Escuela
Politécnica
Superior

Asistente conversacional inteligente para estudiantes universitarios

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Álvaro Lario Sánchez
Tutor/es:
Juan Carlos Martínez Sevilla
Jorge Calvo Zaragoza

Universitat d'Alacant
Universidad de Alicante

Asistente conversacional inteligente para estudiantes universitarios

Asistente mediante arquitectura híbrida basada en *Retrieval-Augmented Generation*

Autor

Álvaro Lario Sánchez

Tutor/es

Juan Carlos Martínez Sevilla

Departamento de Lenguajes y Sistemas Informáticos

Jorge Calvo Zaragoza

Departamento de Lenguajes y Sistemas Informáticos

Grado en Ingeniería Informática

Escuela Politécnica Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2025

Preámbulo

"La realización de este Trabajo Fin de Grado surge del interés personal por el campo del procesamiento del lenguaje natural y su aplicación práctica en entornos educativos. Durante mis estudios en el Grado en Ingeniería Informática, he podido comprobar las dificultades que encuentran muchos estudiantes para acceder de forma rápida y clara a información académica esencial, como normativas, calendarios, planes de estudios o trámites administrativos. Esta experiencia ha motivado el diseño de una solución basada en inteligencia artificial que permita automatizar y mejorar ese proceso de consulta.

El trabajo se centra en el desarrollo de un asistente conversacional inteligente para estudiantes universitarios, utilizando una arquitectura híbrida basada en *Retrieval-Augmented Generation* (RAG). La finalidad del proyecto es demostrar la viabilidad de aplicar modelos de lenguaje ligeros y herramientas de código abierto para construir un sistema capaz de responder preguntas académicas de manera eficiente, contextualizada y ejecutable en entornos locales con recursos limitados.

El alcance del proyecto abarca tanto el diseño arquitectónico como la implementación técnica del sistema, así como su evaluación mediante pruebas funcionales. Su estructura modular permite escalarlo a otros entornos universitarios y contextos institucionales en el futuro.

Este trabajo ha sido posible gracias al apoyo y guía de mis tutores, Juan Carlos Martínez Sevilla y Jorge Calvo Zaragoza, cuyas observaciones y aportaciones han sido clave durante todo el proceso. También quiero expresar mi profundo agradecimiento a mi familia, sin ellos nada de estos hubiese sido posible."

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas que han formado parte de este camino y han contribuido, directa o indirectamente, a la realización de este Trabajo Fin de Grado.

En primer lugar, agradezco profundamente a mis tutores, Juan Carlos Martínez Sevilla y Jorge Calvo Zaragoza, por su dedicación, orientación técnica y confianza durante todo el proceso. Sus aportaciones y sugerencias han sido clave para la evolución del proyecto.

A mi familia, por su paciencia, su apoyo incondicional y por creer en mí incluso en los momentos de mayor incertidumbre. Gracias por estar siempre ahí, ofreciendo ánimo, comprensión y estabilidad.

A mi pareja, por acompañarme con cariño, motivarme día a día y recordarme que incluso en los momentos más exigentes es posible encontrar equilibrio, sentido y alegría. Su apoyo ha sido fundamental a nivel personal y emocional.

A mis amigos, por su cercanía, por hacer más llevaderos los años de estudio y por recordarme la importancia de compartir, desconectar y disfrutar del proceso. Su compañía ha sido clave para mantener la motivación.

Este trabajo no solo representa el cierre de una etapa académica, sino también el reflejo del acompañamiento de todas las personas que me han rodeado y que, de una forma u otra, han contribuido a que este reto se haya convertido en una realidad.

A quienes me han acompañado, animado y creído en mí.
Este logro también es vuestro.

Aquellos que están lo suficientemente locos
como para pensar que pueden cambiar el mundo,
son los que lo hacen.

Steve Jobs.

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1 Introducción

En el entorno universitario actual, los estudiantes se enfrentan a una creciente complejidad administrativa y académica, caracterizada por la dispersión de la información, la diversidad de normativas y la constante necesidad de resolver dudas específicas sobre asignaturas, calendarios, planes de estudio o trámites institucionales. Esta situación genera una carga operativa tanto para el alumnado como para el personal de atención, especialmente cuando las consultas se repiten de forma recurrente y requieren respuestas personalizadas o contextualizadas.

La automatización de la atención a estudiantes mediante sistemas conversacionales ha demostrado ser una solución prometedora en múltiples instituciones educativas. Sin embargo, gran parte de los chatbots implementados hasta ahora siguen modelos basados en reglas rígidas o flujos predefinidos, lo que limita su capacidad de adaptación y comprensión semántica ante preguntas más complejas o fuera de lo previsto.

En este contexto, los Modelos de Lenguaje de Gran Escala (LLMs), como GPT (Brown y cols., 2020a), Claude (Bai y cols., 2022) o DeepSeek (DeepSeek-AI, 2024), han marcado un hito en el desarrollo de sistemas conversacionales inteligentes. Estos modelos permiten generar respuestas contextualizadas, coherentes y con un lenguaje natural fluido. Aun así, presentan limitaciones cuando deben proporcionar información actualizada o específica que no fue parte de su entrenamiento. Para solventar esta carencia, ha surgido un enfoque híbrido conocido como Retrieval-Augmented Generation (RAG), el cual combina la capacidad generativa de los LLMs con mecanismos de recuperación de información semántica desde una base documental específica (Lewis y cols., 2020).

Este proyecto tiene como objetivo el desarrollo de un asistente conversacional inteligente para estudiantes universitarios, utilizando una arquitectura híbrida basada en RAG. La solución propuesta permite a los usuarios consultar de forma natural y eficiente una base documental compuesta por guías docentes, reglamentos, horarios y otros documentos insti-

tucionales, obteniendo respuestas precisas y adaptadas a su contexto.

Para ello, se ha implementado una interfaz propia basada en la biblioteca Gradio[1], que interactúa directamente con un backend desarrollado en Python[2] y desplegado mediante FastAPI[3].

Con este Trabajo Fin de Grado se busca no solo implementar una solución técnica funcional, sino también analizar su viabilidad, eficacia y aplicabilidad dentro del ecosistema universitario. El enfoque propuesto, centrado en el uso de tecnologías avanzadas de procesamiento del lenguaje natural y recuperación semántica, pretende sentar las bases para el desarrollo de asistentes académicos escalables, reutilizables y alineados con las necesidades reales de los estudiantes.

1.1 Evolución de los Modelos de Lenguaje de gran tamaño

Los modelos de lenguaje han evolucionado significativamente en las últimas décadas, transformando por completo el Procesamiento del Lenguaje Natural (NLP). Estos modelos son capaces de comprender, generar y contextualizar texto de forma mucho más precisa y coherente que las tecnologías anteriores, permitiendo crear asistentes conversacionales más naturales, flexibles y adaptativos.

Los primeros enfoques se basaban en modelos estadísticos como los n-gramas (Jurafsky y Martin, 2009), que predecían palabras en función de la frecuencia de aparición de secuencias previas en un corpus. Estos modelos eran muy limitados en su capacidad para entender contexto o significados complejos.

Posteriormente, con el auge del aprendizaje profundo, surgieron numerosos modelos basados en Redes Neuronales Recurrentes (RNN) (Elman, 1990) y Long Short-Term Memory (LSTM) (Hochreiter y Schmidhuber, 1997), capaces de capturar dependencias a largo plazo en el texto. Sin embargo, estos modelos presentaban dificultades para escalar y sufrían problemas como la pérdida de información contextual.

El verdadero punto de inflexión llegó con la introducción de Transformers (Vaswani y cols., 2017), una arquitectura basada en mecanismos de atención que permitía procesar secuencias

[1]Documentación de Gradio
[2]Documentación de Python
[3]Documentación de FastAPI

de texto de forma más eficiente y paralelizable. A partir de ahí, comenzaron a desarrollarse modelos preentrenados a gran escala, como BERT (Devlin y cols., 2019) para tareas de comprensión del lenguaje y GPT (Brown y cols., 2020b) para generación de texto.

Estos LLMs marcan un salto cualitativo al ser capaces de generar texto coherente, realizar tareas de Quality Assurance (QA), resumir, traducir y más, con un rendimiento cercano al humano en muchas tareas. La más reciente evolución, como GPT-4, incorpora aprendizajes multimodales y arquitecturas aún más complejas, además de mejoras en alineación, contexto y precisión.

1.2 Del chatbot clásico al chatbot basado en LLMs

Los chatbots son programas diseñados para simular una conversación con humanos. Aunque hoy en día están estrechamente relacionados con los LLMs, lo cierto es que su historia comienza mucho antes del auge de la inteligencia artificial moderna.

El primer chatbot reconocido fue ELIZA, creado por Weizenbaum (1966). Simulaba un terapeuta mediante un sistema de coincidencia de patrones y reglas predefinidas. Le siguieron otros como PARRY (Colby y cols., 1972), que imitaba el comportamiento de una persona con esquizofrenia.

Estos primeros chatbots no entendían realmente el lenguaje, sino que respondían mediante reglas sintácticas simples, sin comprensión semántica ni aprendizaje automático.

Durante las décadas siguientes, los chatbots evolucionaron incorporando técnicas más avanzadas, como árboles de decisión, procesamiento de lenguaje natural básico NLP y motores de diálogo. Sin embargo, la mayoría seguía siendo dependiente de estructuras fijas y programadas manualmente.

Con la introducción del modelo Transformer (Vaswani y cols., 2017), y posteriormente modelos como GPT, BERT y T5 (Raffel y cols., 2020), el paradigma cambió radicalmente. Los LLMs comenzaron a entender el contexto, generar lenguaje natural coherente, e incluso razonar a partir del texto.

La aparición de GPT-3 y especialmente ChatGPT (OpenAI, 2022), marcó la llegada de una nueva generación de chatbots, capaces de mantener conversaciones fluidas, adaptarse al estilo del usuario y generar contenido de alta calidad.

Los chatbots actuales, como ChatGPT, Claude o Gemini (DeepMind, 2023b), ya no dependen de árboles de decisión o reglas fijas. Están impulsados por modelos de lenguaje preentrenados con billones de palabras, capaces de realizar tareas conversacionales, técnicas y creativas.

Esta evolución no solo ha mejorado la naturalidad de las conversaciones, sino que ha ampliado drásticamente las capacidades de los chatbots, integrándolos en áreas como la educación, la atención al cliente, el desarrollo de software, y más (ver Figura 1.1).

Figura 1.1: Línea cronológica de los principales chatbots

1.3 Una nueva generación de chatbots inteligentes

En la actualidad, ha surgido una nueva tecnología que está marcando un punto de inflexión en los asistentes conversacionales modernos: RAG.

Esta técnica combina el poder generativo de los modelos de lenguaje con mecanismos de recuperación de información, logrando resultados más actualizados, precisos y verificados.

El concepto de RAG fue introducido formalmente por Meta AI (Lewis y cols., 2020) quien presentó RAG como una arquitectura que fusiona modelos de recuperación de documentos con modelos generativos tipo encoder-decoder. Fue especialmente diseñada para tareas knowledge-intensive, como QA, donde el modelo necesita acceder a hechos específicos y actualizados.

RAG es una arquitectura híbrida que permite a un modelo generativo (como GPT o BERT) consultar una base de datos, documentos o conocimiento externo antes de generar una respuesta. En lugar de generar solo con lo que ha aprendido en su entrenamiento, accede a una fuente actualizada y específica. Es importante en los chatbots ya que mejora la precisión de las respuestas, sobre todo cuando se necesita información reciente, un chatbot con tecnología RAG puede acceder a documentos internos, bases de conocimiento empresariales, wikis, bases

de datos, etc. Además, evita las famosas alucinaciones que pueden ocurrir en modelos LLM sin acceso a fuentes externas.

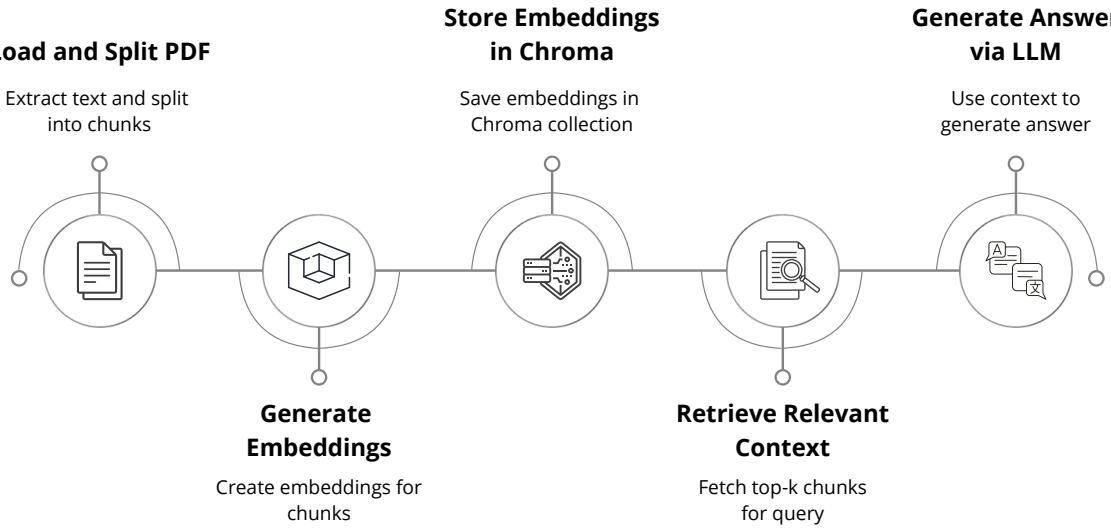


Figura 1.2: Ejemplo de Retrieval Augmented Generation

La base de este trabajo se muestra en la Figura 1.2, la cual ilustra el concepto fundamental sobre el que se estructura este proyecto. En esta figura, se pueden observar algunos de los aspectos clave que definimos como parte de nuestro enfoque, los cuales serán analizados en mayor profundidad más adelante. En particular, destacamos las tecnologías principales que vamos a utilizar para desarrollar nuestro chatbot académico. Estas incluyen técnicas avanzadas en el manejo de datos, NLP, y algoritmos de recuperación de información, que forman el núcleo de la arquitectura de nuestro sistema. A lo largo de este trabajo, explicaremos cómo cada una de estas tecnologías contribuye a la creación de un chatbot capaz de interactuar de manera efectiva con los usuarios, proporcionando respuestas precisas y relevantes basadas en el conocimiento almacenado en el sistema.

2 Marco Teórico

Antes de realizar el proyecto que tenemos entre manos, es necesario hacer un breve recorrido histórico que nos permita comprender el origen y la evolución de la Inteligencia Artificial (IA). Aunque hoy en día asociamos la IA con sistemas avanzados como asistentes virtuales, coches autónomos o chatbots inteligentes, sus raíces se remontan a mediados del siglo XX, cuando investigadores comenzaron a plantearse si las máquinas podían imitar ciertas capacidades humanas, como razonar, aprender o incluso tomar decisiones.

La pregunta clave —¿pueden las máquinas pensar?— fue formulada por Turing (1950), marcando un hito fundacional en el campo de la IA. A partir de ese momento, se empezó a desarrollar una nueva rama de la informática con el objetivo de dotar a las máquinas de un comportamiento inteligente. Desde los primeros algoritmos basados en reglas hasta las modernas redes neuronales profundas, la IA ha recorrido un largo camino, alimentada por avances tanto en teoría como en capacidad computacional.

Este marco teórico pretende contextualizar el desarrollo del proyecto mediante una visión progresiva de los conceptos clave, empezando por el nacimiento de la IA, pasando por los primeros modelos de aprendizaje automático como el perceptrón, hasta llegar a los modelos de lenguaje de gran escala (LLMs) y su aplicación en sistemas conversacionales mediante técnicas como RAG.

2.1 Orígenes de la Inteligencia Artificial

La IA como campo formal se consolidó en el verano de 1956, durante el histórico taller de investigación en Dartmouth College (EE.UU.). Allí se acuñó por primera vez el término IA, y se estableció el objetivo de encontrar métodos que permitieran a una máquina razonar como un ser humano.

Durante esta etapa inicial, se confiaba en que la IA podría resolverse en poco tiempo. Se crearon programas que resolvían problemas matemáticos, jugaban al ajedrez o demostraban teoremas. Sin embargo, estas soluciones eran muy limitadas y dependían de reglas explícitas. Con el paso del tiempo, estas expectativas tan optimistas se fueron enfrentando a obstáculos técnicos y teóricos, lo que llevó a los llamados "inviernos de la IA", periodos donde el entusiasmo y la financiación disminuyeron drásticamente.

2.2 El Perceptrón y los inicios del Aprendizaje Automático

A medida que se reconocieron las limitaciones de la IA simbólica basada en reglas fijas escritas por humanos, surgió una nueva corriente de pensamiento: permitir que las máquinas aprendan por sí mismas a partir de datos. Esto dio lugar al campo del aprendizaje automático (machine learning).

Uno de los primeros modelos inspirados en el funcionamiento del cerebro humano fue el Perceptrón, propuesto por Rosenblatt (1958). El Perceptrón es una neurona artificial capaz de clasificar datos linealmente separables.

Aunque simple, su importancia radica en haber sentado las bases del desarrollo de redes neuronales artificiales más complejas.

¿Cómo funciona el Perceptrón? El Perceptrón toma varias entradas numéricas, les asigna un peso a cada una, las suma, y pasa el resultado por una función de activación (generalmente el signo o una función escalón) para determinar la salida (ver Ecuación 2.1). Su objetivo es aprender los pesos adecuados para clasificar correctamente los ejemplos de entrenamiento.

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.1}$$

2.3 ¿Cómo aprenden las redes neuronales?

Las redes neuronales artificiales aprenden ajustando los pesos de sus conexiones para minimizar el error entre sus predicciones y los resultados esperados. Este proceso de ajuste se conoce como entrenamiento, y se basa en dos técnicas fundamentales: el descenso por gradiente y la retropropagación del error (backpropagation).

Descenso por Gradiente

El descenso por gradiente (gradient descent) es un algoritmo de optimización que permite encontrar los parámetros (pesos y bias) que minimizan una función de coste (o pérdida). En redes neuronales, esta función mide cuán lejos está la salida de la red con respecto al valor correcto.

La idea es sencilla: se calculan los gradientes de la función de pérdida respecto a cada parámetro, y luego se actualizan los valores de los pesos moviéndose "cuesta abajo" en la dirección en la que la función de error disminuye más rápidamente.

$$\theta := \theta - \eta \cdot \nabla_\theta J(\theta)$$

Retropropagación del Error (Backpropagation)

El descenso por gradiente necesita calcular cómo cada peso contribuye al error final. Esto se logra con el algoritmo de retropropagación del error, introducido formalmente por Rumelhart y cols. (1986).

La retropropagación permite calcular los gradientes de la función de coste en redes multicapa aplicando la regla de la cadena del cálculo diferencial. El algoritmo consta de tres fases:

1. Propagación hacia adelante: se calcula la salida de la red.

2. Cálculo del error: se mide la diferencia entre la salida real y la esperada.

3. Propagación hacia atrás: se calcula el gradiente del error desde la salida hasta las capas anteriores, actualizando los pesos en consecuencia.

Esta combinación hace posible entrenar redes neuronales profundas y es uno de los pilares del aprendizaje profundo moderno.

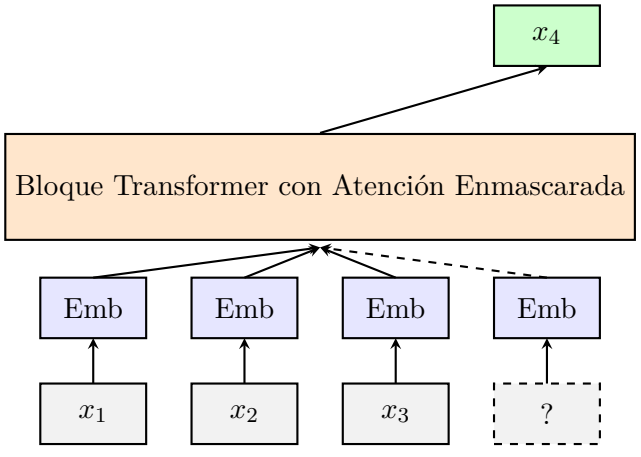
2.4 Modelos de Lenguaje de Gran Escala (LLMs)

Los LLMs son una categoría de modelos de inteligencia artificial entrenados con enormes cantidades de texto para aprender las probabilidades de ocurrencia de palabras y frases en un lenguaje natural. Estos modelos han revolucionado el campo del NLP, permitiendo a las máquinas comprender, generar y responder en lenguaje humano con una fluidez sin precedentes.

¿Cómo funcionan?

Un Modelo de Lenguaje de Gran Escala (LLM) predice la siguiente palabra en una secuencia a partir del contexto anterior. Para lograrlo, se entrena sobre grandes corpus de texto, ajustando sus millones o incluso billones de parámetros para que capte las estructuras gramaticales, patrones semánticos, relaciones entre conceptos y mucho más.

La arquitectura más influyente en estos modelos es Transformer, presentada por Vaswani y cols. (2017). Esta arquitectura reemplaza mecanismos tradicionales como las redes recurrentes por mecanismos de atención, lo que permite paralelizar el entrenamiento y capturar relaciones de largo alcance entre palabras (ver Figura 2.1).



El modelo predice la siguiente palabra x_4 condicionada en x_1, x_2, x_3

Figura 2.1: Funcionamiento de la arquitectura Transformer: la salida x_4 se genera utilizando únicamente el contexto anterior mediante mecanismos de atención enmascarada.

Ejemplos de LLMs actuales

A lo largo de los últimos años, han surgido numerosos LLMs desarrollados por distintas organizaciones, cada uno con enfoques y capacidades particulares. Entre los más destacados se encuentran:

- **BERT (Bidirectional Encoder Representations from Transformers, Google)**: Orientado a la comprensión de texto, utiliza atención bidireccional y está optimizado para tareas como clasificación, respuesta a preguntas o análisis de sentimientos (Devlin y cols., 2019).

- **T5 (Text-to-Text Transfer Transformer, Google)**: Propone convertir cualquier tarea de procesamiento del lenguaje en un problema de traducción de texto a texto, unificando múltiples tareas bajo una misma arquitectura (Raffel y cols., 2020).

- **PaLM (Pathways Language Model, Google)**: Destacado por su escalabilidad y rendimiento multimodal, entrenado con cientos de miles de millones de parámetros, sobresale en tareas que requieren razonamiento complejo (Chowdhery y cols., 2022).

- **LLaMA (Large Language Model Meta AI)**: Diseñado por Meta para la investigación, ofrece modelos ligeros y eficientes ampliamente utilizados como base para desarrollos open-source y despliegues en entornos locales (Touvron y cols., 2023).

- **Claude (Anthropic) y Gemini (Google DeepMind)**: Representan una nueva generación de modelos conversacionales avanzados que incorporan principios de alineación, reducción de alucinaciones y razonamiento ético, siendo empleados en contextos productivos y empresariales (Bai y cols., 2022; DeepMind, 2023a).

Estos modelos presentan una capacidad emergente de generalización, lo que les permite abordar tareas para las que no han sido específicamente entrenados, mediante estrategias como zero-shot (sin ejemplos) o few-shot learning (con muy pocos ejemplos). Esta propiedad ha abierto la puerta a aplicaciones en dominios altamente variados, desde la educación hasta la biomedicina.

Aplicaciones de los LLMs

Los LLMs han revolucionado el campo del NLP debido a su versatilidad y capacidad de adaptación a tareas lingüísticas complejas. Entre sus principales aplicaciones prácticas se encuentran:

- **Asistentes conversacionales y chatbots**: utilizados en plataformas como *ChatGPT*[1] (OpenAI, 2022), *Claude*[2] (Bai y cols., 2022), o *Gemini*[3] (DeepMind, 2023b), permiten mantener conversaciones fluidas, adaptativas y contextualizadas con los usuarios.
- **Traducción automática**: modelos como *mBART*[4] (F. AI, 2020) o *NLLB-200*[5] (M. AI, 2022) son capaces de traducir entre decenas de idiomas con gran fidelidad contextual.
- **Resúmenes automáticos**: herramientas como *SciSummary*[6] (SciSummary, 2023) aplican modelos de lenguaje para generar resúmenes científicos a partir de publicaciones académicas, facilitando la comprensión de artículos extensos.
- **Análisis de sentimientos**: plataformas como *MonkeyLearn*[7] (MonkeyLearn, 2023) permiten clasificar opiniones de usuarios en redes sociales o encuestas, utilizando LLMs entrenados en clasificación emocional.
- **Búsqueda semántica avanzada**: soluciones como *Haystack*[8] (deepset, 2023) o *Google Cloud Vertex AI Search*[9] (Cloud, 2023) utilizan embeddings generados por LLMs para localizar información relevante incluso si no hay coincidencia textual exacta.
- **Generación de código**: asistentes como *GitHub Copilot*[10] (GitHub, 2021) o *CodeWhisperer*[11] (Services, 2022) integran modelos de lenguaje (como Codex) para generar y completar código en múltiples lenguajes de programación.

[1] https://chat.openai.com
[2] https://claude.ai
[3] https://deepmind.google/technologies/gemini
[4] https://huggingface.co/facebook/mbart-large-50-many-to-many-mmt
[5] https://ai.facebook.com/research/publications/nllb-200
[6] https://scisummary.com
[7] https://monkeylearn.com/sentiment-analysis/
[8] https://haystack.deepset.ai/
[9] https://cloud.google.com/vertex-ai/docs/search
[10] https://github.com/features/copilot
[11] https://aws.amazon.com/codewhisperer/

En este trabajo, los LLMs se utilizan como núcleo del sistema conversacional del chatbot, ofreciendo respuestas naturales y contextualizadas a las consultas del usuario, especialmente cuando se combinan con técnicas de recuperación de información, como veremos más adelante.

2.5 Chatbots inteligentes

A la hora de implementar asistentes conversacionales, existen distintas estrategias según el objetivo, los recursos disponibles y el tipo de interacción que se desea lograr. Este trabajo adopta un enfoque basado exclusivamente en el paradigma RAG, que permite generar respuestas contextualizadas a partir de documentos reales sin necesidad de flujos conversacionales predefinidos.

En lugar de emplear herramientas externas como Dialogflow [12], que requieren la definición manual de intenciones y entidades, se ha optado por desarrollar una interfaz propia con tecnologías abiertas. Esta elección permite un mayor control sobre la arquitectura, una mejor integración con modelos locales y mayor trazabilidad en la recuperación y generación de respuestas.

El motor conversacional desarrollado en este proyecto se fundamenta en una arquitectura modular que combina recuperación de información, reordenamiento contextual y generación de lenguaje natural. Concretamente, se compone de los siguientes elementos:

- Recuperación semántica: se utiliza el modelo de embeddings multilingües multilingual-e5-small (Reimers y Gurevych, 2023) para codificar el corpus documental en vectores densos, los cuales se indexan mediante Chroma (C. Team, 2023), permitiendo búsquedas eficientes por similitud contextual.

- Reordenamiento contextual (reranking): los fragmentos recuperados se reordenan utilizando el modelo CohereRerank (multilingual-v3.0) (Cohere, 2023), que asigna mayor prioridad a los contenidos más relevantes en relación con la consulta del usuario.

- Generación de respuestas: se emplea un modelo de lenguaje local, gemma2:2b (DeepMind, 2024), ejecutado con Ollama, que recibe como entrada el contexto recuperado y genera respuestas precisas, claras y adaptadas al entorno académico.

[12] Documentación de Dialogflow

Este enfoque permite abordar preguntas complejas y específicas sobre normativas, calendario, asignaturas o trámites institucionales, manteniendo una interacción fluida y profesional sin necesidad de depender de servicios externos en la nube.

RAG (Retrieval-Augmented Generation)

El enfoque RAG constituye una evolución significativa en el diseño de asistentes conversacionales, ya que combina dos capacidades clave: la recuperación de información relevante desde una base documental y la generación de texto contextualizado mediante LLMs.

Este paradigma se estructura en tres fases principales:

- Recuperación: ante una consulta del usuario, se localizan los fragmentos más relevantes dentro de un corpus previamente indexado mediante técnicas de búsqueda semántica.
- Enriquecimiento contextual: los fragmentos recuperados se concatenan junto con la pregunta original y se proporcionan como entrada al modelo generativo.
- Generación: el modelo de lenguaje elabora una respuesta natural, coherente y fundamentada exclusivamente en el contexto proporcionado.

Esta arquitectura permite ofrecer respuestas precisas, actualizadas y trazables, basadas en documentación real como normativas académicas, guías docentes o calendarios universitarios. En el presente proyecto, se implementa un sistema RAG completo que automatiza este flujo para proporcionar asistencia académica a estudiantes de la Universidad de Alicante (ver Figura 2.1).

Enfoque actual

2.6 Interfaz conversacional autónoma basada en Gradio y FastAPI

En lugar de integrar el sistema con plataformas de diseño conversacional como Dialogflow, este proyecto opta por una arquitectura autónoma y autoejecutable, diseñada completamente en Python. La interacción con el usuario se gestiona mediante una interfaz desarrollada con

Componente	**Tecnología empleada**	**Justificación principal**
Interfaz conversacional	Gradio	Sí (LLM preentrenado)
Personalización	Limitada a intents y entidades	Ligero, personalizable, permite ejecución local
Backend	FastAPI + LangChain	Modular, escalable, fácil de exponer como microservicio
Recuperación semántica	Chroma + E5-multilingual	Precisión alta en embeddings multilingües
Reordenamiento de documentos	CohereRerank	Mejora la relevancia contextual antes de generar respuesta
Modelo generativo	gemma2:2b	Ligero, ejecutable en GPU local, buenas respuestas

Tabla 2.1: Tabla de ventajas del enfoque actual

Gradio (G. Team, 2023), que se conecta a un backend implementado con FastAPI (Ramírez, 2023) . Esta decisión responde a la necesidad de:

1. Mantener el control total sobre el flujo conversacional.
2. Evitar la dependencia de servicios externos en la nube.
3. Asegurar la ejecución local y replicabilidad del sistema, especialmente útil en entornos académicos con recursos limitados.

El funcionamiento general de la interfaz se basa en un ciclo sencillo:

- Entrada del usuario: el usuario introduce su consulta a través de la interfaz web de Gradio.
- Procesamiento backend: FastAPI recibe la consulta, la procesa mediante el sistema RAG (recuperación + generación), y devuelve una respuesta enriquecida con información contextual.
- Respuesta final: Gradio muestra la respuesta al usuario en un formato conversacional, incluyendo —cuando es posible— las fuentes documentales utilizadas.

Esta implementación no solo simplifica el despliegue, sino que también permite realizar pruebas, evaluaciones e iteraciones de mejora con gran flexibilidad. A futuro, esta arquitectura modular facilitaría la integración en otras plataformas, como aplicaciones móviles o portales institucionales.

3 Objetivos

El presente capítulo expone los objetivos que guían el desarrollo de este Trabajo Fin de Grado. A partir de la problemática detectada en el acceso a la información académica por parte del estudiantado universitario, se plantean tanto el objetivo general del proyecto como un conjunto de objetivos específicos que estructuran y orientan su implementación. Estos objetivos no solo definen el alcance técnico del sistema propuesto, sino que también delimitan las fases metodológicas necesarias para garantizar su funcionalidad, escalabilidad y utilidad dentro del contexto institucional de la Universidad de Alicante.

3.1 Objetivo general

El objetivo general de este proyecto es desarrollar un chatbot inteligente basado en la técnica RAG, con el propósito de facilitar a los estudiantes de la Universidad de Alicante el acceso rápido, preciso y contextualizado a información académica relevante. Esta necesidad responde a la dispersión habitual de documentos institucionales, la complejidad de normativas administrativas y la carga operativa que suponen las consultas repetitivas en los servicios de atención. La solución propuesta busca mejorar significativamente la eficiencia en la consulta de recursos institucionales, tales como normativas, calendarios académicos, planes de estudio, guías docentes y trámites administrativos. Si bien el sistema se ha validado inicialmente con un subconjunto representativo de documentos académicos, se plantea su escalabilidad y adaptabilidad a distintas titulaciones y servicios de la universidad, con el objetivo de convertirse en una herramienta transversal de apoyo académico dentro del entorno universitario.

3.2 Objetivos especificos

Los objetivos específicos de este proyecto definen metas concretas que permiten operacionalizar el objetivo general anteriormente expuesto. A través de estos objetivos se abordan las distintas fases técnicas y metodológicas necesarias para garantizar la funcionalidad, utilidad y aplicabilidad del sistema dentro del entorno universitario.

- **Diseñar la arquitectura del sistema** combinando técnicas de recuperación de información y generación de lenguaje natural siguiendo el enfoque RAG, para asegurar respuestas coherentes y contextuales a las consultas de los usuarios.
- Recolectar, estructurar y preparar un **corpus documental** relevante de la Universidad de Alicante, que incluya guías docentes, normativas académicas, calendarios, procedimientos institucionales y otros documentos útiles para el estudiantado.
- Implementar un sistema de **indexación** y **búsqueda semántica** que permita recuperar los fragmentos de información más relevantes del corpus ante cada consulta del usuario.
- Integrar un modelo de lenguaje **LLM de última generación**, capaz de generar respuestas naturales, comprensibles y ajustadas al contexto académico, a partir de los documentos recuperados.
- Realizar **pruebas de validación** con usuarios reales del grado, con el fin de evaluar la precisión de las respuestas, la utilidad percibida, la experiencia de uso y posibles áreas de mejora del chatbot.
- **Documentar** todo el proceso de desarrollo del proyecto, incluyendo las decisiones técnicas, herramientas utilizadas, dificultades encontradas y propuestas de mejora o ampliación del sistema a otras áreas de la universidad.

4 Metodología

Este capítulo describe el enfoque metodológico adoptado para el desarrollo del asistente conversacional inteligente basado en la técnica RAG. La propuesta se fundamenta en una arquitectura híbrida que combina LLMs con mecanismos de recuperación semántica de información, y se articula a través de distintas fases que abarcan desde la construcción del corpus documental hasta la implementación conversacional mediante Gradio.

4.1 Fase 1: Recolección y preparación del corpus

La preparación del corpus constituye una etapa fundamental en el diseño de sistemas basados en RAG, especialmente cuando se trabaja con información institucional compleja. En este proyecto, el corpus está compuesto por **fragmentos de documentos oficiales de la Universidad de Alicante**, centrados en normativas, plazos administrativos, convocatorias de ayudas, procedimientos académicos y otros aspectos clave del entorno universitario.

Uno de los principales desafíos en esta fase fue la *heterogeneidad formal y semántica* de las fuentes. Los documentos incluían desde resoluciones normativas extensas hasta tablas de fechas, anexos explicativos o respuestas breves extraídas de archivos CSV de preguntas frecuentes. Esta variedad requería un enfoque de preprocesamiento cuidadoso y flexible:

- **Extracción textual**: para los documentos PDF se utilizó la biblioteca **PyMuPDF**, capaz de preservar la estructura jerárquica y el flujo de texto (Software, 2023). En el caso de los archivos CSV, cada fila se trató como un par pregunta–respuesta explícito.
- **Normalización y limpieza**: se eliminaron cabeceras redundantes, artefactos de codificación y se unificó el formato de los textos a UTF-8. Esto garantizó consistencia en el corpus y evitó errores durante la vectorización.

- **Segmentación semántica**: se aplicó el algoritmo **RecursiveCharacterTextSplitter** proporcionado por la biblioteca LangChain (Developers, 2023), con un *chunk_size* de 800 y un *chunk_overlap* de 100. Esta configuración permitió conservar el contexto entre fragmentos y mantener la coherencia semántica.

- **Anotación contextual**: cada fragmento fue enriquecido con metadatos como el tema (extraído del subdirectorio de origen) y el nombre del archivo, lo que mejora la trazabilidad de la información y permite justificar las fuentes en las respuestas generadas.

Gracias a este procesamiento, el corpus resultante presenta una estructura semántica robusta y organizada, adecuada para ser indexada en un sistema vectorial con **ChromaDB** (C. Team, 2023) y utilizada como base documental en la generación de respuestas académicas contextualizadas.

4.2 Fase 2: Vectorización e indexación semántica

Una vez estructurado el corpus, se procede a su vectorización utilizando modelos de *sentence embeddings* proporcionados por la plataforma **Hugging Face** (Reimers y Gurevych, 2023). Para ello, se emplean modelos preentrenados como **all-MiniLM-L6-v2** del repositorio **sentence-transformers**, que permiten codificar cada fragmento textual en un vector denso en un espacio de alta dimensión.

Estos vectores se almacenan en un índice semántico basado en **faiss** o **chromadb** (C. Team, 2023), permitiendo realizar consultas eficientes por similitud de contexto.

4.3 Fase 3: Implementación del sistema RAG

El núcleo del sistema se basa en el paradigma *Retrieval-Augmented Generation*. Para cada entrada del usuario, se recuperan los k fragmentos más relevantes del índice vectorial, los cuales son concatenados junto con la consulta original y enviados al modelo generativo.

Este proceso se implementa en Python utilizando la biblioteca **uvicorn** (Developers, 2023) para desplegar una API REST, y modelos de lenguaje (LLMs) como **Gemma**, **Llama** o **Deepseek**, a través de endpoints locales o APIs externas.

4.4 Fase 4: Integración conversacional con interfaz propia basada en Gradio y FastAPI

Para proporcionar una experiencia conversacional fluida y accesible, se ha desarrollado una interfaz gráfica propia utilizando la biblioteca Gradio (G. Team, 2023), que permite interactuar en tiempo real con el sistema RAG desde un navegador web. Esta interfaz se comunica directamente con un backend construido con FastAPI (Ramírez, 2023), el cual se encarga de orquestar el flujo completo de procesamiento: desde la recuperación de contexto relevante hasta la generación de la respuesta final.

Esta solución sustituye a plataformas externas, ofreciendo las siguientes ventajas:

- Independencia tecnológica: permite ejecutar todo el sistema localmente, sin depender de servicios de terceros.
- Control total sobre la lógica de diálogo, la presentación de resultados y la gestión de errores.
- Modularidad: el backend puede integrarse fácilmente en otras aplicaciones o plataformas si se desea una futura expansión.

El flujo completo de interacción es el siguiente:

- El usuario introduce su consulta mediante el interfaz Gradio.
- La consulta se envía al backend FastAPI, que activa el proceso RAG: recuperación de documentos (mediante embeddings vectoriales y reordenamiento), y generación de respuesta con el modelo gemma2:2b (DeepMind, 2024).
- La respuesta generada se devuelve a la interfaz junto con las fuentes utilizadas, si las hay, y se presenta de forma estructurada.

Este diseño permite validar de forma ágil el comportamiento del asistente en sesiones reales, así como registrar logs de interacción útiles para el análisis posterior del sistema.

4.5 Fase 5: Evaluación y validación

Finalmente, se llevarán a cabo la evaluación del sistema conversacional desarrollado, tanto desde un punto de vista cuantitativo como, en futuras fases, desde una perspectiva cualitativa basada en la experiencia del usuario.

En primer lugar, se plantea una evaluación técnica basada en métricas objetivas del procesamiento del lenguaje natural, como BERTScore-F1, ROUGE-L, Exact Match y tiempo medio de inferencia. Estas métricas permiten comparar la calidad semántica y sintáctica de las respuestas generadas por distintos modelos de lenguaje, así como su eficiencia computacional.

Para llevar a cabo esta evaluación se prevé construir un banco de preguntas representativas del ámbito académico universitario, junto con sus respectivas respuestas de referencia. Cada modelo sería evaluado ejecutándose en condiciones controladas (por ejemplo, con una GPU local de 4GB), a fin de determinar cuál ofrece el mejor equilibrio entre rendimiento y coste computacional.

Por otro lado, también se contempla una futura fase de validación cualitativa con usuarios reales de la comunidad universitaria, mediante la realización de pruebas de uso controladas y encuestas de satisfacción. Estas pruebas permitirían analizar la utilidad percibida, la claridad de las respuestas, la cobertura temática y la experiencia global de interacción con el sistema, aportando una visión complementaria a la evaluación técnica.

4.6 Herramientas y tecnologías utilizadas

El desarrollo del asistente conversacional propuesto se ha apoyado en una serie de herramientas y bibliotecas modernas que permiten implementar de forma modular una arquitectura basada en RAG. La selección tecnológica se ha guiado por varios criterios: compatibilidad con modelos ejecutables en local, disponibilidad de documentación, soporte para el procesamiento del lenguaje natural en español, facilidad de integración entre componentes, rendimiento computacional y versatilidad para experimentación y evaluación.

Asimismo, se ha priorizado el uso de herramientas de código abierto ampliamente adoptadas en entornos académicos y profesionales, con el objetivo de facilitar la replicabilidad del sistema

en futuras investigaciones o adaptaciones institucionales. A continuación, se describen las tecnologías empleadas, incluyendo sus versiones concretas para asegurar la reproducibilidad del entorno de desarrollo.

- **Python** (v3.10): lenguaje principal de desarrollo del backend y lógica de procesamiento de lenguaje natural.
- **LangChain** (v0.3.26): biblioteca modular para construir pipelines de recuperación y generación en arquitecturas RAG.
- **Hugging Face Transformers** (implícito vía `sentence-transformers v4.1.0`) y **LangChain-HuggingFace** (v0.3.0): utilizados para cargar modelos de embeddings y LLMs.
- **ChromaDB** (v1.0.13): motor de almacenamiento e indexación vectorial local, gestionado mediante `langchain-chroma v0.2.4`.
- **Ollama** (entorno externo, usado junto con `langchain-ollama v0.3.3`): ejecución local de modelos de lenguaje ligeros como `gemma2:2b`.
- **FastAPI** (v0.115.13): framework asincrónico para desarrollar el servidor web y exponer la API REST.
- **Gradio** (v5.34.2): interfaz gráfica conversacional para la interacción con el usuario final desde el navegador.
- **Git** y **GitHub**: herramientas de control de versiones utilizadas para gestionar el código fuente y registrar los avances.
- **PyMuPDF** (v1.26.1): biblioteca para la extracción de texto desde documentos PDF manteniendo la estructura del documento.
- **pandas** (v2.3.0): manipulación y análisis de datos, especialmente en el preprocesamiento de preguntas y respuestas.
- **evaluate** (v0.4.4), **bert-score** (v0.3.13) y **rouge-score** (v0.1.2): librerías para calcular métricas de evaluación lingüística de las respuestas generadas.

Estas herramientas han sido seleccionadas por su rendimiento, compatibilidad con modelos locales y la posibilidad de ejecutar todo el sistema en entornos con recursos limitados.

5 Desarrollo

Este capítulo detalla la implementación técnica del asistente conversacional desarrollado, desde la construcción del índice semántico hasta la generación de respuestas mediante un modelo de lenguaje local. La aplicación se ha estructurado de forma modular utilizando *Python*, combinando bibliotecas especializadas como **LangChain** (Developers, 2023), **Gradio** (G. Team, 2023) y **FastAPI** (Ramírez, 2023).

5.1 Construcción del índice semántico

El objetivo de esta fase es transformar los documentos fuente en representaciones vectoriales densas, adecuadas para realizar búsquedas semánticas eficientes dentro del sistema RAG. Este proceso consta de varias etapas: recolección de documentos, segmentación en fragmentos, generación de vectores mediante un modelo de embeddings y almacenamiento en una base de datos vectorial (ver Figura 5.1).

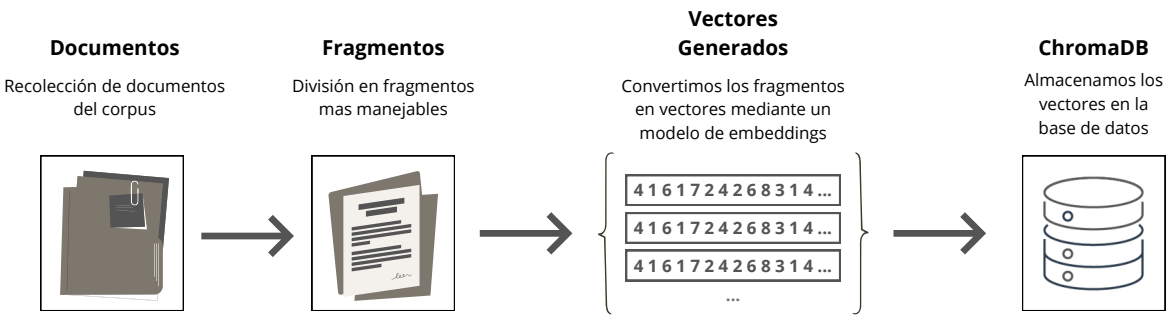


Figura 5.1: Diagrama del proceso de construcción del índice semántico

5.1.1 Recolección y procesamiento de documentos

Los documentos se obtienen del directorio de almacenamiento, y pueden ser de tipo PDF o CSV.

Procesamiento de archivos CSV

Cada fila de un archivo CSV contiene una pregunta y su respuesta correspondiente. Estas se convierten en objetos **Document**, manteniendo la estructura de pregunta–respuesta como contenido y agregando metadatos como la fuente y el tema.

Procesamiento de archivos PDF

Para los documentos PDF, se utiliza el cargador **PyMuPDFLoader**, que extrae el contenido textual. Los textos se enriquecen con metadatos adicionales antes de ser divididos.

5.1.2 Segmentación semántica del texto

Antes de convertir el contenido en vectores, es necesario dividirlo en fragmentos manejables y coherentes desde el punto de vista semántico. Para ello, se emplea la clase **RecursiveCharacterTextSplitter** [1] de LangChain, que permite trocear textos largos con solapamiento configurable. Esto ayuda a preservar el contexto entre fragmentos consecutivos, mejorando la coherencia de las respuestas generadas.

En este proyecto se ha optado por un tamaño de fragmento de 800 caracteres y un solapamiento de 100 caracteres.

5.1.3 Generación de vectores y almacenamiento

Los fragmentos generados se convierten en vectores mediante el uso del modelo multilingüe **intfloat/multilingual-e5-small** [2] (Reimers y Gurevych, 2023), seleccionado por su compatibilidad con los idiomas español y valenciano y su eficiencia en recuperación semántica.

Los vectores resultantes se almacenan en un índice persistente mediante la base de datos local **ChromaDB** (C. Team, 2023). Para evitar reprocesamientos, se mantiene un registro

[1] Función de segmentación del texto
[2] Modelo de embeddings

en `processed_docs.json`.

Esta estructura semántica robusta sienta las bases del sistema RAG, mejorando la relevancia y coherencia de las respuestas generadas.

5.2 Recuperación y reordenamiento contextual

Para cada consulta, se emplea un *retriever* que localiza los documentos más relevantes por similitud semántica mediante **VectorStoreRetriever** [3] con búsqueda tipo MMR (*Maximal Marginal Relevance*), que equilibra diversidad y relevancia. Posteriormente, se aplica un modelo de *reordenamiento contextual* con **CohereRerank** [4] (Cohere, 2023), que asigna prioridad a los fragmentos más significativos (ver Figura 5.2).

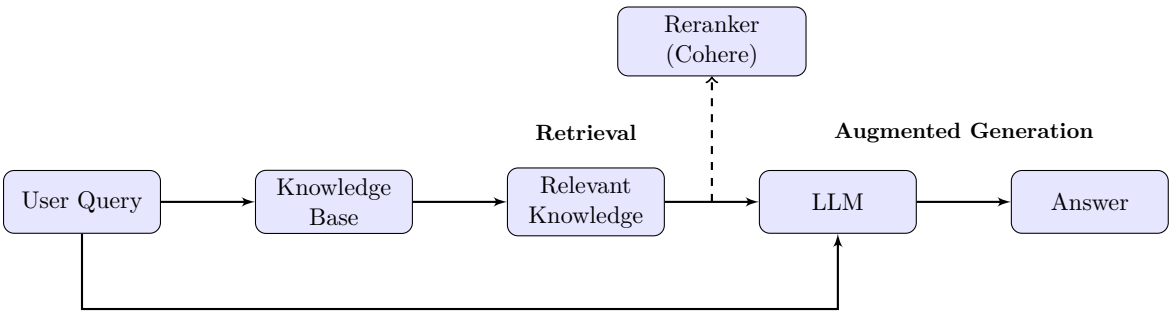


Figura 5.2: Diagrama del proceso Retrieval-Augmented Generation (RAG) con Reranker.

Este doble filtrado garantiza que los documentos proporcionados al modelo generativo no solo sean relevantes, sino también jerárquicamente óptimos para la generación textual. El **reranker** se entrena específicamente para tareas multilingües, lo que favorece el rendimiento en un entorno como el universitario, donde coexisten textos en castellano, valenciano e inglés.

En comparación con una recuperación simple por similitud, este enfoque ha mostrado una mejora en la coherencia temática de las respuestas y una reducción del *hallucination rate* (respuestas inventadas) del modelo generador.

[3] Definición de Retrieval
[4] Definición del Reranker

5.3 Generación de respuestas con modelo local

Para la generación final, se emplea el modelo **gemma2:2b** ejecutado localmente con **Ollama** (O. Team, 2024). Este modelo de lenguaje ligero permite realizar inferencias en tiempo razonable dentro de un entorno con recursos computacionales limitados.

5.3.1 Inicialización del modelo generativo.

El modelo **gemma2:2b** [5] se ejecuta localmente a través de **Ollama**, utilizando la clase **OllamaLLM** de LangChain. Esta configuración permite integrar modelos ligeros sin necesidad de conexión externa a servicios en la nube.

5.3.2 Definición del prompt y construcción del pipeline.

Se construye un prompt específico mediante **ChatPromptTemplate** de LangChain, que incluye tanto la consulta como el contexto recuperado.

Código 5.1: Definición del prompt y construcción del pipeline

```
prompt = ChatPromptTemplate.from_template("""
Eres un asistente académico especializado en la Universidad de Alicante.
Tu tarea es responder preguntas con información relevante y fiable extraída exclusivamente del contexto proporcionado.

{question}

 Contexto recuperado:
{context}
""")

chat_chain = RunnableSequence(prompt | model | StrOutputParser())
```

Este prompt ha sido cuidadosamente diseñado para fomentar respuestas informativas, profesionales y precisas, evitando que el modelo genere información no contenida en el contexto.

[5] Documentación del modelo **Gemma2:2b**

Se especifica que el asistente debe declarar expresamente cuando no disponga de información suficiente, lo cual es esencial para la fiabilidad del sistema en entornos académicos.

5.3.3 Ejecución del modelo con entrada dinámica.

El pipeline de generación se invoca con dos entradas: la pregunta del usuario y el contexto documental previamente recuperado y reordenado. El resultado es una cadena de texto generada por el modelo, interpretada como la respuesta final.

Código 5.2: Generación de respuesta basada en contexto

```
response = chat_chain.invoke({
    "question": user_question,
    "context": context_text
})
```

La elección de **gemma2:2b** se justifica por su equilibrio entre rendimiento y eficiencia computacional: ejecuta localmente en GPU de 4 GB, responde con claridad en español y tiene tiempos de inferencia razonables (~10 segundos por consulta en pruebas con GTX 1050).

5.4 Exposición mediante API REST

Una vez definido el pipeline de generación, se expone un endpoint REST a través de **FastAPI**. Esta API permite recibir peticiones externas desde otras aplicaciones o interfaces web, facilitando la integración del asistente conversacional con distintos entornos.

Se utiliza **pydantic.BaseModel** para definir la estructura de la petición entrante. El parámetro obligatorio es la pregunta del usuario, y opcionalmente se puede incluir un **session_id**.

5.4.1 Definición del endpoint.

El endpoint **/chat** acepta peticiones POST y ejecuta el pipeline RAG utilizando el texto de la pregunta y el contexto recuperado (ver Figura 5.3).

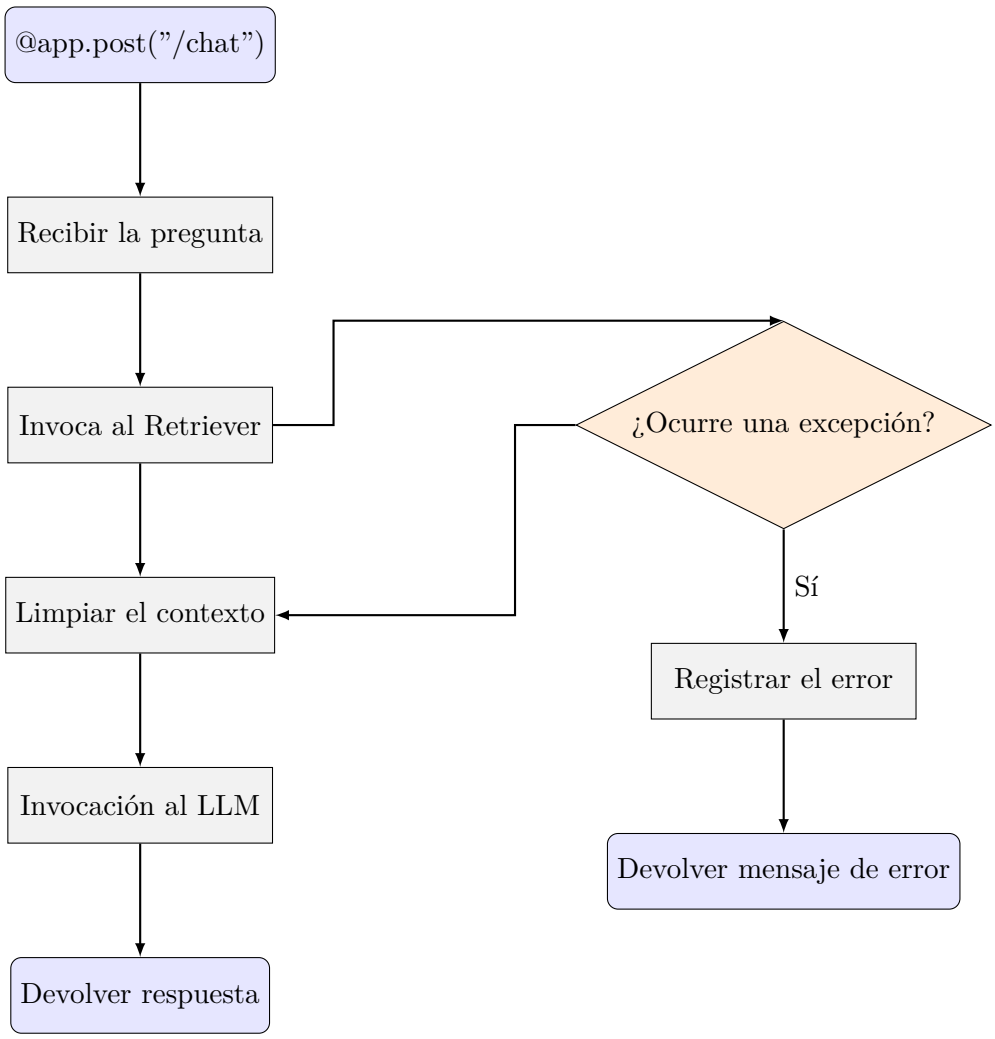


Figura 5.3: Diagrama de flujo del endpoint `/chat`

Este endpoint se puede probar directamente con herramientas como **curl**, **Postman**, o desde una interfaz gráfica como la que se verá en la siguiente sección. Su diseño permite integrar el backend del asistente con otras capas como frontends web, aplicaciones móviles o asistentes por voz.

5.5 Interfaz conversacional con Gradio

Con el objetivo de ofrecer una experiencia de uso accesible, multiplataforma y sin necesidad de conocimientos técnicos por parte del usuario final, se ha desarrollado una interfaz gráfica basada en la biblioteca *Gradio*. Esta herramienta permite crear y desplegar aplicaciones web

interactivas de forma sencilla y eficiente, integrándose perfectamente con modelos de lenguaje y sistemas RAG.

La interfaz ha sido diseñada para simular una conversación natural, permitiendo al usuario introducir preguntas en lenguaje natural y visualizar las respuestas generadas por el sistema, junto con las fuentes documentales utilizadas. Además, se ha incluido un historial de mensajes y funcionalidades como copiar respuesta o actualizar la base documental.

Esta solución resulta especialmente útil durante la fase de pruebas y demostración, al facilitar la validación del comportamiento del sistema sin depender de plataformas externas. Gracias a su naturaleza ligera y modular, Gradio ha permitido iterar rápidamente sobre la interfaz y mantener una integración directa con el núcleo del sistema implementado.

5.5.1 Lanzamiento de la interfaz.

La interfaz se configura y despliega mediante `gr.ChatInterface` [6], especificando el título, el aspecto visual y la función principal que gestiona la entrada del usuario.

Gracias a esta interfaz, el sistema puede utilizarse directamente desde un navegador, sin necesidad de programación por parte del usuario final. Además, su arquitectura modular permite que esta misma lógica pueda integrarse fácilmente en otras plataformas institucionales o educativas.

5.5.2 Despliegue en Hugging Face Spaces

Con el objetivo de facilitar la replicación y permitir pruebas públicas del asistente conversacional, se ha publicado una instancia funcional del sistema en la plataforma Hugging Face Spaces [7]. Este entorno permite ejecutar la interfaz Gradio directamente desde un navegador, sin necesidad de configurar un entorno local, facilitando así la validación por parte de usuarios externos y futuros investigadores interesados en extender o adaptar el proyecto.

Este despliegue complementa el sistema local desarrollado con Gradio y FastAPI, ofreciendo una alternativa ligera para demostraciones y pruebas en tiempo real.

[6] Configuración de Gradio
[7] Demo pública en la plataforma Hugging Face Spaces

5.6 Evaluación automática del sistema

Para validar el funcionamiento del asistente conversacional desarrollado, se ha implementado una evaluación automática basada en un conjunto de preguntas reales extraídas del corpus institucional de la Universidad de Alicante. Esta evaluación permite comparar de forma objetiva las respuestas generadas por el sistema con las respuestas esperadas o de referencia.

En primer lugar, se carga el conjunto de preguntas desde un archivo CSV. Para cada una de ellas, se recupera el contexto mediante el componente `reranked_retriever`, y se genera una respuesta utilizando el pipeline `chat_chain`. Las respuestas generadas se almacenan en un nuevo archivo junto con las respuestas reales correspondientes.

Una vez completado este proceso, se calcula la calidad de las respuestas generadas mediante métricas estándar del procesamiento de lenguaje natural. En concreto, se emplean las métricas **BERTScore-F1**, **ROUGE-L** y **Exact Match**, todas ellas implementadas mediante la biblioteca `evaluate` de Hugging Face (ver Figura 5.4).

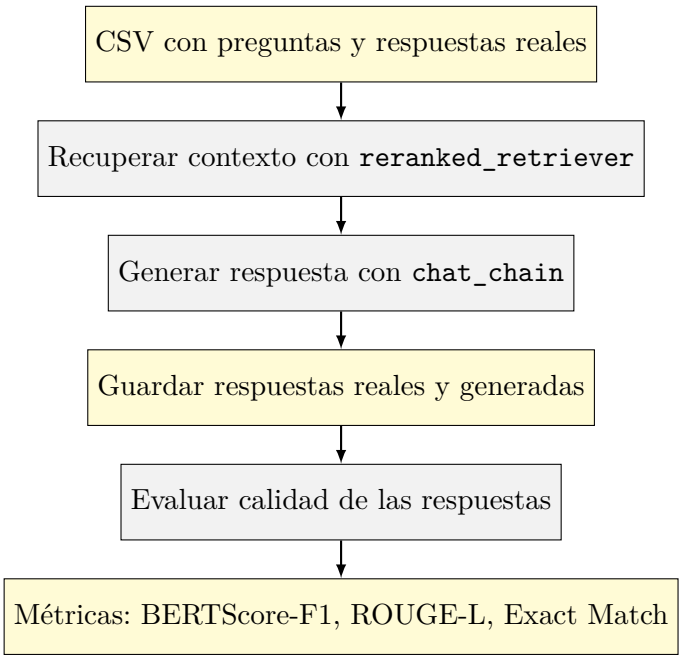


Figura 5.4: Proceso de evaluación automática del sistema conversacional

5.6.1 Interpretación de los resultados.

La métrica **BERTScore-F1** permite evaluar la similitud semántica entre las respuestas generadas y las de referencia. Por su parte, **ROUGE-L** mide el solapamiento léxico, mientras que **Exact Match** evalúa coincidencias literales. Esta última suele arrojar valores bajos en sistemas generativos como RAG, debido a la reformulación libre del lenguaje por parte del modelo.

Este enfoque de evaluación permite comparar modelos con criterios objetivos y replicables, facilitando la toma de decisiones sobre cuál ofrece un mejor equilibrio entre precisión, claridad y eficiencia para su uso en entornos reales.

5.6.2 Resumen del desarrollo

A lo largo de este capítulo se ha descrito con detalle la arquitectura técnica y la implementación práctica del asistente conversacional basado en RAG. El sistema se ha construido de forma modular, empleando herramientas modernas y de código abierto, lo que facilita su mantenimiento, ampliación y reutilización en distintos entornos.

En primer lugar, se ha procesado un corpus institucional formado por documentos académicos en formato PDF y CSV. Tras su limpieza, segmentación semántica y vectorización, los fragmentos resultantes se han almacenado en una base vectorial ChromaDB, optimizada para búsquedas por similitud contextual.

Posteriormente, se ha definido un mecanismo de recuperación híbrido: una búsqueda inicial por embeddings mediante MMR, seguida de un reranking con Cohere para seleccionar los fragmentos más relevantes antes de generar una respuesta. La respuesta final se obtiene con un modelo local de lenguaje natural (Gemma 2B) ejecutado mediante Ollama, a partir de un prompt cuidadosamente diseñado para limitar la generación a la información presente en el contexto.

Para facilitar el acceso al sistema, se han desarrollado dos interfaces: una API REST mediante FastAPI y una interfaz gráfica conversacional con Gradio. Ambas permiten realizar consultas de manera directa, presentando respuestas claras y trazables.

Finalmente, se ha implementado una evaluación automática utilizando métricas estándar (BERTScore, ROUGE-L y Exact Match), lo que permite medir con precisión la calidad

semántica y sintáctica de las respuestas generadas.

Este desarrollo demuestra la viabilidad de integrar modelos ligeros de lenguaje con técnicas modernas de recuperación documental para construir asistentes inteligentes funcionales, ejecutables en local y adaptados al contexto académico universitario.

6 Resultados

En este capítulo se presentan los resultados obtenidos tras evaluar distintos modelos de lenguaje utilizados en la arquitectura RAG implementada. La evaluación tiene como objetivo determinar cuál de estos modelos ofrece un mejor equilibrio entre calidad de respuesta, coherencia semántica y viabilidad computacional para su ejecución en entornos locales con recursos limitados.

6.1 Metodología de evaluación

Para evaluar el comportamiento de los modelos se emplearon las siguientes métricas:

- **BERTScore-F1**: mide la similitud semántica entre la respuesta generada y la respuesta de referencia utilizando embeddings de un modelo BERT preentrenado (Zhang y cols., 2019). Se considera una métrica robusta para tareas de lenguaje natural generativo.

- **ROUGE-L**: evalúa el solapamiento de subsecuencias comunes más largas entre la generación y la referencia (Lin, 2004). Es útil para medir la calidad sintáctica de las respuestas.

- **Exact Match (EM)**: indica si la respuesta generada coincide literalmente con la referencia tras normalizar puntuación y capitalización. Aunque estricta, se incluye para evidenciar las diferencias entre QA cerrado y sistemas generativos RAG (Rajpurkar y cols., 2016).

- **Tiempo medio de inferencia**: tiempo medio (en segundos) que tarda el modelo en generar una respuesta, excluyendo la fase de recuperación de documentos.

Todas las respuestas fueron evaluadas utilizando un conjunto de 20 preguntas reales, representativas de las consultas académicas más frecuentes, con sus correspondientes respuestas de referencia. Se empleó un entorno controlado con una GPU NVIDIA GTX 1050 de 4GB.

6.2 Resultados obtenidos

La Tabla 6.1 muestra los valores obtenidos por cada modelo evaluado en las métricas descritas. Los resultados reflejan tanto la calidad de las respuestas como la eficiencia computacional de cada modelo.

En términos de precisión semántica, el modelo `mistral:7b` obtiene el mejor BERTScore-F1, aunque a costa de un mayor tiempo de inferencia. Por su parte, `gemma2:2b` ofrece un equilibrio óptimo entre rendimiento y velocidad, siendo capaz de ejecutarse en entornos locales con GPU de 4GB.

Es importante destacar que el valor de `Exact Match` es 0% en todos los modelos, lo cual es un comportamiento esperado en sistemas generativos como RAG. Este tipo de modelos reformulan las respuestas en lenguaje natural, priorizando la coherencia y la utilidad sobre la literalidad exacta.

En base a estos resultados, se selecciona `gemma2:2b` como modelo principal del sistema debido a su viabilidad técnica y su comportamiento robusto en tareas de respuesta académica contextualizada.

Modelo	BERTScore-F1	ROUGE-L	EM	Tiempo (s)	Observaciones
gemma2:2b	72.31%	28.73%	0.00%	10.2	Preciso y rápido
llama3.2:1b	72.49%	27.84%	0.00%	11.8	Alta calidad y buena coherencia
mistral:7b	76.37%	37.36%	0.00%	39.7	Muy alta calidad y cobertura amplia
deepseek-r1:1.5b	68.55%	22.00%	0.00%	31.1	Baja calidad, claridad insuficiente
phi3:mini	69.80%	18.85%	0.00%	49.8	Respuestas evasivas y confusas

Tabla 6.1: Comparativa de modelos evaluados en el sistema RAG

6.3 Gráficas comparativas

Con el objetivo de visualizar de manera clara y directa el rendimiento del sistema conversacional, se han generado una serie de gráficas que ilustran los resultados obtenidos en las distintas métricas de evaluación. Estas representaciones permiten detectar patrones, identificar posibles desviaciones entre las respuestas generadas y las de referencia, y facilitar una interpretación comparativa más intuitiva que la tabla numérica aislada.

6.3.1 Comparación de BERTScore-F1

La Figura 6.1 muestra una comparación del rendimiento de distintos modelos de lenguaje según la métrica BERTScore-F1. El modelo `mistral-7b` destaca con un 76.37%, siendo el que mayor similitud semántica presenta respecto a las respuestas de referencia. Le siguen `llama3:2.1b` y `gemma:2.2b` con puntuaciones muy similares, mientras que `deepseek-r1.1b` y `phi3:mini` presentan un rendimiento inferior. Esta comparación permite seleccionar el modelo más adecuado en función de la fidelidad semántica que se desea preservar en las respuestas generadas.

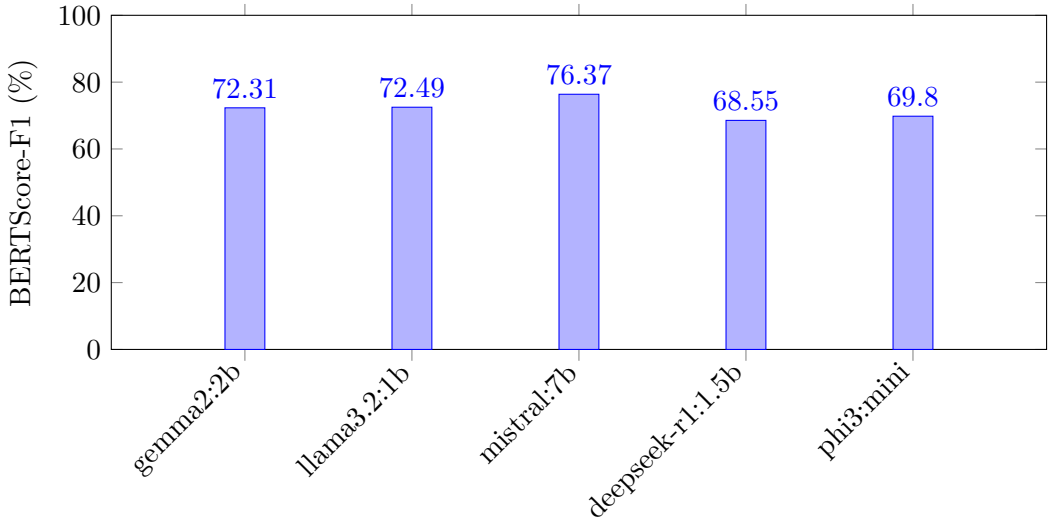


Figura 6.1: Comparación de modelos según BERTScore-F1

6.3.2 Comparación de ROUGE-L

La métrica ROUGE-L evalúa el solapamiento literal entre las respuestas generadas por cada modelo y las respuestas de referencia. Tal como muestra la Figura 6.2, el modelo `mistral-7b` destaca con un 37.36%, seguido por `gemma:2.2b` y `llama3:2.1b`, ambos cercanos al 28%. Los modelos `deepseek-r1.1b` y `phi3:mini` presentan valores considerablemente más bajos, lo que indica menor fidelidad léxica. Este resultado evidencia las diferencias entre modelos en su capacidad para reproducir estructuras textuales similares a las del corpus original.

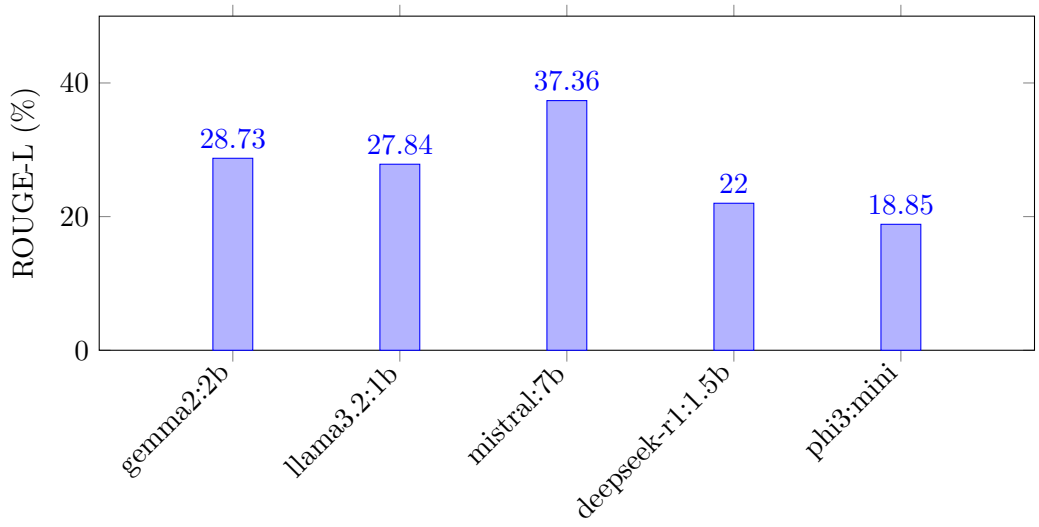


Figura 6.2: Comparación de modelos según ROUGE-L

6.3.3 Comparación del tiempo medio de inferencia

La Figura 6.3 muestra el tiempo medio necesario para que cada modelo genere una respuesta. Los modelos `gemma:2.2b` y `llama3:2.1b` destacan por su eficiencia, con tiempos inferiores a 12 segundos por inferencia. En contraste, `phi3:mini` es el modelo más lento, alcanzando casi 50 segundos, a pesar de ser uno de los modelos más ligeros. El modelo `mistral-7b`, aunque es el más preciso en términos semánticos y léxicos, presenta un tiempo de inferencia elevado (39.7 s), lo que puede limitar su uso en entornos interactivos. Estos resultados reflejan el clásico compromiso entre precisión y eficiencia, fundamental al seleccionar un modelo para producción.

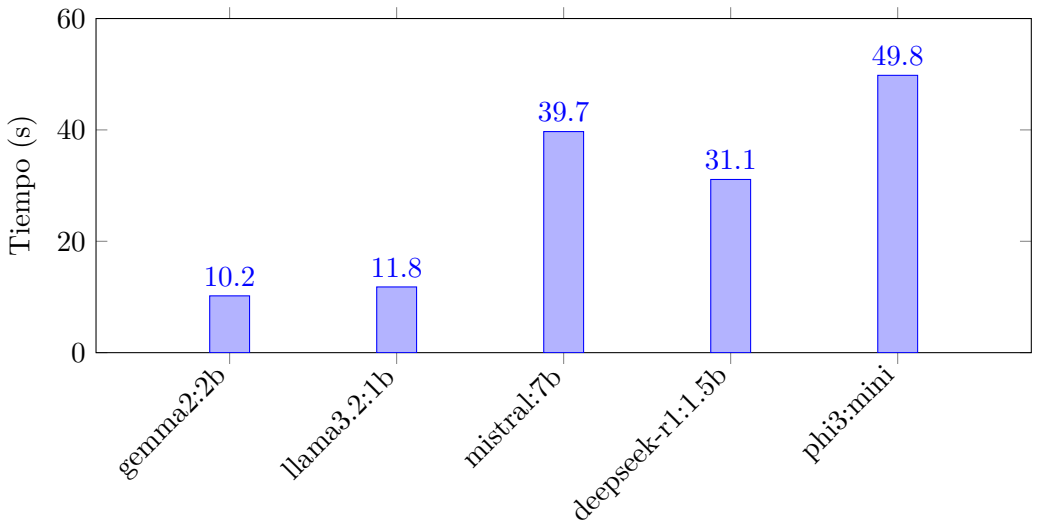


Figura 6.3: Tiempo medio de inferencia por modelo

6.4 Discusión sobre la métrica Exact Match

La métrica *Exact Match* ha arrojado un valor del 0% en todos los modelos. Este resultado es esperado en sistemas RAG, ya que las respuestas generadas por el modelo no se limitan a replicar textualmente un fragmento del corpus, sino que combinan información y reescriben contenido de forma natural. Esto evidencia la necesidad de utilizar métricas semánticas como BERTScore para una evaluación más realista.

A modo ilustrativo, la siguiente comparación muestra una referencia y una respuesta generada que, aunque semánticamente equivalentes, no son consideradas iguales por EM:

> **Referencia:** "En el Servicio de Información Universitaria (SIU), en la página web institucional y en los tablones informativos."
> **Generada:** "Puedes consultar las actividades en la web de la UA y en el SIU."

6.5 Código de evaluación

La evaluación se ha realizado utilizando las siguientes herramientas de Hugging Face 'evaluate', combinadas con pandas para la carga del corpus. A continuación se muestra un fragmento representativo:

Código 6.1: Código para evaluar BERTScore, ROUGE-L y Exact Match

```
import evaluate

bertscore = evaluate.load("bertscore")
rouge = evaluate.load("rouge")
exact_match = evaluate.load("exact_match")

bertscore_result = bertscore.compute(predictions=preds, references=refs, lang=↩
    ↪ "es")
rouge_result = rouge.compute(predictions=preds, references=refs)
em_result = exact_match.compute(predictions=preds, references=refs)
```

6.6 Ejemplos representativos de interacción

Para ilustrar el funcionamiento práctico del sistema desarrollado, a continuación se presentan varios ejemplos reales de interacción con la interfaz conversacional. Cada ejemplo incluye una pregunta formulada en lenguaje natural, la respuesta generada por el asistente y el fragmento del documento original que ha servido como base para construir dicha respuesta.

Estos ejemplos permiten observar el comportamiento del sistema en situaciones reales, así como evaluar su capacidad para recuperar información relevante, sintetizar el contenido y mantener la coherencia semántica con los documentos institucionales.

Además, se han seleccionado casos que cubren distintos tipos de preguntas frecuentes entre estudiantes de la Universidad de Alicante, lo que ayuda a evidenciar la aplicabilidad del sistema en entornos educativos reales.

Ejemplo 1: Derechos lingüísticos

Pregunta del usuario: ¿Qué derechos lingüísticos tienen los estudiantes?

Respuesta generada por el asistente: Los estudiantes tienen los siguientes derechos lingüísticos:

- Comunicación en cualquier lengua oficial: El alumnado puede expresarse en cualquiera de las lenguas oficiales.

- Relaciones con la universidad: Pueden relacionarse en cualquier lengua oficial con todos los órganos de la Universidad de Alicante.

- Aprender y evaluar en cualquier idioma: Se pueden estudiar y evaluar de forma que se realice en cualquier lengua oficial dentro del marco de la programación docente.

- Publicación y defensa de actividades: Pueden publicar y defender sus actividades docentes e investigadoras en cualquiera de las lenguas oficiales.

- Expresión en lengua extranjera (si la docencia es en esa lengua): En caso de que la docencia sea en una lengua extranjera, el estudiante tiene derecho a expresarse en esa lengua.

Figura 6.4: Interacción real con la interfaz: pregunta sobre derechos lingüísticos

Ejemplo 2: Convocatoria SICUE

Pregunta del usuario: ¿Cual es el plazo de solicitud de SICUE?

Respuesta generada por el asistente: El plazo de solicitud para la SICUE es del **13 al 28 de febrero de 2025**, para el alumnado que solicita cualquier periodo de estudios (primer semestre, segundo semestre o anual).

Figura 6.5: Interacción real con la interfaz: pregunta sobre la convocatoria SICUE

Ejemplo 3: Elaboración de guias docentes

Pregunta del usuario: ¿Qué procedimiento se sigue para aprobar y modificar una guía docente?

Respuesta generada por el asistente: Para aprobar o modificar una guía docente, se debe seguir el siguiente procedimiento:

1. Aprobarla: Se deberá cumplir con los pasos descritos en el artículo 7 de la normativa, incluyendo la aprobación por parte del órgano responsable y la Junta de Centro.

2. Modificación durante el periodo habilitado: Si se desea modificar la guía docente, se

deben realizar las modificaciones correspondientes en el periodo habilitado, luego debe ser aprobada por el Consejo de Departamento y ratificada por la Junta de Centro.

3. Información para la matriculación: La guía docente se actualizará al inicio del nuevo curso con los datos del anterior, y se comunicará a los profesores responsables.

4. Modificación en titulaciones propias: En las titulaciones propias, el Vicerrectorado con competencias en Calidad dará instrucciones para la apertura de las guías docentes, después de lo cual el profesorado responsable cumplimentarán las mismas. La guía deberá ser aprobada por el órgano responsable antes del inicio de la matriculación.

Figura 6.6: Interacción real con la interfaz: pregunta sobre guias docentes

Ejemplo 4: Preinscripción de matricula

Pregunta del usuario: ¿Cómo se consultan los resultados de preinscripción?

Respuesta generada por el asistente: Los resultados de preinscripción se publican en el calendario de publicación del listado de admisión. Para consultar los resultados, debes acceder

al website de preinscripción de la Comunidad Valenciana. Una vez conocidos los resultados, debes seguir las instrucciones de matriculación que establece la universidad de admisión.

Para obtener la clave de acceso a UACloud (campus virtual UA) y completar el programa de matrícula, deberás acceder al siguiente formulario web.

Figura 6.7: Interacción real con la interfaz: pregunta sobre la preinscripción universitaria

Ejemplo 4: Adaptación curricular

Pregunta del usuario: Actualmente estoy trabajando por las mañanas y no puedo asistir a una asignatura que se imparte en ese horario. ¿Sería posible cambiar mi horario a las tardes?".

Respuesta generada por el asistente: Para solicitar un cambio de horario a las tardes, debes seguir estos pasos:

1. Contáctate con tu coordinador de la UA: Es necesario hacer modificaciones en el Bloque destino de tu UACloud.

2. Documenta tu situación laboral: Deberás presentar un informe que acredite tu jornada

laboral incompatible con el horario de las asignaturas. El informe debe contener la información de tu trabajo y un contrato de trabajo con horario de trabajo o certificado de empresa con el horario laboral.

3. Modifica tu plan curricular: Si no se puede cambiar el grupo de estudio, debes realizar las modificaciones en UACloud –>Programas de Movilidad–>Historial–>Learning agreement.

Recuerda que los plazos para modificar el plan curricular son determinados por tu centro y la UA.

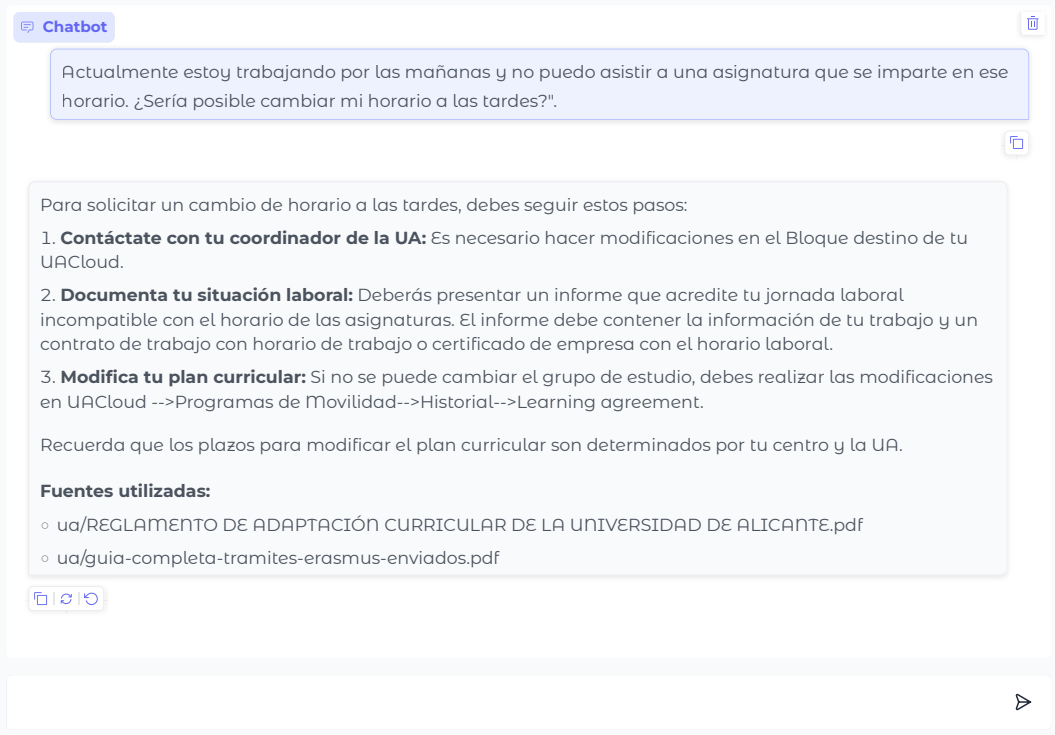


Figura 6.8: Interacción real con la interfaz: pregunta sobre adaptaciones curriculares

6.7 Conclusiones del análisis

Los resultados del análisis muestran un equilibrio claro entre precisión semántica y eficiencia computacional en los modelos evaluados. El modelo `mistral:7b` destacó como el más competente en términos de calidad de respuesta, alcanzando los valores más altos tanto en

BERTScore-F1 como en ROUGE-L. Sin embargo, esta precisión viene acompañada de un tiempo de inferencia considerablemente mayor, lo que limita su idoneidad para aplicaciones interactivas en tiempo real.

En cambio, el modelo `gemma:2.2b` ofrece una alternativa muy competitiva. Aunque sus métricas de calidad son ligeramente inferiores a las del modelo líder, mantiene una coherencia semántica aceptable con tiempos de respuesta significativamente más bajos. Este equilibrio lo posiciona como una opción óptima para sistemas que deban ejecutarse en entornos con recursos computacionales limitados o con requisitos de baja latencia.

Por otro lado, modelos como `phi3:mini` o `deepseek-r1.1b`, si bien presentan ciertas ventajas en términos de portabilidad, no han alcanzado niveles de rendimiento aceptables en las métricas semánticas ni en ROUGE-L, mostrando limitaciones importantes tanto en calidad como en eficiencia.

En conjunto, el análisis sugiere que el modelo `gemma:2.2b` representa la opción más equilibrada para una implementación funcional del sistema, mientras que `mistral:7b` podría ser reservado para entornos donde la precisión sea prioritaria sobre el coste computacional.

Además de la evaluación técnica realizada localmente, el despliegue en Hugging Face Spaces proporciona un canal accesible para que potenciales usuarios exploren el asistente y ofrezcan retroalimentación cualitativa. Este enfoque abre la posibilidad de recopilar impresiones directas de estudiantes universitarios, identificar casos límite no contemplados en el corpus inicial y analizar la utilidad percibida del sistema en escenarios reales, sentando así las bases para futuras iteraciones del proyecto.

7 Conclusiones

El presente Trabajo Fin de Grado ha abordado el desarrollo de un asistente conversacional inteligente orientado a facilitar el acceso a información académica por parte de estudiantes universitarios, aplicando una arquitectura de tipo *Retrieval-Augmented Generation* (RAG). La solución propuesta se caracteriza por combinar técnicas de recuperación semántica de documentos con la generación de respuestas en lenguaje natural, mediante el uso de modelos ejecutados en local sin dependencia de plataformas en la nube.

A lo largo del proyecto se ha diseñado e implementado una arquitectura modular basada en herramientas de código abierto como **LangChain**, **Gradio**, **FastAPI**, **ChromaDB** y modelos de lenguaje alojados en **Ollama**. Esta combinación ha permitido construir un sistema robusto, accesible y adaptable a nuevas fuentes documentales y entornos de despliegue limitados (GPU local de 4GB).

Entre los resultados más relevantes destaca la selección del modelo **gemma2:2b** como generador principal, tras una comparativa empírica con varios modelos ligeros. Dicho modelo ha ofrecido un excelente equilibrio entre precisión, claridad y eficiencia, situándose como la opción más adecuada para el entorno definido. La evaluación funcional del sistema ha evidenciado su capacidad para responder con coherencia a consultas frecuentes del ámbito universitario, especialmente en temas como normativa, calendario o guías docentes.

7.0.1 Limitaciones detectadas

A pesar de los buenos resultados obtenidos, el sistema presenta ciertas limitaciones:

- La calidad de las respuestas depende en gran medida de la precisión del proceso de recuperación semántica. En algunos casos, si el documento adecuado no se encuentra entre los más relevantes, el modelo generativo puede fallar.

- El rendimiento en términos de tiempo de inferencia y consumo de memoria es sensible al tamaño del modelo, lo que impone restricciones en entornos con recursos limitados.
- La cobertura temática está restringida al contenido procesado. Si un área no está representada en el corpus, el sistema no puede responder adecuadamente.

7.0.2 Líneas futuras de mejora

A partir del desarrollo realizado, se plantean diversas líneas de mejora y expansión:

- **Ampliación del corpus**: incluir documentos de otros grados y servicios de la universidad para hacer el asistente transversal.
- **Uso de embeddings híbridos**: combinar distintos modelos de embeddings para mejorar la cobertura semántica multilingüe.
- **Integración en plataformas institucionales**: conectar el backend actual con portales universitarios, apps móviles o sistemas de ticketing académico.
- **Sistema de retroalimentación**: permitir al usuario valorar la utilidad de las respuestas y ajustar dinámicamente el sistema a partir de esos datos.
- **Cota pesimista y búsqueda voraz**: integrar estrategias heurísticas adicionales para mejorar la eficiencia del sistema en consultas complejas.
- **Validación externa mediante Spaces**: realizar campañas controladas de prueba del sistema en Hugging Face Spaces, con el objetivo de obtener datos cualitativos adicionales sobre la satisfacción del usuario y la cobertura temática del asistente.

En conclusión, este trabajo demuestra que es posible construir un asistente académico funcional, eficiente y ejecutable localmente utilizando modelos ligeros y herramientas de código abierto. Su arquitectura modular y replicable lo convierte en una base sólida sobre la cual seguir desarrollando soluciones adaptadas al entorno universitario.

Bibliografía

AI, F. (2020). *mbart: Multilingual denoising pretraining for neural machine translation.* Descargado de https://huggingface.co/facebook/mbart-large-50-many-to-many-mmt (Accedido en junio de 2025)

AI, M. (2022). *Nllb-200: Enabling high-quality multilingual machine translation.* Descargado de https://ai.facebook.com/research/publications/nllb-200-enabling-high-quality-multilingual-machine-translation/ (Accedido en junio de 2025)

Bai, Y., Kadavath, S., Kundu, S., y cols. (2022). Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*. Descargado de https://arxiv.org/abs/2212.08073

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., … Amodei, D. (2020a). Language models are few-shot learners. *CoRR*, *abs/2005.14165*. Descargado de https://arxiv.org/abs/2005.14165

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., … Amodei, D. (2020b). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, *33*, 1877–1901. Descargado de https://arxiv.org/abs/2005.14165

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., … Fiedel, N. (2022). *Palm: Scaling language modeling with pathways.* Descargado de https://arxiv.org/abs/2204.02311

Cloud, G. (2023). *Vertex ai search documentation.* Descargado de https://cloud.google.com/vertex-ai/docs/search (Accedido en junio de 2025)

Cohere. (2023). *Cohere rerank documentation.* Descargado de `https://docs.cohere.com/docs/rerank`

Colby, K. M., Hilf, F. D., Weber, S., y Kraemer, H. C. (1972). Turing-like indistinguishability tests for the validation of a computer simulation of paranoid processes. *Artificial Intelligence*, *3*(3), 199–221. doi: 10.1016/0004-3702(72)90013-X

DeepMind, G. (2023a). *Gemini.* Descargado de `https://deepmind.google/technologies/gemini` (Accedido en junio de 2025)

DeepMind, G. (2023b). Gemini: A family of highly capable multimodal models. *Technical report.* Descargado de `https://storage.googleapis.com/deepmind-media/gemini/gemini-technical-report.pdf`

DeepMind, G. (2024). *Gemma: Lightweight, open models.* Descargado de `https://ai.google.dev/gemma`

DeepSeek-AI. (2024). Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954.* Descargado de `https://github.com/deepseek-ai/DeepSeek-LLM`

deepset. (2023). *Haystack - nlp framework for search.* Descargado de `https://haystack.deepset.ai/` (Accedido en junio de 2025)

Developers, L. (2023). *Langchain documentation.* Descargado de `https://docs.langchain.com`

Devlin, J., Chang, M.-W., Lee, K., y Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint*, *arXiv:1810.04805*.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, *14*(2), 179–211.

GitHub. (2021). *Github copilot.* Descargado de `https://github.com/features/copilot` (Accedido en junio de 2025)

Hochreiter, S., y Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Jurafsky, D., y Martin, J. H. (2009). *Speech and language processing* (2a ed.). Prentice Hall.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., … Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. En *Advances in neural information processing systems (neurips)* (Vol. 33, pp. 9459–9474). Descargado de `https://arxiv.org/abs/2005.11401`

Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. En *Text summarization branches out* (pp. 74–81). ACL.

MonkeyLearn. (2023). *Sentiment analysis tool.* Descargado de `https://monkeylearn.com/sentiment-analysis/` (Accedido en junio de 2025)

OpenAI. (2022). *Introducing chatgpt.* Descargado de `https://openai.com/blog/chatgpt`

Raffel, C., y cols. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, *21*(140), 1–67.

Rajpurkar, P., Zhang, J., Lopyrev, K., y Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*. Descargado de `https://arxiv.org/abs/1606.05250`

Ramírez, S. (2023). *Fastapi: Modern web framework for apis with python 3.6+.* Descargado de `https://fastapi.tiangolo.com`

Reimers, N., y Gurevych, I. (2023). *Sentence transformers: Multilingual embeddings.* Descargado de `https://www.sbert.net`

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408. doi: 10.1037/h0042519

Rumelhart, D. E., Hinton, G. E., y Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533–536. doi: 10.1038/323533a0

SciSummary. (2023). *Scisummary - ai-powered scientific paper summaries.* Descargado de `https://scisummary.com` (Accedido en junio de 2025)

Services, A. W. (2022). *Codewhisperer.* Descargado de `https://aws.amazon.com/codewhisperer/` (Accedido en junio de 2025)

Software, A. (2023). *Pymupdf documentation.* (`https://pymupdf.readthedocs.io`)

Team, C. (2023). *Chroma: Open-source embedding database.* Descargado de `https://docs.trychroma.com`

Team, G. (2023). *Gradio: Create uis for machine learning models in python.* Descargado de `https://www.gradio.app`

Team, O. (2024). *Ollama: Run llms locally.* Descargado de `https://ollama.com`

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., y cols. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971.*

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *LIX*(236), 433–460. doi: 10.1093/mind/LIX.236.433

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*.

Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, *9*(1), 36–45. doi: 10.1145/365153.365168

Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., y Artzi, Y. (2019). Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*. Descargado de `https://arxiv.org/abs/1904.09675`

Lista de Acrónimos y Abreviaturas

IA Inteligencia Artificial.

LLM Modelo de Lenguaje de Gran Escala.

LLMs Modelos de Lenguaje de Gran Escala.

LSTM Long Short-Term Memory.

NLP Procesamiento del Lenguaje Natural.

QA Quality Assurance.

RAG Retrieval-Augmented Generation.

RNN Redes Neuronales Recurrentes.