de problemas, toma de decisiones técnicas y gestión de proyectos tecnológicos reales.

## Mejoras futuras y aplicación real

Además de las funcionalidades ya implementadas, el sistema permite distintas líneas de mejora que ampliarían su impacto y viabilidad en un entorno universitario real. En primer lugar, sería posible expandir el dataset para cubrir nuevas categorías temáticas, como becas, movilidad internacional, prácticas externas o procedimientos de evaluación, lo que aumentaría su utilidad para un mayor número de estudiantes. También podría implementarse un sistema de validación avanzada que incorpore indicadores de confianza ante entradas poco comunes o mal formuladas, permitiendo filtrar mejor aquellas consultas que no pertenecen al dominio previsto.

Otra evolución relevante sería su integración directa en la infraestructura web del CEU. El sistema podría desplegarse como microservicio accesible desde la Secretaría Virtual o desde el portal del alumno, mediante una API REST o un widget embebido. Esto permitiría que los estudiantes interactuaran con el chatbot sin salir de los canales institucionales. Además, podría incorporarse un sistema de retroalimentación con reentrenamiento continuo, donde las consultas no resueltas quedaran registradas y fueran analizadas por los responsables académicos para mejorar la cobertura del sistema.

Por último, la arquitectura modular permitiría escalar esta solución a otros centros del CEU o incluso a otras universidades, adaptando únicamente el dataset base sin necesidad de modificar el núcleo técnico del sistema. Todas estas mejoras podrían aplicarse de forma progresiva, manteniendo la estabilidad de la solución ya desarrollada.

# Bibliografía

# Anexos

## Ejemplos de preguntas y respuestas del dataset

## Fragmentos de código representativos

## Capturas de la interfaz funcional implementada