Universidad de Cantabria

Facultad
de
Ciencias

DESARROLLO DE UN COMPONENTE DE CHATBOT CON IA PARA EL ENTORNO INDUSTRIAL

(Development of an AI-powered chatbot component for industrial environments)

Trabajo de Fin de Grado
para acceder al

GRADO EN INFORMÁTICA

Autor: Aimar Gómez Díaz

Director: Diego García Saiz

Co-Director: Miguel Sierra Sánchez

Junio – 2025

Resumen

Este Trabajo de Fin de Grado se elabora en colaboración con una empresa del sector industrial. Tiene como principal objetivo el diseño e implementación de un asistente conversacional basado en Inteligencia Artificial, destinado a mejorar el rendimiento en las tareas diarias de los operarios de las fábricas. El software desarrollado facilita a los trabajadores la consulta de información relevante sobre los activos de la planta.

El asistente se basa en un modelo de lenguaje de gran tamaño, combinado con técnicas de recuperación de la información mediante *embeddings* semánticos, almacenados en una base de datos vectorial.

Este trabajo sienta las bases de una herramienta con un alto potencial de mejora en eficiencia y accesibilidad de la información para los trabajadores de planta, maximizando su rendimiento.

Abstract

This Final Degree Project is developed in collaboration with a company in the industrial sector. It aims to design and implement a conversational assistant based on artificial intelligence, focused on improving the efficiency in the daily operations of factory workers. The developed system allows employees to access relevant information about plant assets

The assistant is built upon a large language model combined with an information retrieval system using semantic embeddings stored in a vector database.

This project lays the foundation for a tool with great potential to improve the efficiency and accessibility of information for plant workers, maximizing their performance.

Palabras clave

Gran Modelo de Lenguaje, *Embeddings* Semánticos, Procesamiento del Lenguaje Natural, Generación Aumentada por Recuperación, Base de Datos Vectorial.

Keywords

Large language Model, Semantic Embeddings, Natural Language Processing, Retrieval-Augmented Generation, Vector Database.

Índice

Tabla de contenido

Tabla de contenido

Tabla de contenido

1. Introducción

En este apartado se introduce el proyecto, presentando su contexto, la motivación que lo impulsa, los objetivos planteados y la estructura general del proyecto

1.1 Contextualización del proyecto

En la actualidad y en lo que respecta a la transformación digital en la industria, las empresas tienen la necesidad de optimizar sus procesos de producción. Esto se consigue gracias a la mejora en la eficiencia de sus procesos y a garantizar la disponibilidad de información de forma ágil y precisa. En este contexto, las tecnologías basadas en Inteligencia Artificial (IA), y más concretamente en procesamiento del lenguaje natural, tienen un papel fundamental, facilitando la interacción entre los trabajadores y los distintos sistemas de información [18].

En los entornos industriales los operarios de planta se enfrentan diariamente a la necesidad de consultar documentación técnica, procedimientos de mantenimiento, protocolos de seguridad o registros de incidencias. Normalmente, esta información se encuentra en manuales o archivos físicos, bases de datos, hojas de cálculo o archivos PDF, lo que dificulta su disponibilidad inmediata, retrasando así la jornada laboral. Esta falta de inmediatez en el acceso a toda la información puede derivar en tiempos de inactividad, retrasos o errores en la ejecución de tareas e incluso dependencias innecesarias entre distintos operarios y supervisores.

Ante esta problemática surge la necesidad de desarrollar asistentes inteligentes capaces de ofrecer, de forma intuitiva, rápida y accesible, la consulta de información técnica necesaria para resolver las necesidades surgidas. El uso de modelos de lenguaje, junto con el uso de diferentes técnicas que permitan la personalización de estos y el empleo de *embeddings* semánticos almacenados en bases de datos vectoriales, permitiendo la búsqueda de la información relevante, permite diseñar sistemas conversacionales capaces de ofrecer respuestas precisas y contextualizadas, incluso ante preguntas complejas formuladas en lenguaje natural.

El desarrollo del prototipo inicial en Google Colab[1], y su posterior despliegue en una instancia de Google Compute Engine (GCE), han permitido construir una infraestructura funcional capaz de ofrecer el servicio a través de una API con Uvicorn, facilitando así su integración en plataformas industriales existentes.

Este proyecto se enmarca, por tanto, dentro de la evolución de los sistemas industriales hacia entornos más conectados, autónomos y eficientes, aportando una solución tecnológica basada en IA que contribuye directamente a la mejora del rendimiento y la autonomía de los operarios de planta, alineándose con los principios de la Industria 4.0 [9].

[1] Para más información: https://colab.research.google.com/

1.2 Motivación

La elaboración de este TFG tiene tanto a un interés académico como profesional. Desde el inicio de mi formación, he mostrado un alto interés por el desarrollo de la IA aplicada a entornos reales. En este sentido, el proyecto es una oportunidad única para aplicar todos los conocimientos adquiridos durante la carrera en un entorno real, desarrollando una solución que puede tener un alto impacto en la eficiencia y el día a día de los operarios de fábricas.

La posibilidad de colaborar con una empresa como Soincon – EMI Suite 4.0, especializada en soluciones de software industrial, me ha permitido comprobar de primera mano, tanto el funcionamiento real de una empresa como la realización de un proyecto real, todo ello desde una perspectiva profesional, trabajando con datos reales y enfrentándome a los distintos retos propios del entorno. Además, me ha permitido familiarizarme con herramientas y tecnologías punteras como modelos de lenguaje, bases de datos vectoriales o entornos de despliegue en la nube, que no solo enriquecen mi perfil técnico, sino que también abren nuevas posibilidades para mi futuro.

Más allá del componente técnico, este proyecto representa también una motivación personal: contribuir a crear soluciones tecnológicas que sean realmente útiles para las personas que trabajan en entornos exigentes, como lo es una planta industrial. Poder facilitar su trabajo diario mediante herramientas más intuitivas y accesibles me parece una forma concreta y valiosa de aplicar la ingeniería al servicio de las personas.

Por último, el enfoque del proyecto, basado en tecnologías actuales como la IA, el procesamiento del lenguaje natural o técnicas como la Generación Aumentada por Recuperación (RAG), representa una excelente oportunidad para adentrarme en un campo de innovación con gran proyección, y sentar las bases para futuras especializaciones o investigaciones en el ámbito de los sistemas inteligentes aplicados a la industria.

1.3 Objetivos

Objetivo General

Desarrollar un *chatbot* basado en IA que permita a los operarios de planta consultar información técnica relevante de forma rápida y sencilla, sin necesidad de acceder manualmente a archivos físicos o digitales, mejorando así la eficiencia y autonomía en sus tareas diarias.

Objetivos

- Implementar un mecanismo de recuperación de información basado en *embeddings* semánticos, almacenados y consultados a través de una base de datos vectorial.

- Procesar e indexar diversas fuentes de información proporcionadas por la empresa, incluyendo bases de datos estructuradas y documentación técnica en formato PDF.

- Desplegar el prototipo funcional en la nube mediante GCE, asegurando su accesibilidad a través de una API REST. Tabla 2 Requisitos no funcionales

- Permitir a los operarios realizar consultas mediante lenguaje natural sobre información técnica relevante (errores comunes, procedimientos, normativas, etc.).

- Garantizar que las respuestas generadas por el sistema sean precisas, claras y contextualizadas con respecto a las necesidades del operario.

1.4 Estructura del documento

Este documento está organizado en nueve capítulos, los cuales abordan de forma progresiva el desarrollo del trabajo de fin de grado hasta llegar a la presentación de los resultados, conclusiones y anexos complementarios.

En primer lugar, la Introducción presenta el contexto general en el que se enmarca el trabajo, junto con la motivación personal que impulsa su realización, los objetivos que se persiguen y una explicación final de cómo está organizado el documento.

A continuación, el capítulo 2, dedicado al estado del arte, ofrece una revisión crítica y detallada de diferentes enfoques y soluciones existentes en el ámbito de los *chatbots* y los modelos de lenguaje, con el fin de situar el proyecto en el marco actual del desarrollo tecnológico y detectar oportunidades de mejora o innovación.

El marco teórico profundiza en los conceptos fundamentales que sustentan el proyecto, abordando temas como los asistentes conversacionales, los sistemas RAG, los modelos de procesamiento del lenguaje natural y las tecnologías utilizadas, proporcionando una base sólida para comprender el desarrollo posterior.

En el capítulo 3 se describen las distintas fases del desarrollo del proyecto, desde la planificación inicial hasta la obtención de un prototipo funcional, incluyendo la metodología aplicada.

El capítulo 4 recoge la especificación de requisitos, tanto funcionales como no funcionales, que definen el comportamiento esperado del sistema, así como los casos de uso y restricciones del entorno industrial.

En el capítulo de diseño y desarrollo se detallan tanto la estructura técnica del sistema como su funcionamiento. Se abordan los procesos de extracción y recuperación de información, el diseño de la arquitectura, la implementación del sistema y las decisiones tecnológicas adoptadas.

Posteriormente, el apartado de análisis, resultados y pruebas examina el rendimiento del sistema implementado, evalúa la calidad de los resultados obtenidos mediante pruebas de aceptación y uso real, y reflexiona sobre las principales limitaciones encontradas durante la validación del sistema.

Finalmente, en el capítulo de Conclusiones y líneas de trabajo a futuro, se resumen los hallazgos más significativos del proyecto, se valora el impacto de los resultados obtenidos tanto a nivel técnico como práctico, y se proponen posibles mejoras, así como nuevas líneas de desarrollo que podrían explorarse en trabajos futuros.

El documento se completa con los capítulos de Bibliografía, que recoge todas las fuentes consultadas y utilizadas a lo largo del proyecto. A través de una revisión de la situación actual de las tecnologías involucradas, se busca fundamentar el enfoque implementado en este TFG, así como identificar las líneas principales de investigación, aplicaciones y desafíos existentes.

2. Estado del arte

En los últimos años, la transformación digital en la industria, impulsada por el paradigma de la Industria 4.0, ha favorecido la aparición de tecnologías basadas en IA en busca de la optimización de procesos, reducción de errores y una mejora en la eficiencia operativa. En este contexto, los asistentes conversacionales o *chatbots* han emergido como herramientas clave para facilitar el acceso a información técnica, reducir la carga de los operarios y mejorar la respuesta ante incidencias en tiempo real.

Este estado del arte examina los avances más relevantes en la aplicación de chatbots industriales, el uso de técnicas de procesamiento del lenguaje natural (NLP) en contextos técnicos, y el papel emergente de los modelos RAG como solución para entornos donde la información debe recuperarse de fuentes complejas y no estructuradas.

2.1 Chatbots en entornos industriales

El uso de asistentes conversacionales ha sido ampliamente explorado en sectores como el comercio electrónico, la atención al cliente o la educación. Sin embargo, su aplicación en entornos industriales requiere un enfoque especializado. A diferencia de los escenarios más comunes, donde las interacciones suelen ser genéricas y orientadas a tareas repetitivas, en la industria los operarios demandan información técnica precisa, contextualizada y, a menudo, crítica para el funcionamiento seguro de equipos o procesos [11].

En este contexto, los *chatbots* deben responder a requerimientos más exigentes en cuanto a fiabilidad, robustez y adaptación al dominio. Su principal función no es únicamente la atención al usuario, sino la consulta eficaz de documentación técnica, la resolución de incidencias operativas y el soporte a la toma de decisiones en tiempo real. Esto implica trabajar con fuentes de información muy heterogéneas como manuales, hojas de datos, protocolos o historiales de mantenimiento que normalmente no están estructuradas, y que requieren técnicas avanzadas de recuperación de información.

Diversas investigaciones han abordado el diseño de *chatbots* industriales con enfoques centrados en la mejora de la productividad, la reducción de errores humanos y la formación continua de los operarios.

En aplicaciones reales, algunas empresas del sector han empezado a experimentar con soluciones propietarias o adaptaciones de *frameworks* conversacionales existentes, como IBM Watson Assistant o Microsoft Power Virtual Agents, aunque estas soluciones presentan limitaciones cuando se enfrentan a información no estructurada o altamente técnica [8].

En resumen, aunque los chatbots industriales aún están lejos de una adopción masiva, se reconoce su potencial para mejorar la eficiencia operativa y la gestión del conocimiento dentro de entornos industriales. Esta línea de innovación se alinea directamente con la visión de fábricas inteligentes, donde la interacción entre humanos y sistemas debe ser fluida, accesible y efectiva.

2.2 Sistemas RAG y su papel

Los sistemas de RAG han emergido como una solución eficaz para tareas de respuesta a preguntas y generación de texto en dominios donde la información no está contenida íntegramente en los parámetros del modelo de lenguaje, sino que debe ser recuperada desde fuentes externas. Esta arquitectura combina dos componentes fundamentales: un motor de recuperación de información y un generador de texto basado en modelos de lenguaje natural [7].

A diferencia de los modelos puramente generativos, que intentan responder utilizando únicamente el conocimiento aprendido durante el entrenamiento, los sistemas RAG consultan una base documental para encontrar fragmentos relevantes que luego son utilizados como contexto para generar una respuesta. Esto los hace especialmente adecuados para dominios técnicos, donde la información suele ser muy específica, dinámica y difícil de generalizar en un modelo preentrenado.

Esta arquitectura ha demostrado ser particularmente útil en escenarios donde la documentación técnica está en formato no estructurado (PDFs, manuales, registros) y se requiere una forma de consulta accesible y rápida para el usuario final. En el ámbito industrial, esto puede traducirse en herramientas que permiten a los operarios acceder a manuales de mantenimiento, guías de solución de errores o procedimientos de seguridad sin necesidad de recorrer múltiples documentos.

Además, RAG permite mantener el modelo base sin necesidad de un *fine-tuning* costoso, ya que la personalización se logra mediante la actualización del corpus documental y del índice vectorial, lo que facilita la escalabilidad y el mantenimiento del sistema.

Entre sus limitaciones, cabe destacar la necesidad de una buena calidad documental, el riesgo de recuperar fragmentos irrelevantes si los *embeddings* no están bien optimizados, y los posibles problemas de trazabilidad si el sistema no expone claramente la fuente de cada respuesta. A pesar de ello, su flexibilidad y capacidad de adaptación

hacen de RAG una de las arquitecturas más prometedoras para la implementación de asistentes conversacionales técnicos en entornos reales.

En la Figura 1, se puede observar el funcionamiento completo del sistema descrito. El usuario introduce la *query* o pregunta, esta se envía a la base de datos vectorial, la cual nos devuelve los datos relevantes a dicha consulta. Posteriormente, se envía al LLM todos los datos, la pregunta y los *chunks* de información. Una vez el LLM procesa la información, genera la respuesta, la cual es enviada al usuario de vuelta. De esta forma, el LLM obtiene la información suficiente para responder a la pregunta del usuario, siempre y cuando se tenga información relevante sobre la misma.

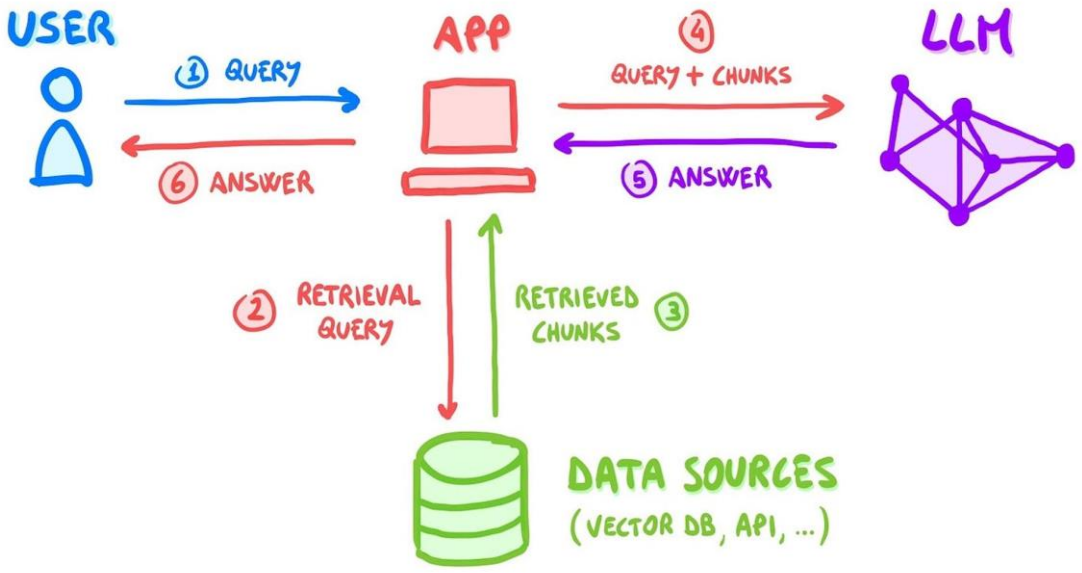


Figura 1 Funcionamiento general de un chatbot[2]

2.3 Chatbots y asistentes conversacionales

Los *chatbots* y asistentes conversacionales son sistemas de software diseñados para simular una conversación con seres humanos, a través de interfaces de texto o voz, con el objetivo de brindar asistencia, resolver dudas, ejecutar tareas o proporcionar información de manera automatizada. Estos sistemas forman parte del campo de la IA y, en particular, del Procesamiento del Lenguaje Natural (PLN), un área encargada de dotar a las máquinas de la capacidad de comprender, interpretar, generar y responder en lenguaje humano.

[2] Adaptado de "Meeting buffers, naming files, and how not to be boring," por Refactoring.fm, 2023, https://refactoring.fm/p/meeting-buffers-naming-files-and

2.3.1 ChatBot: Definición y tipología

Un *chatbot* (contracción de "chat" y "robot") es una aplicación informática que interactúa con los usuarios mediante una interfaz conversacional. Dependiendo de su complejidad y capacidades, los *chatbots* pueden clasificarse en dos grandes categorías:

1. *Chatbots* basados en reglas:
    - Funcionan mediante un conjunto de reglas predefinidas y flujos de decisión.
    - Son útiles para escenarios simples y estructurados, como responder preguntas frecuentes (FAQs) o guiar formularios.
    - No entienden el contexto ni el significado semántico; responden en función de palabras clave o decisiones condicionales.

2. *Chatbots* basados en IA:
    - Utilizan técnicas de aprendizaje automático y procesamiento del lenguaje natural.
    - Son capaces de comprender la intención del usuario (*intents*), extraer entidades relevantes (*entities*), mantener el contexto conversacional y aprender con el tiempo.
    - Se adaptan a una mayor variedad de expresiones humanas, incluyendo errores gramaticales, sinónimos o ambigüedades.

2.3.2 Asistentes conversacionales: definición y diferenciación

Un asistente conversacional es una evolución más avanzada del *chatbot*, generalmente con una capacidad más amplia de interacción, comprensión del lenguaje natural y ejecución de tareas complejas. A diferencia de los *chatbots* simples, los asistentes conversacionales:

- Son capaces de mantener diálogos contextuales prolongados, recordando la información mencionada anteriormente en la conversación.
- Integran múltiples fuentes de datos o APIs para ejecutar tareas (por ejemplo, consultar una base de datos, activar dispositivos IoT, reservar una cita, etc.).
- Pueden comunicarse mediante texto, voz, o incluso lenguaje visual (como asistentes multimodales).
- Están diseñados para adaptarse a diferentes dominios (por ejemplo, atención al cliente, salud, finanzas, industria, etc.).

Ejemplos de asistentes conversacionales ampliamente conocidos son Google Assistant, Amazon Alexa, Apple Siri o Microsoft Cortana. Sin embargo, también se desarrollan asistentes conversacionales específicos para entornos industriales, empresariales o educativos.

2.3.3 Componentes clave de un *chatbot* o asistente conversacional

Para comprender en profundidad cómo funcionan estos sistemas, es esencial analizar sus principales componentes:

1. **Procesamiento del lenguaje natural:**
    - Permite interpretar el lenguaje humano. Se divide en varios procesos:
        - NLU (*Natural Language Understanding*): interpreta lo que el usuario quiere decir.
        - NLG (*Natural Language Generation*): genera una respuesta coherente en lenguaje natural.
        - NPL (*Natural Languaje Processing*): se encarga del procesado básico del lenguaje natural, como *tokenizacion*, lematización o detección de intenciones.
    - Dentro de NLU se encuentran tareas como extracción de entidades, análisis sintáctico y semántico, etc.

2. **Motor de diálogo:**
    - Controla el flujo de la conversación.
    - Decide cómo reaccionar ante la entrada del usuario y qué acción o respuesta generar.
    - Puede incluir lógica condicional, aprendizaje por refuerzo, o gestión de contexto.

3. **Base de conocimientos:**
    - Es la fuente de información desde donde se extraen datos para responder.
    - Puede ser estática (FAQs, documentos) o dinámica (consultas a bases de datos, APIs, sistemas externos).

4. **Módulo de integración:**
    - Se encarga de conectar el chatbot con otros sistemas o servicios (por ejemplo, CRM, sistemas ERP, motores de búsqueda, dispositivos industriales).

5. **Interfaz de usuario:**
    - Es el canal por el que el usuario interactúa con el chatbot.
    - Puede ser textual (webchat, Telegram), de voz (Alexa, Google Assistant), o híbrida (aplicaciones móviles con entrada de texto/voz).

Todos estos elementos se pueden observar con claridad en la Figura 2, junto con los pasos que se realizan en el flujo conversacional. Un aspecto relevante que destacar es que, en dicha imagen, no se menciona explícitamente el uso de un LLM. Sin embargo, en arquitecturas modernas, un LLM sustituye los bloques de NLP, NLG, NLU y ML (*Machine Learning*), ya que estos modelos integran de forma nativa capacidades de comprensión y generación de lenguaje, así como la integración de técnicas que sustituyen al componente ML.

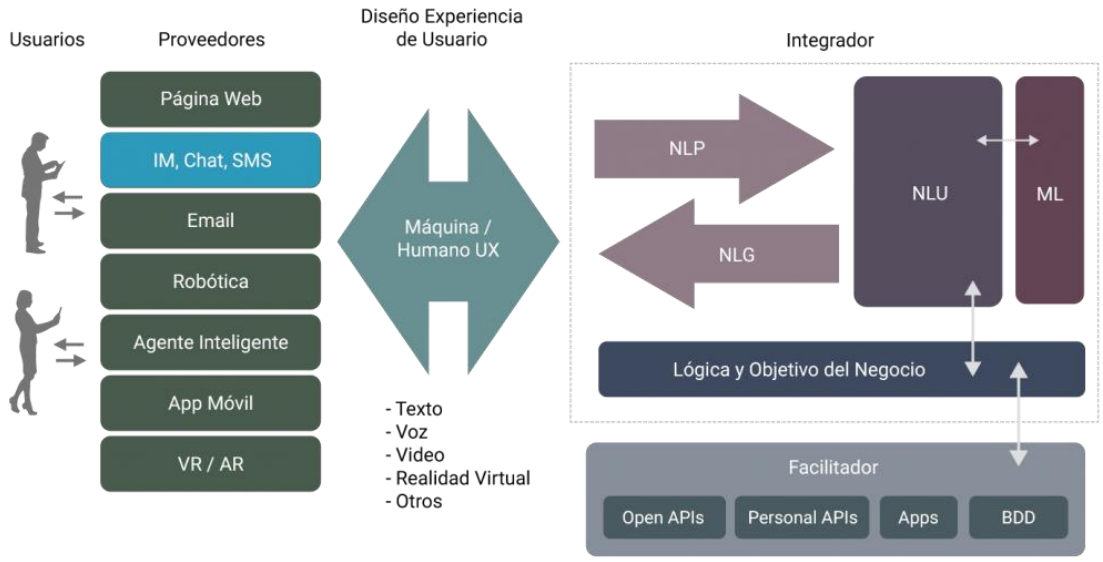


Figura 2 Arquitectura de un chatbot[3]

2.3.4 Tecnologías involucradas

El desarrollo de chatbots y asistentes conversacionales requiere la integración de múltiples tecnologías provenientes de distintas ramas de la informática, la IA y el desarrollo web. A continuación, se describen las principales tecnologías involucradas:

Lenguajes de programación

Los lenguajes de programación constituyen la base para implementar la lógica, la integración con modelos de lenguaje, y la comunicación con bases de datos o interfaces de usuario. Entre los más utilizados destacan:

- Python: Ampliamente utilizado por su sintaxis clara y su ecosistema de librerías para IA (como TensorFlow, PyTorch o HuggingFace Transformers) y procesamiento de lenguaje natural (NLP).
- JavaScript: Fundamental para el desarrollo de interfaces web interactivas, especialmente mediante frameworks como React o Vue.js.
- Java y otros lenguajes: Empleados en sistemas empresariales por su robustez, escalabilidad y compatibilidad con entornos corporativos.

Modelos de lenguaje

Los modelos de lenguaje son el núcleo de un chatbot avanzado, ya que permiten comprender y generar texto de manera coherente. Algunos de los más relevantes son:

- GPT (Generative Pre-trained Transformer): Desarrollado por OpenAI, es uno de los modelos más avanzados en comprensión y generación de texto.

[3] De "Arquitectura de un chatbot," por P. Cornejo, 2019, *Medium*. https://medium.com/@patcornejo/arquitectura-de-un-chatbot-cb2d1c5f86c7.

- Mistral: Modelos open-source eficientes diseñados para tareas de NLP a gran escala.
- LLaMA: Desarrollado por Meta, es un modelo entrenado con enfoque en eficiencia y rendimiento, cada vez más usado en entornos académicos y empresariales.
- Otros modelos open-source o propietarios también pueden integrarse, según los requisitos de precisión, coste o privacidad.

Almacenamiento de datos y APIs

Los chatbots suelen necesitar acceso a datos dinámicos, persistencia de conversaciones o integración con sistemas externos:

- Bases de datos: Se utilizan bases de datos relacionales (MySQL, PostgreSQL) para almacenar información estructurada, historial de conversaciones, perfiles de usuario, etc.
- APIs: Las interfaces de programación de aplicaciones permiten conectar el *chatbot* con servicios externos, como sistemas de gestión empresarial (ERP), plataformas de mensajería (Telegram, WhatsApp), o servicios de terceros (OpenWeather, bases de datos, etc.).

Inteligencia artificial y redes neuronales

El uso de técnicas de IA es esencial para mejorar la capacidad del *chatbot* de comprender el lenguaje humano y responder adecuadamente:

- Procesamiento de lenguaje natural: Permite el análisis semántico y sintáctico del texto. Incluye técnicas como el reconocimiento de entidades, análisis de sentimientos, clasificación de intenciones, etc.
- Redes neuronales: Arquitecturas como Transformers se utilizan en el entrenamiento y funcionamiento de modelos de lenguaje.
- Sistemas de aprendizaje automático (Machine Learning): Se aplican para adaptar el comportamiento del *chatbot* a patrones de uso, realizar mejoras personalizadas o entrenar clasificadores para tareas específicas.

***Frameworks* y herramientas de desarrollo**

Para facilitar el diseño, implementación y despliegue de *chatbots*, se emplean múltiples herramientas y *frameworks* especializados:

- FastAPI, Flask: *Frameworks* para crear APIs RESTful en Python.
- LangChain, Haystack: Herramientas para construir sistemas RAG, muy útiles en *chatbots* que acceden a documentos o bases de conocimiento.
- ChromaDB, FAISS, Weaviate: Bases de datos vectoriales utilizadas para realizar búsquedas semánticas sobre *embeddings* generados a partir de texto.
- Docker[4], Kubernetes[5]: Para el despliegue escalable y reproducible de *chatbots* en entornos productivos.

[4] Para más información: https://www.docker.com/

[5] Para más información: https://cloud.google.com/kubernetes-engine?hl=es-419

A continuación, se describen brevemente las principales herramientas y tecnologías empleadas en el desarrollo del trabajo:

- Python: Es un lenguaje de programación de alto nivel, ampliamente utilizado por su sintaxis sencilla y su gran ecosistema de librerías. En este trabajo se ha utilizado como lenguaje principal para implementar la lógica del sistema y gestionar la interacción con las demás herramientas.

- ChromaDB[6]: Es una base de datos vectorial especializada en el almacenamiento y recuperación eficiente de representaciones vectoriales (*embeddings*). Se ha empleado en este proyecto para almacenar y consultar datos semánticos, facilitando la búsqueda y recuperación de información basada en similitud.

- GCE[7]: Es un servicio de infraestructura en la nube proporcionado por Google Cloud Platform que permite desplegar y ejecutar máquinas virtuales. En este trabajo, se ha utilizado para alojar el entorno de desarrollo y desplegar los servicios del sistema, asegurando escalabilidad y disponibilidad.

2.3.5 Aplicaciones en la industria

En el contexto de la industria, los *chatbots* y asistentes conversacionales permiten:

- Asistir a operarios en tareas de mantenimiento o resolución de averías.
- Consultar manuales técnicos o procedimientos sin necesidad de interrumpir el trabajo.
- Acceder a datos de sensores o históricos de producción mediante comandos de voz o texto.
- Digitalizar la experiencia de usuario en entornos industriales complejos.

Esta aplicación está en línea con los principios de la Industria 4.0, donde la automatización, el acceso a la información en tiempo real y la interconectividad son claves para aumentar la eficiencia, reducir errores y mejorar la toma de decisiones.

2.4 Modelos de lenguaje (LLMs)

2.4.1 ¿Qué es un LLM y cómo funciona?

Un LLM es un tipo de modelo de lenguaje basado en IA diseñado para procesar, comprender y generar texto de forma coherente y contextualizada, a partir de grandes volúmenes de datos. Los LLMs modernos están construidos sobre arquitecturas de *deep learning*, en particular sobre el modelo Transformer [17], que ha revolucionado el procesamiento del lenguaje natural [15].

[6] Para más información: https://www.trychroma.com/
[7] Para más información: https://cloud.google.com/products/compute

Estos modelos son la base de sistemas conversacionales avanzados como ChatGPT, asistentes virtuales, motores de búsqueda inteligentes, herramientas de escritura automática, entre otros.

2.4.2 ¿Qué es un modelo de lenguaje?

Un modelo de lenguaje es un tipo de IA diseñado para comprender, generar y trabajar con el lenguaje humano. Estos son entrenados con cantidades ingentes de datos textuales para aprender cómo se utiliza el lenguaje en diferentes contextos y situaciones.

Existen tres tipos principales de modelos:

- Modelos autorregresivos: Son los más comunes y a este tipo pertenecen los modelos más grandes, OpenAI ChatGPT, Google PaLM, Meta LlaMa entre otros. Utilizan únicamente el decodificador *transformer* y son entrenados para predecir la siguiente palabra basándose en el contexto anterior, asignando una probabilidad a una secuencia de palabras o tokens y eligiendo una.

- Modelos enmascarados (bidireccionales): A este grupo pertenecen modelos como BERT (Google), RoBERTa (Facebook) o DistilBERT. Durante su entrenamiento se oculta aleatoriamente un porcentaje del texto (o tokens) y el modelo debe predecir los fragmentos ocultos con los disponibles. Este mecanismo obliga al modelo a comprender el contexto completo y le permite capturar dependencias semánticas complejas.

- Modelos encoder-decoder (seq2seq): Estos modelos emplean la arquitectura completa de los *transformers*. El codificador procesa la entrada y genera una representación intermedia y el decodificador produce de forma autorregresiva la secuencia de salida correspondiente. Por ejemplo, un traductor de idiomas trata de convertir una oración de inglés a español. Primero el codificador traduce la oración en inglés a una representación interna para que, posteriormente, el decodificador genere la oración equivalente en español. Algunos ejemplos de estos modelos son BART de Facebook o T5 de Google.

En este proyecto, los modelos utilizados pertenecen al grupo de los autorregresivos. Para comprender mejor su funcionamiento pondremos un ejemplo.
Si decimos: “El coche es de color _”

El modelo de lenguaje buscará predecir cuál es la palabra más probable para completar esa frase, como “rojo”, “verde” o “azul”, dependiendo del contexto aprendido.

Cada una de las palabras tendrá una probabilidad asignada y se seleccionará una aleatoriamente entre ellas [20].

2.4.3 ¿Qué significa "large" en LLM?

El término large hace referencia al tamaño del modelo, que se puede medir principalmente en dos aspectos:

- Número de parámetros: Los parámetros son los pesos ajustables dentro de una red neuronal. LLMs como GPT-4 tienen cientos de miles de millones de parámetros. Cuanto mayor sea este número, mayor será la capacidad del modelo para aprender relaciones complejas y representar el conocimiento.

- Cantidad de datos utilizados para el entrenamiento: Los LLMs se entrenan con enormes volúmenes de texto, que pueden incluir libros, artículos, páginas web, repositorios de código, etc. Esto les permite tener una visión amplia y rica del lenguaje.

2.4.3 ¿Cómo funciona un LLM? Paso a paso

1. Tokenización

Antes de que el texto pueda ser procesado por el modelo, debe convertirse en un formato numérico. Esto se hace mediante un proceso llamado tokenización.
Un *token* es una unidad básica de texto que puede representar:

- Una palabra.
- Un subgrupo de caracteres.
- Incluso un solo carácter, en modelos más básicos.

Por ejemplo, la frase:
"Los modelos de lenguaje son potentes."
Podría ser tokenizada como:
["Los", " modelos", " de", " lenguaje", " son", " potentes", "."]

Cada uno de esos *tokens* se convierte luego en un número entero que representa una posición en el vocabulario del modelo.

2. Embeddings

Los tokens son solo números enteros. Para que el modelo los procese de forma útil, estos números se convierten en vectores en un espacio de alta dimensión, mediante una técnica llamada *embedding*.

Los *embeddings* permiten representar palabras de manera que:

- Palabras con significados similares estén cerca en el espacio vectorial.
- Se conserve la información semántica.

Por ejemplo, los vectores de "rey" y "reina" estarán cerca, y la relación entre "hombre" y "mujer" será similar a la de "rey" y "reina". Esto puede apreciarse en la Figura 3.

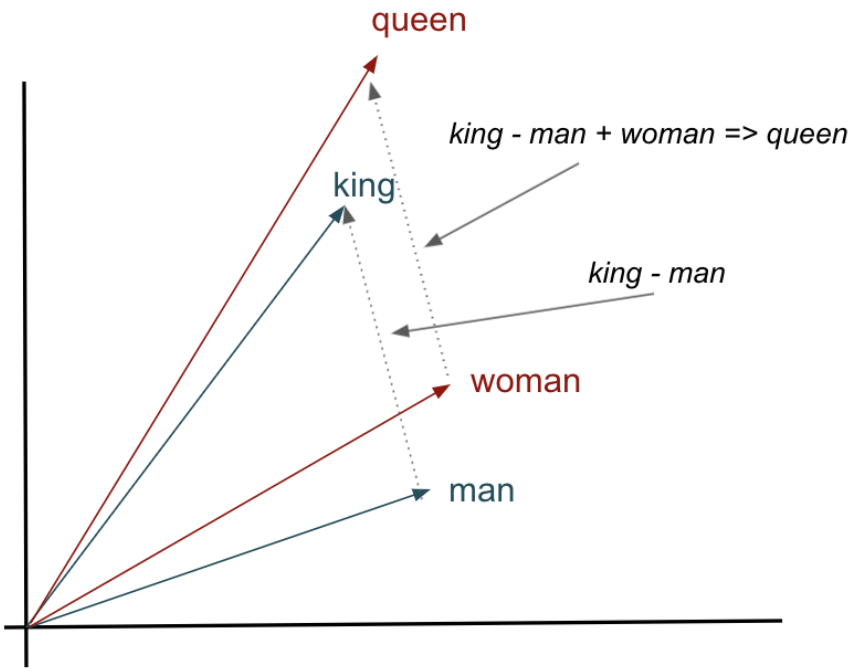


Figura 3 Representación vectorial del significado de palabras. *Fuente: Adaptado de Towards Data Science (s.f.).*

3. El modelo Transformer

Una vez obtenidos los *embeddings*, estos son procesados por una arquitectura llamada Transformer.

Los componentes clave del Transformer son:

- Atención (*Attention Mechanism*):
  - Es el corazón del modelo.
  - Permite que el modelo preste atención a diferentes partes del texto para entender el significado.
  - En la frase: “El perro que perseguía al gato era rápido”, el modelo puede usar la atención para entender que “era rápido” se refiere a “el perro” y no al gato.
- *Multi-head Attention:*
  - No solo se calcula una atención, sino múltiples al mismo tiempo, para capturar diferentes tipos de relaciones en paralelo.
- Capas de *feed-forward*:
  - Después del bloque de atención, se pasa la información por capas densas (completamente conectadas) que aprenden transformaciones no lineales del texto.

4. Entrenamiento (Pre-entrenamiento)
Durante esta fase, el modelo se expone a miles de millones de frases y aprende a predecir la siguiente palabra/tokens en una secuencia. Este entrenamiento se hace sin supervisión directa, mediante tareas como:

- Modelado causal de lenguaje (como en GPT): el modelo ve una secuencia de tokens y predice el siguiente.
- Enmascaramiento de tokens (como en BERT): el modelo debe adivinar qué palabra falta en una frase donde una o más han sido ocultadas.

Este proceso ajusta los millones o miles de millones de parámetros internos del modelo para minimizar el error de predicción.

5. Inferencia (Generación de texto)
Una vez entrenado, el modelo puede utilizarse para generar texto. A partir de un prompt (texto inicial), predice un *token*, lo añade, y luego repite el proceso de forma iterativa.

Durante la generación, se pueden usar diferentes técnicas:

- *Greedy decoding*: elige siempre el token más probable.
- *Sampling*: elige aleatoriamente entre los tokens más probables, para aumentar la creatividad.
- Top-k / Top-p *sampling*: limitan las elecciones a los tokens más probables o al conjunto que acumula una cierta probabilidad.
- Temperature: controla la aleatoriedad; valores bajos implica respuestas más seguras, valores altos más diversidad.

2.4.4 Limitaciones de los LLMs

Los LLMs tienen varias limitaciones:

- No tienen conocimiento real ni conciencia: solo predicen texto en base a patrones estadísticos, sin comprensión profunda del mundo.
- Sensibles al *prompt*: la salida puede variar mucho con pequeños cambios en la entrada.
- Alucinaciones: pueden generar información incorrecta o inventada de forma convincente.
- Coste computacional: entrenar y ejecutar LLMs requiere una gran cantidad de recursos computacionales.

2.4.5. Aplicaciones de los LLMs

- Chatbots y asistentes virtuales.
- Generación automática de código.
- Traducción de idiomas.
- Resumen de documentos.
- Clasificación de texto y análisis de sentimiento.
- Búsqueda semántica y recuperación de información (como en RAG).

2.5 Embeddings y recuperación semántica

2.5.1 ¿Qué son los embeddings?

Los *embeddings* son representaciones vectoriales de datos (como palabras, frases, documentos o incluso imágenes) en un espacio numérico de alta dimensión. Su objetivo principal es capturar el significado semántico de los elementos representados, de forma que elementos similares en contenido estén cercanos entre sí en ese espacio vectorial.

En LLMs modernos, los *embeddings* suelen derivarse de capas intermedias del modelo. Por ejemplo, BERT, GPT o OpenAI *embeddings* API pueden usarse para generar vectores que representan no solo palabras, sino frases completas con su contexto.
Los *embeddings* se representan como vectores de, por ejemplo, 384, 768 o 1536 dimensiones (dependiendo del modelo). Estos vectores permiten operar con distancias y similitudes utilizando métricas como:

- Distancia euclidiana
- Cosine similarity (la más común): mide el ángulo entre dos vectores.

2.5.2 ¿Qué es la recuperación semántica?

La recuperación semántica (*semantic search* o *semantic retrieval*) es una técnica que permite encontrar información relevante no solo por coincidencia exacta de palabras clave, sino por significado. A diferencia de los motores de búsqueda tradicionales basados en texto, la búsqueda semántica utiliza *embeddings* para comparar el significado del contenido.

a) ¿Cómo funciona?

1. **Indexación previa:**
    - Se toman todos los documentos (o fragmentos, párrafos, frases) y se generan sus *embeddings* usando un modelo previamente entrenado.
    - Estos vectores se almacenan en una base vectorial, en este caso ChromaDB.

2. **Consulta del usuario:**
    - Cuando el usuario realiza una pregunta, esta también se convierte en un *embedding* para poder ser procesada.

3. **Cálculo de similitud:**
    - Se compara el *embedding* de la pregunta con los *embeddings* almacenados.
    - Se calcula la similitud para recuperar los textos más cercanos semánticamente.

4. **Recuperación de resultados:**
    - Se devuelven los fragmentos más relevantes, aunque no contengan las mismas palabras exactas, si el significado es el adecuado.

b) Ejemplo práctico:
Supongamos que tenemos una base de datos con este texto:
"La vitamina C fortalece el sistema inmunológico."

Y el usuario pregunta:
"¿Qué nutrientes ayudan a prevenir enfermedades?"

Un motor de búsqueda tradicional probablemente no lo encuentre, ya que no hay coincidencias exactas. Pero un sistema de recuperación semántica sí, porque "fortalece el sistema inmunológico" está conceptualmente relacionado con "prevenir enfermedades", y "vitamina C" es un nutriente.

3. Relación entre *embeddings* y recuperación semántica

- Los *embeddings* permiten transformar contenido textual en representaciones matemáticas con significado.
- La recuperación semántica usa esos vectores para encontrar información relevante en función del contenido, no solo de la forma.
- Esta técnica es especialmente útil en aplicaciones como RAG, donde el sistema necesita buscar documentos relacionados antes de generar una respuesta.

4. Aplicaciones de la recuperación semántica

- Motores de búsqueda inteligentes (como el de Notion, Google Drive, etc.)
- Sistemas de preguntas y respuestas sobre bases documentales.
- *Chatbots* que acceden a información de PDFs, bases de datos, wikis, etc.
- Herramientas de asistencia a la escritura o código, que recuperan ejemplos relevantes.

3. Fases del desarrollo

3.1 Metodología: Prototipado Incremental

Para el desarrollo de este Trabajo de Fin de Grado se ha empleado una metodología de prototipado incremental, orientada a construir un sistema funcional desde etapas tempranas, con mejoras progresivas en base a pruebas prácticas. Esta metodología es especialmente adecuada en entornos de desarrollo experimental y exploratorio, como lo es la creación de un *chatbot* basado en modelos de lenguaje (LLM).

Ventajas de esta metodología
Este enfoque de prototipado permitió:

- Obtener resultados funcionales desde etapas tempranas, facilitando la validación de conceptos.
- Adaptar el desarrollo en función de pruebas, sin necesidad de definir todos los requisitos desde el principio.

- Detectar errores y realizar mejoras rápidamente gracias al enfoque iterativo.
- Evaluar la experiencia de usuario de forma práctica, a través de una interfaz accesible vía web.

Las fases de esta metodología se pueden observar en la Figura 3

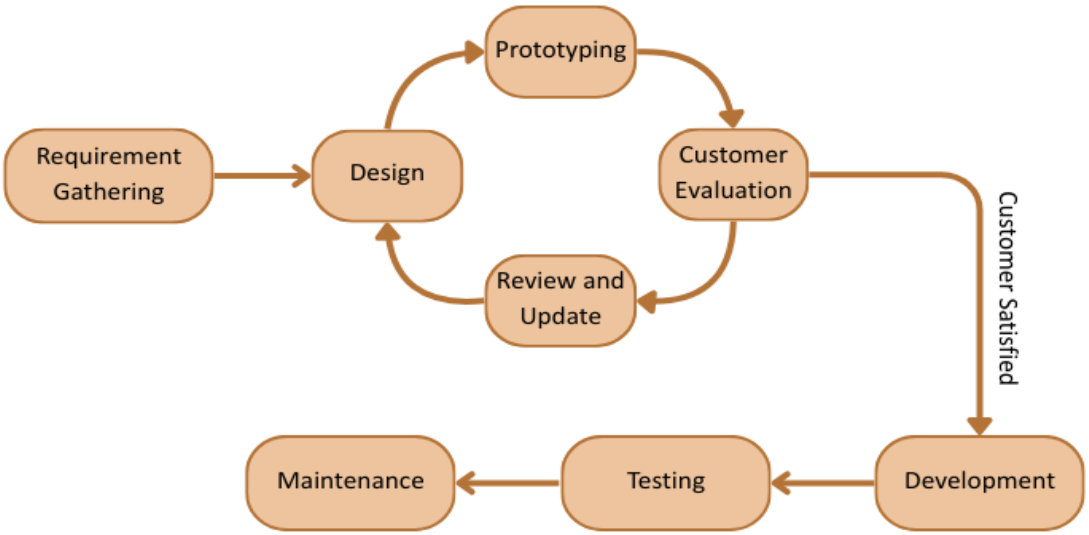


Figura 4: Metodología Prototipado Incremental

En caso de querer conocer más sobre esta metodología, se puede consultar GeeksforGeeks (2025, abril), donde se explica detalladamente el modelo de prototipado.

3.2 Desarrollo

El proceso se dividió en las siguientes fases:

1. **Definición de objetivos y requisitos:** Se definieron los objetivos principales del *chatbot*, centrado en ofrecer respuestas coherentes mediante un modelo LLM. Se determinó la arquitectura del sistema, las herramientas necesarias y el flujo general de funcionamiento: desde la entrada del usuario, pasando por el modelo, hasta la respuesta renderizada en una interfaz web.

2. **Desarrollo del primer prototipo:** El primer prototipo se implementó utilizando Google Colab como entorno de desarrollo, donde se cargó un modelo LLM básico (Falcon). El objetivo inicial fue verificar la capacidad del modelo para generar respuestas a preguntas simples, sin conexión todavía a una interfaz externa.

3. **Integración de *embeddings*:** Para mejorar la relevancia de las respuestas, se integró un modelo de *embeddings*, que permitió representar el significado de los textos en forma vectorial. Se desarrolló un sistema básico basado en RAG, en el que el usuario introducía una consulta y el sistema buscaba la respuesta más relevante, utilizando similitud de vectores.

4. **Implementación de la interfaz web:** Para facilitar las pruebas de usuario, se desarrolló una interfaz web sencilla en HTML, conectada a través de un backend implementado con Flask. Este servidor Flask servía como intermediario entre la interfaz y el modelo de lenguaje alojado en Google Colab. La interfaz fue desplegada de forma local y accesible desde el exterior utilizando ngrok[8], lo que permitió simular su uso en condiciones reales.

5. **Pruebas y mejoras iterativas:** A medida que se avanzaba, se fueron añadiendo algunas funciones más, como la consulta de datos a una base de datos o la implementación de un sistema *feedback*. A demás se fueron realizando pruebas funcionales del *chatbot*, tanto a nivel de lógica como de experiencia de usuario. Se detectaron errores, se ajustaron respuestas, se mejoró la estructura conversacional y se optimizó el procesamiento de entrada y salida. Cada iteración del prototipo incorporaba pequeñas mejoras a nivel técnico y de diseño.

6. **Versión final y documentación:** Tras varias iteraciones, se consolidó una versión estable del *chatbot* con funcionamiento completo: integración del modelo, recuperación semántica mediante *embeddings* y la integración con la aplicación de la empresa.

4. Especificación de requisitos

Este capítulo está dedicado a la descripción de los requisitos obtenidos. Se muestran tanto los requisitos funcionales como los no funcionales.

Requisitos Funcionales (RF)

A continuación, se presentan los requisitos funcionales. La Tabla 1 incluye un breve resumen de estos y posteriormente se ofrece una descripción más detallada.

[8] Para más información: https://ngrok.com/

Requisito	Identificador	Descripción
Consulta de documentos PDF	**RF1**	El *chatbot* debe ser capaz de recibir preguntas del usuario y extraer información relevante desde documentos PDF previamente cargados.
Acceso a base de datos	**RF2**	El *chatbot* debe poder consultar datos técnicos, operativos o de producción desde una base de datos estructurada.
Procesamiento del lenguaje natural	**RF3**	El sistema debe interpretar preguntas en lenguaje natural y devolver respuestas claras y concisas.
Sistema de feedback	**RF4**	El *chatbot* debe permitir que los usuarios valoren la utilidad de una respuesta y dejen comentarios si lo desean.
Respuestas precisas	**RF5**	El *chatbot* debe ser capaz de responder coherente y correctamente a las distintas preguntas sobre la información disponible.

Tabla 1 Requisitos funcionales

RF1-Consulta de documentos PDF

El *chatbot* debe permitir recibir preguntas del usuario y obtener las respuestas a partir de la información contenida en los PDFs previamente cargados. Estos pueden ser manuales técnicos, instrucciones de mantenimiento, procedimientos operativos u otros materiales relevantes para los operarios. Para lograrlo, el sistema deberá extraer el contenido de los mismos, dividirlo en fragmentos más manejables y generando los *embbedings* que serán almacenados en la base de datos vectorial. Para obtener la información más relevante, se utilizarán técnicas de recuperación (RAG).

RF2-Acceso a base de datos

Además de poder consultar PDFs, el software desarrollado deberá ser capaz de acceder a bases de datos estructuradas que contengan información relevante para el usuario. Esas bases de datos pueden incluir, por ejemplo, registros de averías, parámetros de equipos e historiales de mantenimiento entre otros. El *chatbot* deberá ser capaz de elaborar sentencias SQL automáticas en función de la intención del usuario.

RF3-Procesamiento del lenguaje natural
El sistema debe contar con capacidades avanzadas de procesamiento de lenguaje natural que le permitan interpretar preguntas formuladas por los usuarios de manera flexible. Dado que los operarios pueden utilizar expresiones informales, abreviaturas o estructuras gramaticales no estrictas, el *chatbot* debe poder comprender una amplia variedad de formulaciones y responder con claridad y precisión. Para ello, se utilizará un modelo de lenguaje entrenado o ajustado para el dominio industrial, capaz de manejar ambigüedades y extraer la intención subyacente en las consultas.

RF4-Sistema de *feedback*
El sistema debe incorporar un mecanismo mediante el cual los usuarios puedan valorar las respuestas proporcionadas por el *chatbot*. Esta retroalimentación puede expresarse en forma de puntuaciones y también debe permitir comentarios adicionales. Esta funcionalidad es clave para evaluar el rendimiento del sistema, detectar errores o respuestas poco útiles y aplicar mejoras progresivas en el modelo o la base de conocimiento.

RF5-Respuestas precisas
Una funcionalidad esencial del sistema es la capacidad de ofrecer respuestas coherentes, precisas y contextualizadas. El *chatbot* debe combinar la información recuperada de documentos PDF y técnicas de *prompt engineering*, dándole indicaciones al LLM, para poder formular respuestas completas que respondan directamente a la consulta del usuario, evitando ambigüedades, errores de interpretación o en la generación de la respuesta.

Requisitos No Funcionales (RNF)
Los requisitos no funcionales hacen referencia a las características de calidad que debe cumplir el sistema, garantizando su rendimiento, disponibilidad y facilidad de uso
En la Tabla 2 se mencionan con brevedad y, posteriormente, son descritos en detalle.

Requisito	Identificador	Descripción
Rendimiento	**RNF1**	El *chatbot* debe responder en menos de 5 segundos al 90% de las consultas.
Escalabilidad	**RNF2**	El sistema debe ser escalable para soportar múltiples usuarios simultáneamente en una planta o varias ubicaciones.
Disponibilidad	**RNF3**	El sistema debe estar disponible en las horas operativas.
Usabilidad	**RNF4**	La interfaz debe estar adaptada a usuarios sin conocimientos técnicos, con lenguaje claro y navegación intuitiva.
Seguridad	**RNF5**	El sistema debe garantizar la confidencialidad de los datos consultados, especialmente si accede a información sensible de la planta.
Mantenibilidad	**RNF6**	El sistema debe estar diseñado para facilitar actualizaciones futuras, tanto de los datos como del modelo de lenguaje.
Funcionamiento en Local	**RNF7**	El sistema debe de ser capaz de ejecutarse en entornos locales, sin necesidad de conexión a internet.

Tabla 2 Requisitos no funcionales

RNF1-Rendimiento
El sistema debe ser capaz de responder en menos de cinco segundos en al menos el 90% de las consultas realizadas por los usuarios. Este tiempo incluye tanto la interpretación de la pregunta como la recuperación de información y la generación de la respuesta. Un tiempo de respuesta bajo es esencial para no interrumpir el flujo de trabajo de los operarios.

RNF2-Escalabilidad
El sistema debe ser escalable, es decir, debe poder soportar múltiples usuarios operando simultáneamente desde diferentes ubicaciones o dispositivos, sin comprometer su rendimiento. Esta escalabilidad debe contemplar tanto la infraestructura como el diseño

del software, facilitando la incorporación de nuevos usuarios o la conexión desde distintas áreas de la planta.

RNF3-Disponibilidad
El sistema debe estar disponible de forma continua durante las horas operativas de la planta. Esto implica que el *chatbot* debe estar activo y accesible en todo momento mientras los operarios se encuentren trabajando, ya sea en turnos diurnos, nocturnos o durante jornadas extendidas.

RNF4-Usabilidad
La interfaz del sistema debe ser clara, intuitiva y accesible para usuarios sin conocimientos técnicos avanzados. El lenguaje empleado debe ser comprensible y adaptado al contexto industrial, y la navegación debe permitir a los operarios interactuar con el sistema de forma fluida y sin necesidad de formación extensa.

RNF5-Seguridad
Dado que el *chatbot* puede acceder a información sensible sobre la planta, el sistema debe garantizar la confidencialidad, integridad y disponibilidad de los datos. Deben implementarse mecanismos de seguridad como control de acceso, cifrado de datos y restricciones para prevenir usos indebidos.

RNF6-Mantenibilidad
El sistema debe estar diseñado para facilitar su mantenimiento y actualización. Esto incluye la capacidad de incorporar nuevos documentos PDF, modificar las bases de datos, actualizar el modelo de lenguaje o ajustar los parámetros del sistema sin necesidad de reescribir componentes fundamentales.

RNF7-Funcionamiento en Local
El sistema debe poder ejecutarse en entornos locales, sin requerir conexión a internet. Esta característica es fundamental para garantizar la operatividad en entornos industriales donde se trabaja con redes aisladas por razones de seguridad. La solución debe ser autónoma, segura y capaz de operar dentro de la infraestructura local de la planta.

A continuación, en la Figura 5, se presenta el diagrama de casos de uso del sistema junto con una descripción de este.

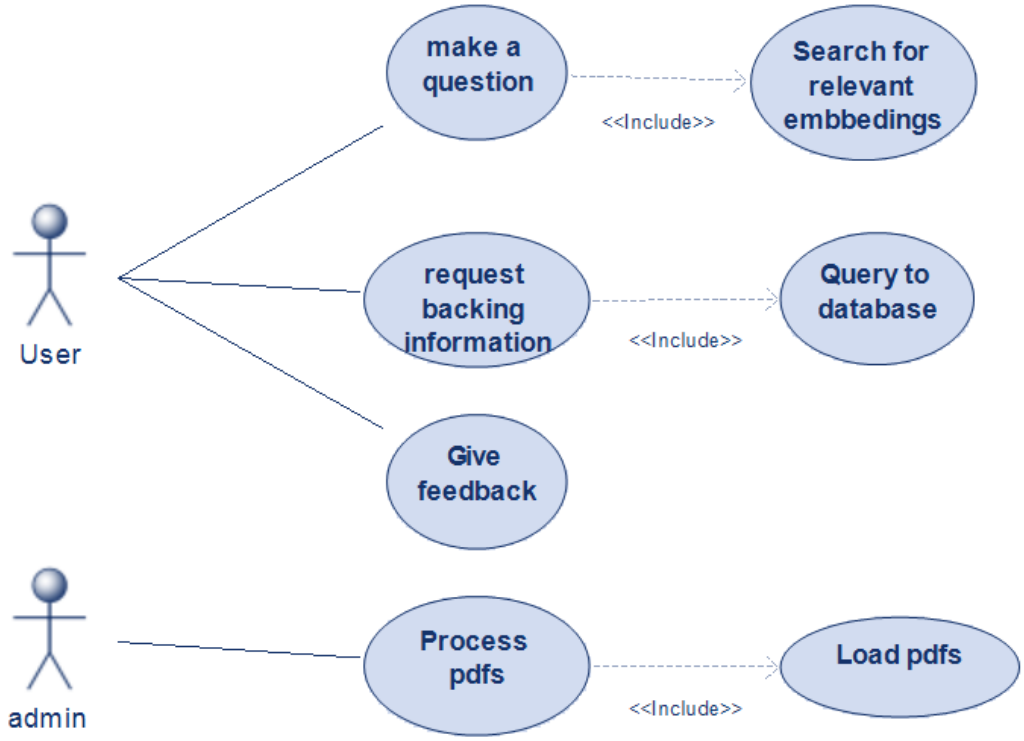


Figura 5 Use Case Diagram

El sistema representado en el diagrama cuenta con dos actores principales: el usuario y el administrador.

El usuario tiene tres funcionalidades principales. La primera es la de hacer una pregunta, lo que activa un mecanismo interno que busca en una base de datos vectorial los *embeddings* más cercanas a la consulta planteada. Esta base de datos se construye a partir de documentos previamente procesados y permite recuperar información de forma eficiente y contextualizada.

La segunda funcionalidad del usuario es la solicitud de refuerzo, mediante la cual se obtiene información adicional o más detallada relacionada con la pregunta inicial. Para ello, el sistema realiza una consulta directa a la base de datos con el objetivo de proporcionar un respaldo más completo y fundamentado a la respuesta ofrecida.

La tercera funcionalidad es la de ofrecer retroalimentación. A través de esta opción, el usuario puede valorar la calidad de la respuesta recibida y dejar comentarios. Esta información es útil para mejorar continuamente el rendimiento del sistema, ya que permite identificar posibles errores o áreas de mejora en la recuperación de información.

Por su parte, el administrador tiene la responsabilidad de alimentar el sistema con nuevos documentos. Para ello, dispone de una funcionalidad que le permite procesar archivos en formato PDF. Este proceso consiste en cargar el documento, fragmentarlo en partes más manejables y generar los correspondientes *embeddings* que luego se

almacenan en la base de datos vectorial. De este modo, el contenido de los nuevos documentos queda disponible para futuras consultas de los usuarios.

5. Diseño y desarrollo

El capítulo actual detalla el diseño técnico del sistema y su funcionamiento junto a una explicación en detalle de estos.

5.1 Proceso de extracción y procesamiento de la información

Para que el chatbot pueda responder de manera efectiva a las preguntas de los operarios, es fundamental proporcionarle acceso a la información contenida en los documentos técnicos y manuales de la fábrica. Estos documentos suelen estar en formato PDF, por lo que se requiere un proceso previo de extracción y estructuración de la información.

El proceso completo se compone de las siguientes etapas:

1. **Carga y lectura de documentos PDF**
   Se utilizan herramientas como PyMuPDF o pdfplumber para extraer el contenido textual de los archivos PDF. Este paso transforma los documentos en texto plano que puede ser procesado posteriormente.

2. **División del texto en fragmentos (splitting)**
   Para facilitar la indexación y posterior búsqueda, el texto se divide en fragmentos más pequeños mediante técnicas de segmentación. Esta división se realiza utilizando LangChain y su utilidad RecursiveCharacterTextSplitter, que permite cortar el texto manteniendo una coherencia semántica y respetando los saltos de sección o párrafo.

3. **Generación de embeddings**
   Cada fragmento de texto es transformado en una representación numérica (*embedding*) que captura su significado semántico. Para ello, se utiliza un modelo de *embeddings* como text-embedding-ada-002 de OpenAI o algún modelo local como los de SentenceTransformers. Estos vectores permiten comparar semánticamente los fragmentos con las consultas realizadas por los usuarios.

4. **Almacenamiento en base de datos vectorial (ChromaDB)**
   Los *embeddings* generados se almacenan junto a su texto original en una base de datos vectorial, en este caso, ChromaDB. Esta base permite búsquedas eficientes mediante la comparación de similitud entre vectores, lo que facilita recuperar los fragmentos más relevantes ante una pregunta.

5. **Consulta mediante RAG**
   Cuando un operario realiza una pregunta, esta se transforma también en un *embedding*. A partir de este vector, se realiza una búsqueda en la base de datos para recuperar los fragmentos de texto más relevantes. Estos fragmentos son

enviados junto con la pregunta original al modelo de lenguaje, el cual genera una respuesta basada tanto en la consulta como en la información recuperada.

Este enfoque permite que el modelo responda de forma más precisa y contextualizada, incluso si la información solicitada no forma parte de su conocimiento previo.

5.2 Diseño de la arquitectura del sistema

A continuación, en la Figura 6, se presenta el diagrama de arquitectura, el cual sirve para comprender, a alto nivel, el funcionamiento y los componentes de la aplicación. Posteriormente se detalla cada elemento.

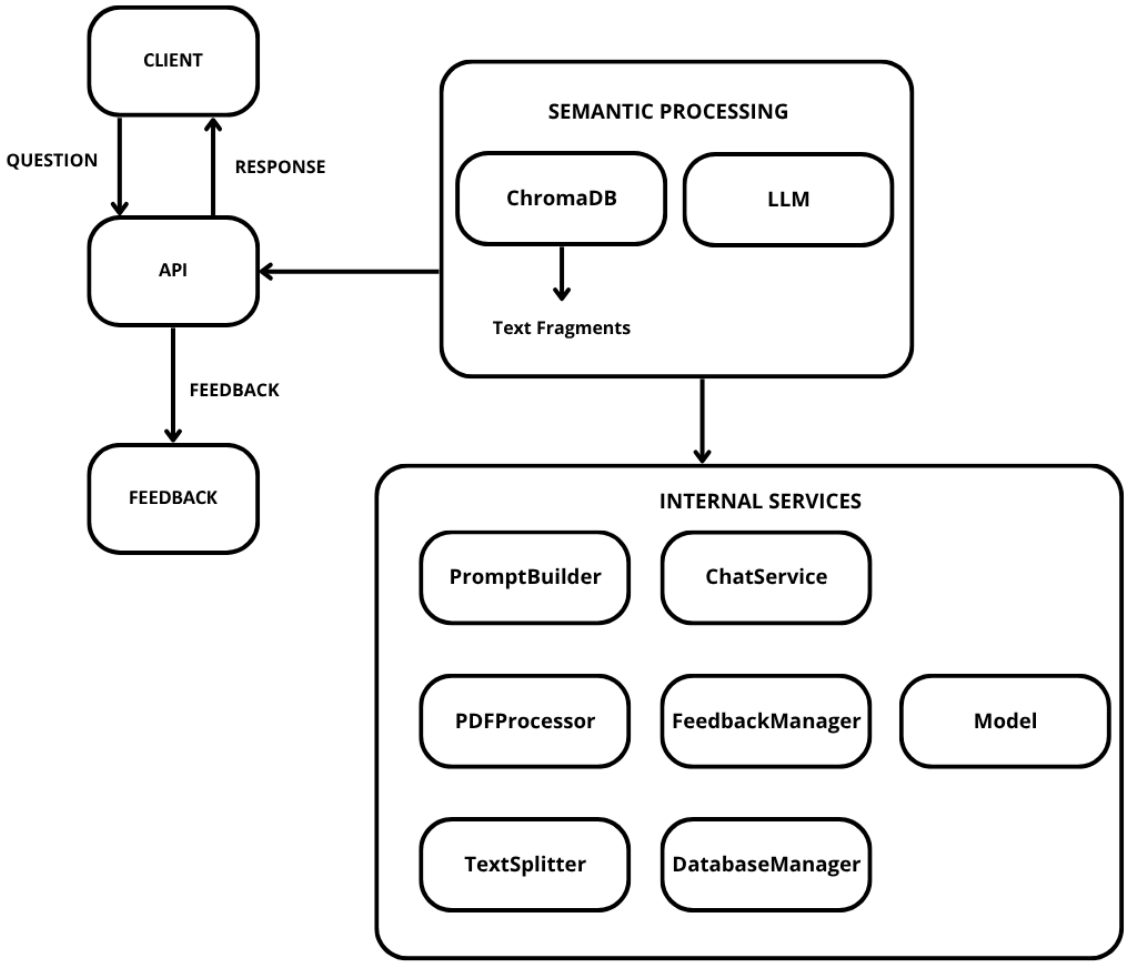


Figura 6 Diagrama de arquitectura

Capa de Cliente
Esta capa corresponde al usuario final, que interactúa con el sistema mediante una interfaz o cliente. Su principal responsabilidad es enviar preguntas y recibir respuestas.

- Entrada: preguntas o solicitudes de información adicional.

- Salida: respuestas generadas, enriquecidas o corregidas.

Capa de API
Es el punto de entrada al sistema. Esta capa recibe las solicitudes del cliente y coordina los módulos internos para dar una respuesta adecuada.

- Gestiona el flujo de mensajes entre los componentes.

- Invoca la búsqueda de *embeddings* y la generación de respuestas.

- Determina si es necesaria más información y solicita su recuperación.

Capa de Procesamiento Semántico
Aquí se encuentran los módulos encargados de buscar información relevante y generar respuestas.

- ChromaDB: almacena y permite la búsqueda semántica de documentos a través de *embeddings*.

- LLM: genera respuestas a partir del contenido recuperado. Puede reestructurar, resumir o elaborar respuestas complejas.

Capa de Servicios Internos
Esta capa agrupa los componentes de lógica interna y transformación de datos, como se representa en el diagrama de clases.

- DatabaseManager: gestiona la interacción con ChromaDB y otras bases de datos. Permite consultar, limpiar y extraer información de forma estructurada.

- PDFProcessor y TextSplitter: procesan los documentos PDF, dividiéndolos en fragmentos y extrayendo texto útil para generar *embeddings*.

- PromptBuilder: construye el *prompt* adecuado para enviar al modelo LLM. Se encarga de incorporar contexto, *feedback* y detalles específicos.

- FeedbackManager: guarda la retroalimentación del usuario, permitiendo ajustar futuras respuestas.

- ChatService: Maneja todo el proceso de consulta, ya sea una pregunta inicial o una solicitud de respaldo. Es la interfaz entre la capa API y los componentes internos.

- Model: representa el modelo de lenguaje que genera la respuesta textual final a partir del *prompt* creado.

Para comprender el funcionamiento del *chatbot*, en la Figura 7 se nos muestra de forma general mediante un diagrama de secuencia, indicando paso por paso el completo funcionamiento.

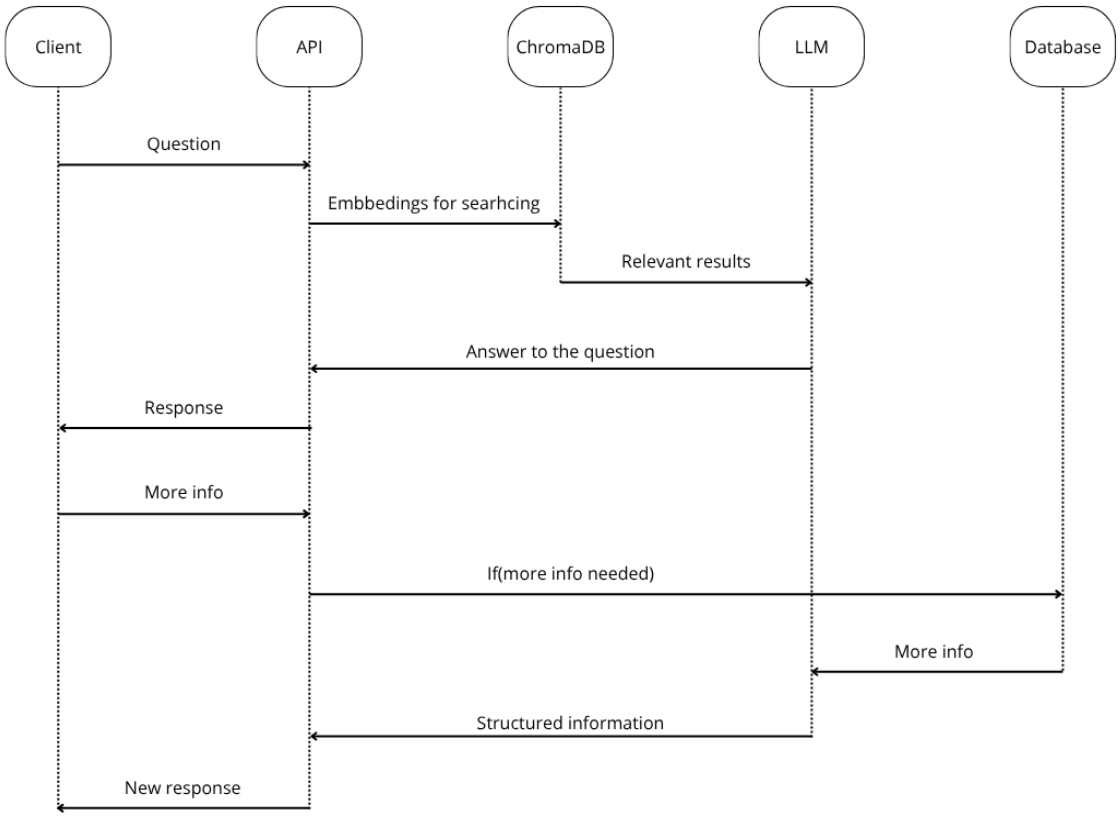


Figura 7 Diagrama de secuencia

1. **El cliente hace una pregunta**
   El usuario (Client) envía una pregunta a través de la aplicación. Esta petición es recibida por la API, que actúa como punto de entrada al sistema.

2. **La API convierte la pregunta en *embeddings* y los utiliza para buscar información**
   La API transforma la pregunta en *embeddings*, que son representaciones vectoriales del contenido, y los envía a ChromaDB, que es una base de datos especializada en búsquedas semánticas.

3. **ChromaDB devuelve los resultados más relevantes**
   ChromaDB compara los *embeddings* de la pregunta con los almacenados y responde con los fragmentos más relevantes.

4. **EL LLM genera una respuesta y se la devuelve a la API**
   El LLM recibe las respuestas más relevantes junto al *prompt*, con lo que elabora una respuesta que será devuelta a la API.

5. **La API devuelve la respuesta al cliente**
   Una vez la API obtiene la respuesta, se la envía al usuario.

6. **El cliente puede pedir más información**
   Si el usuario considera que la respuesta no es suficiente, puede hacer otra petición para solicitar más información.

7. **La API verifica si es necesario consultar más información**
   La API verifica si el usuario solicita una consulta adicional y vuelve a utilizar el LLM, el cual elabora la consulta.

8. **El LLM consulta directamente a la base de datos**
   Después de que el LLM elabore la *query*, se realiza la consulta.

9. **La base de datos devuelve la información**
   La base de datos devuelve la información obtenida a través de la consulta.

10. **El LLM construye una nueva respuesta más completa**
    Una vez obtenida la información, el LLM genera una respuesta en base a los resultados obtenidos de la consulta.

11. **La API envía esta nueva respuesta al cliente**
    Finalmente, la API devuelve la nueva respuesta al cliente.

5.3 Implementación y despliegue del *chatbot*

5.3.1 Implementación

El *chatbot* ha sido desarrollado íntegramente en un único archivo .py, con una estructura similar a un cuaderno de *Jupyter*, sin emplear clases ni separación por módulos. Esto ha facilitado una rápida iteración y prueba de componentes individuales, como el preprocesamiento de textos, el sistema de *embeddings* o la generación de respuestas.

El desarrollo ha seguido una arquitectura basada en RAG, donde los documentos PDF proporcionados por la empresa son procesados y fragmentados utilizando la herramienta *RecursiveCharacterTextSplitter* de LangChain. Estos fragmentos se convierten en vectores mediante el modelo BAAI/bge-m3 y se almacenan en una base de datos vectorial persistente usando ChromaDB.

Para la generación de respuestas se ha empleado un modelo LLM (mistralai/Mistral-Nemo-Instruct-2407), descargado desde Hugging Face. La interacción con el modelo se realiza mediante la librería *transformers*, utilizándose *AutoTokenizer* y AutoModelForCausalLM para la *tokenización* y generación.

Además del motor de búsqueda semántica, el sistema incorpora la capacidad de ejecutar consultas SQL sobre una base de datos relacional externa, filtrando resultados por ciertos parámetros propios de la empresa, y reformulando la información obtenida con el modelo LLM.

El *backend* del *chatbot* se sirve a través de una API REST desarrollada con FastAPI, que incluye configuraciones de CORS para facilitar su integración con posibles interfaces web o aplicaciones de terceros.

5.3.2 Propuesta de diseño orientado a clases

Como se ha comentado anteriormente, el desarrollo original del sistema se realizó utilizando un cuaderno de *Jupyter*, donde la lógica está estructurada en celdas secuenciales sin una separación estricta de responsabilidades. Si bien este enfoque puede ser ágil durante la fase inicial de prototipado, presenta importantes limitaciones en cuanto a escalabilidad, mantenimiento y reutilización del código.

Con el objetivo de mejorar la organización y favorecer la facilidad y claridad a la hora de realizar posibles transiciones a entornos más robustos, se propone una estructura modular basada en clases. En la figura 8 se presenta un diagrama que resume la propuesta.

Descripción del diseño

A continuación, se describen brevemente las funciones principales de cada clase:

- ChatService: Es el punto de entrada al sistema. Se encarga de hacer las llamadas a los métodos de las clases necesarias. Recibe preguntas del usuario y coordina tanto la generación de la respuesta como el guardado del *feedback*.
- Model: Encapsula el modelo de lenguaje. Su método principal (generate_response) genera las respuestas en función del *prompt* recibido.
- PromptBuilder: Construye los *prompts* que se envían al modelo. Puede incluir contexto, preguntas anteriores, *feedback* o inclusive *shcemas* de la base de datos.
- DatabaseManager: Gestiona el acceso y las consultas a la base de datos, incluyendo la obtención del *schema*, ejecución de *queries* a chromaDB y extracción de columnas.
- PDFProcesor y TextSplitter: Juntos permiten procesar los documentos PDF y dividir el contenido en fragmentos más manejables.
- FeedbackManager: Guarda las valoraciones de los usuarios sobre las respuestas generadas.

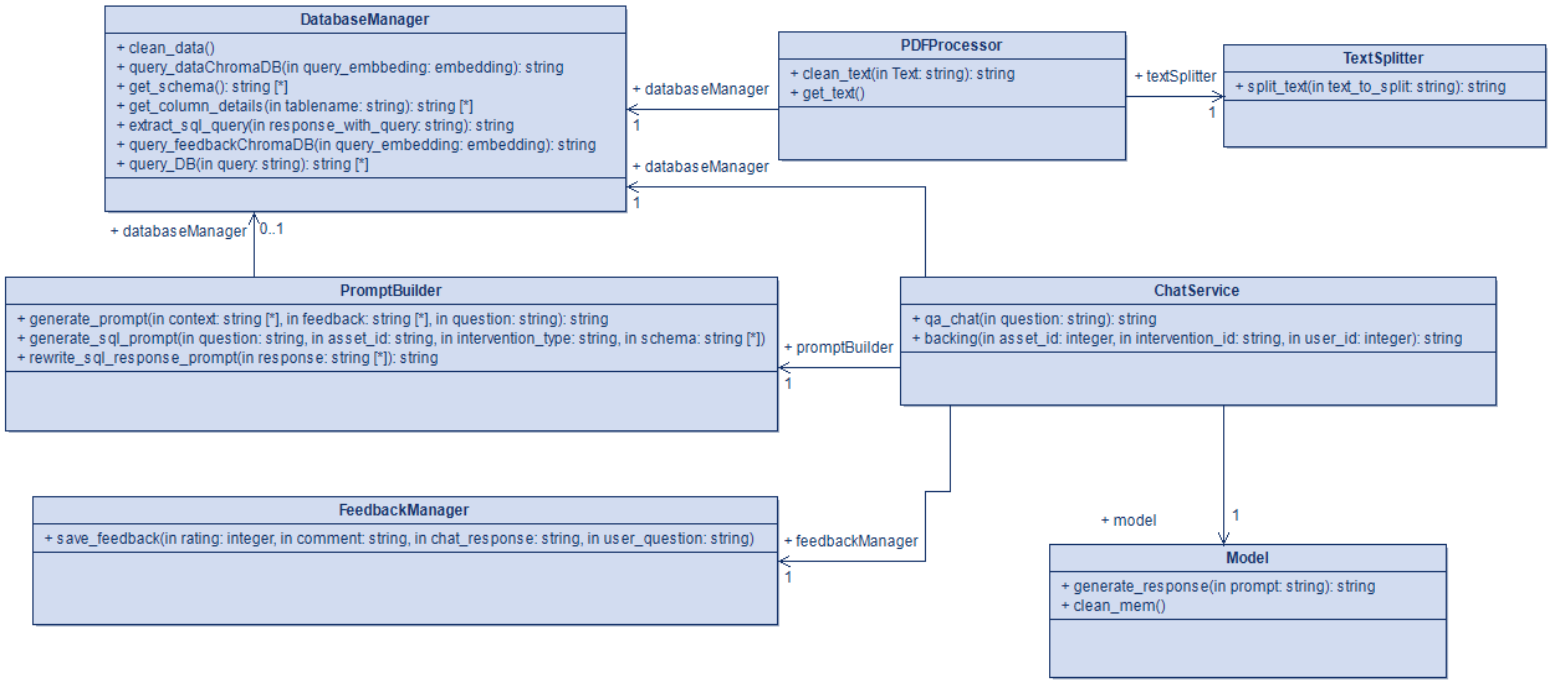


Figura 8 Diagrama de clases

5.3.3 Despliegue

El despliegue del sistema ha sido realizado en una instancia virtual de GCE, la cual ha sido configurada con GPUs compatibles y con las características necesarias para permitir la ejecución del LLM de forma eficiente. La elección de este servicio (GCE) se debe a su flexibilidad, escalabilidad y la posibilidad de acceder a recursos especializados, como pueden ser GPUs de alta gama. En este caso, la máquina virtual consta de dos graficas Nvidia L4 [12], las cuales permiten alojar y ejecutar el modelo sin problema. A esto se le añade la necesidad de descargar los drivers necesarios para el uso de dichas gráficas.

Al tener dos tarjetas gráficas, surgió el problema de que el modelo intentaba se intentaba ejecutar en solo en una de ellas, lo cual no es posible. La solución a esta problemática fue el uso de CUDA (Compute Unified Device Architecture), que además acelera ligeramente las tareas de alto procesamiento, reduciendo el tiempo de cálculo.

La aplicación FastAPI se ejecuta utilizando uvicorn, levantando el servidor en un puerto (8000 en este caso), exponiéndose al exterior mediante las reglas de firewall establecidas en la instancia GCE, permitiendo las conexiones http.

Este enfoque permite escalar el servicio fácilmente en caso de ser necesario, ya sea aumentando la potencia de la instancia o incorporando balanceadores de carga en caso de múltiples usuarios concurrentes.

6. Análisis, resultados y pruebas

En este capítulo se analizan los resultados obtenidos tras el desarrollo del sistema y se detallan las pruebas realizadas para evaluar su funcionamiento. Se describe cómo se

validó el comportamiento del *chatbot* en condiciones reales, incluyendo casos de uso representativos y pruebas de aceptación. Asimismo, se identifican las principales limitaciones detectadas durante la validación y se valora el grado de cumplimiento de los requisitos definidos previamente. El objetivo de este análisis es ofrecer una visión crítica del rendimiento del sistema y establecer una base sólida para posibles mejoras futuras

6.1 Funcionamiento del sistema

Como se ha descrito anteriormente, el objetivo del sistema desarrollado es ofrecer a los operarios de planta industrial una herramienta conversacional capaz de responder a consultas técnicas relacionadas con el funcionamiento de maquinaria y datos específicos de producción. Para ello, se ha implementado un asistente utilizando diferentes técnicas que permiten generar respuestas precisas y contextualizadas a partir de fuentes documentales internas y bases de datos estructuradas.

El flujo de funcionamiento general del sistema puede dividirse en las siguientes etapas:

1. Procesamiento y almacenamiento de documentos
Inicialmente, los manuales técnicos en formato PDF son sometidos a un proceso de preprocesamiento. Este incluye la división de los documentos en fragmentos o splits, cada uno con un tamaño controlado para asegurar una buena semántica a la hora de generar los *embeddings*. A cada fragmento se le añaden metadatos relevantes, como el nombre del documento original y el número de página correspondiente, lo cual facilita la trazabilidad de las respuestas generadas.
Una vez divididos, los fragmentos son transformados en representaciones vectoriales mediante modelos de *embedding*, y almacenados en una base de datos vectorial, en este caso ChromaDB. Esta base de datos permite realizar búsquedas semánticas eficientes en tiempo de consulta.

2. Recepción de la consulta
El operario, a través de una interfaz (desarrollada por la empresa), introduce su consulta en lenguaje natural. Esta petición es enviada al *backend* del sistema, desarrollado en FastAPI y desplegado en GCE.

3. Recuperación de información relevante
Una vez recibida la consulta, se genera su representación vectorial utilizando el mismo modelo de *embeddings* utilizado en la fase de indexación. Con dicho vector, se consulta la base de datos vectorial para recuperar los fragmentos de texto más similares semánticamente a la pregunta original.
Estos fragmentos actúan como contexto para la generación de la respuesta. Además, si la consulta lo requiere, el sistema puede acceder a determinadas tablas específicas de la base de datos relacional interna de la empresa, permitiendo la integración de información dinámica y actualizada.

4. Generación de la respuesta
Los fragmentos recuperados, junto con la consulta original del usuario, son introducidos en un prompt diseñado específicamente para guiar la respuesta del modelo de lenguaje.

En este proyecto se utiliza un modelo LLM basado en Mistral, alojado y gestionado mediante el *framework* NeMo[9]. La respuesta generada se devuelve al operario a través de la interfaz, incluyendo metadatos que permiten identificar el origen de la información (documento y página).

5. Interfaz básica de pruebas

Con el fin de validar el funcionamiento del sistema, se ha desarrollado una interfaz web básica, que permite interactuar con el *chatbot* de manera directa. Esta interfaz no forma parte del producto final, ya que el *frontend* definitivo está siendo desarrollado por la empresa colaboradora, pero ha resultado esencial para la realización de pruebas, depuración y demostraciones.

6.2 Pruebas de aceptación

Para comprobar el cumplimiento de los requisitos se han realizado una serie de pruebas de aceptación:

Prueba de Aceptación 1 – Consulta de documentos PDF (RF1)

- Objetivo: Verificar que el *chatbot* puede responder correctamente a preguntas formuladas sobre información contenida en un PDF.
- Procedimiento: Se plantea la pregunta "¿Cuál es la capacidad del enfriador dorado?" y se comprueba que la respuesta proviene del PDF correspondiente.
- Criterio de aceptación: La respuesta debe contener al menos un fragmento extraído del PDF que describa el procedimiento y debe ser correcta.

Prueba de Aceptación 2 – Acceso a base de datos estructurada (RF2)

- Objetivo: Validar que el *chatbot* puede acceder a datos técnicos almacenados en una base de datos SQL.
- Procedimiento: El usuario solicita más información acerca de la pregunta
- Criterio de aceptación: El sistema debe devolver una respuesta que tenga coherencia con los datos almacenados.

Prueba de Aceptación 3 – Comprensión del lenguaje natural (RF3)

- Objetivo: Evaluar si el *chatbot* entiende expresiones informales o poco técnicas.
- Procedimiento: Se prueba con diferentes formas de preguntar lo mismo: "Tengo problemas en las horquillas, ¿Qué hago?", "Creo que tengo problemas con las horquillas", etc.
- Criterio de aceptación: El *chatbot* debe ofrecer respuestas útiles en todos los casos, mostrando comprensión semántica.

Prueba de Aceptación 4 – Respuestas precisas y contextualizadas (RF5)

- Objetivo: Confirmar que las respuestas generadas son útiles, coherentes y específicas.
- Procedimiento: Se formulan 10 preguntas sobre información técnica contenida en PDFs y base de datos.

[9] Para más información: https://www.nvidia.com/es-es/gpu-cloud/nemo-llm-service/

- Criterio de aceptación: Al menos 9 respuestas deben ser consideradas útiles por el usuario.

Prueba de Aceptación 5 – Tiempo de respuesta (RNF1)

- Objetivo: Comprobar que el sistema responde en menos de 5 segundos.
- Procedimiento: Se realiza un conjunto de 10 preguntas con cronómetro.
- Criterio de aceptación: El 90% de las respuestas debe llegar en menos de 5 segundos.

Prueba de Aceptación 6 – Uso en entorno sin internet (RNF7)

- Objetivo: Verificar que el sistema puede funcionar localmente sin conexión externa.
- Procedimiento: Se desactiva la red y se interactúa con el sistema desplegado en una máquina local.
- Criterio de aceptación: Las respuestas se generan correctamente usando los recursos locales.

Tras realizar las pruebas, se observa que la única prueba no satisfactoria es la prueba de aceptación 5, esto es debido a que el modelo tarda más tiempo en elaborar respuestas completas y correctas.

6.3 Presentación del prototipo en feria tecnológica y proyecto piloto

Un prototipo funcional del *chatbot* fue presentado en la feria Advanced Factories[10], celebrada en Barcelona entre los días 8 y 10 de abril. Este evento, centrado en la innovación, eficiencia y sostenibilidad del sector industrial, reunió a numerosas empresas y profesionales del ámbito tecnológico e industrial.

Durante la presentación, el sistema demostró sus capacidades clave, como la consulta de información técnica contenida en documentos PDF y la generación de respuestas mediante modelos de lenguaje. A pesar de que el desarrollo no estaba completamente finalizado, el prototipo ya contaba con la mayoría de las funcionalidades principales, lo que permitió mostrar su potencial en un entorno real y ante un público especializado.

La participación en la feria fue valorada como un éxito, ya que se recogieron opiniones positivas y se generó interés por parte de distintos asistentes, validando así la propuesta del proyecto en un contexto industrial práctico.

6.4 Resultados de las pruebas

Con el fin de validar el rendimiento y la eficacia del sistema desarrollado, se han realizado una serie de pruebas centradas en distintos aspectos clave de su funcionamiento. Estas pruebas permiten analizar cómo afectan determinadas decisiones de diseño a la calidad de las respuestas generadas por el chatbot, así como a su eficiencia en términos de tiempo de respuesta.

[10] Para más información: https://www.advancedfactories.com/en/

En concreto, las pruebas se han enfocado en tres dimensiones principales:

Evaluación de diferentes modelos de lenguaje: se han comparado varios modelos con capacidades distintas (tanto locales como alojados en la nube), analizando su capacidad de comprensión y generación de respuestas relevantes a partir del contexto proporcionado.

Análisis del tamaño de los fragmentos de texto (*splits*): se han probado distintos tamaños de división del texto extraído de los PDFs para observar cómo afecta esto tanto a la precisión de las respuestas como al rendimiento del sistema en la etapa de recuperación y procesamiento.

Medición de tiempos de respuesta: se han registrado los tiempos necesarios para cada etapa del proceso, incluyendo la búsqueda en la base de datos vectorial y la generación de la respuesta mediante la función generate. Se ha comparado el impacto de distintos modelos y configuraciones sobre el tiempo total de respuesta del sistema.

Además del rendimiento cuantitativo, también se ha realizado una evaluación cualitativa de las respuestas generadas, valorando su relevancia, claridad y utilidad.

Los resultados obtenidos ofrecen una visión global del comportamiento del sistema y sirven como base para justificar las decisiones finales adoptadas en cuanto a arquitectura, modelo de lenguaje, y configuración del procesamiento de los datos.

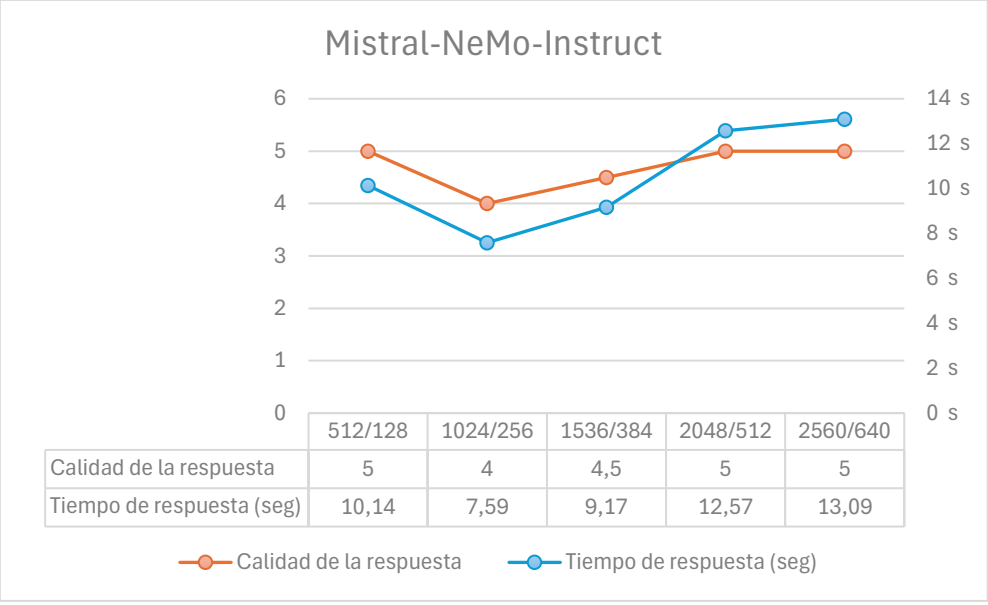


Figura 9 Análisis rendimiento Mistral-NeMo

En la figura 9 se observa el rendimiento del modelo que ha sido seleccionado para la implementación final del sistema, ya que presenta un equilibrio adecuado entre tiempo de respuesta y calidad de las respuestas generadas. A lo largo de las pruebas realizadas, ha demostrado ser capaz de ofrecer respuestas precisas y relevantes en un tiempo

razonable, especialmente en el caso de preguntas frecuentes o de complejidad media, donde los tiempos de respuesta se mantienen bajos y estables.

Si bien es cierto que en consultas más complejas o menos comunes los tiempos pueden incrementarse notablemente (alcanzando en algunos casos entre 20 y 25 segundos), este comportamiento se considera aceptable dentro del contexto del uso previsto, ya que dichas situaciones representan un porcentaje reducido de las interacciones esperadas. En general, el modelo responde de manera eficaz, convirtiéndose en una opción sólida y viable para su integración en el entorno industrial de consulta técnica.

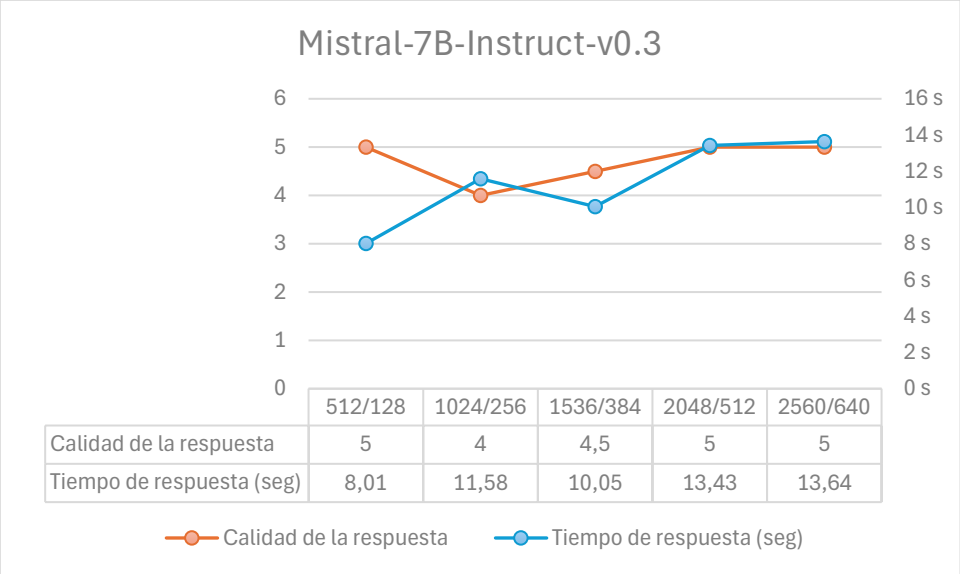


Figura 10 Análisis rendimiento Mistral-7B

Otro de los modelos considerados como candidato fue Mistral-7B-Instruct, observable en la figura 10, que, al igual que el modelo finalmente seleccionado, ofreció un equilibrio razonable entre calidad de respuesta y tiempo de ejecución. Durante las pruebas, demostró una buena capacidad para generar respuestas coherentes y relevantes a partir del contexto proporcionado.

No obstante, la decisión final se inclinó hacia el otro modelo debido a su menor tiempo de respuesta, especialmente en aquellas preguntas más comunes o de menor complejidad. Aunque el rendimiento de Mistral-7B-Instruct era competitivo, las diferencias en los tiempos de respuesta, incluso en los casos más rápidos, resultaron determinantes. En un entorno donde la inmediatez puede marcar la diferencia en la experiencia del usuario, esta ventaja en velocidad fue clave para optar por el modelo seleccionado, pese a que implique un mayor consumo de recursos.

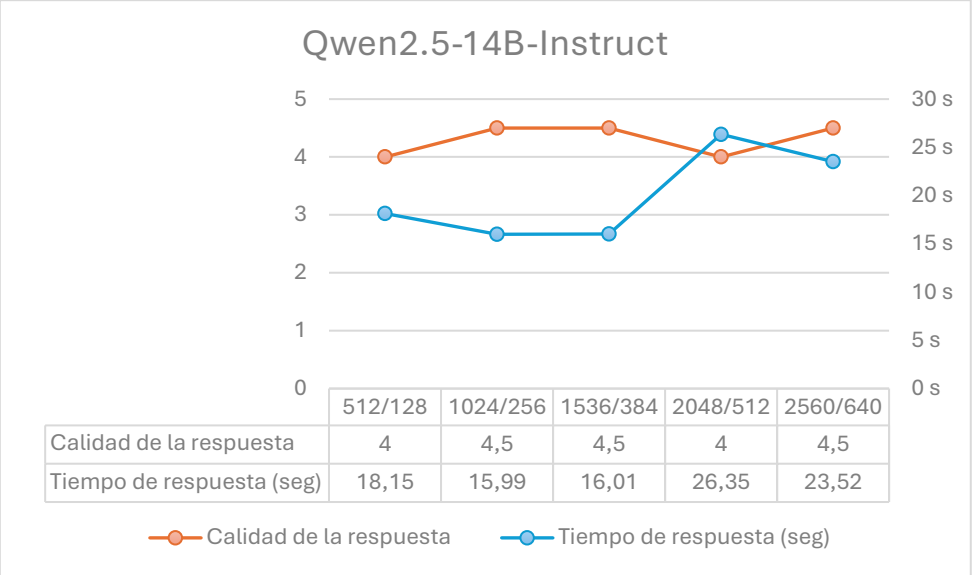


Figura 11 Análisis rendimiento Qwen2.5-14B

Este modelo, Qwen2.5-14B-Instruct, destacó por su alta calidad de respuesta, alcanzando niveles casi perfectos en términos de precisión y relevancia. Sin embargo, su principal limitación radica en los tiempos de respuesta, que, en comparación con los otros modelos evaluados, son considerablemente más lentos. Dado que una de las prioridades del sistema es lograr respuestas rápidas, este modelo no fue seleccionado.

De no ser por la preocupación sobre los tiempos de procesamiento, es probable que hubiera sido la opción preferida debido a su excepcional rendimiento en cuanto a calidad de las respuestas.

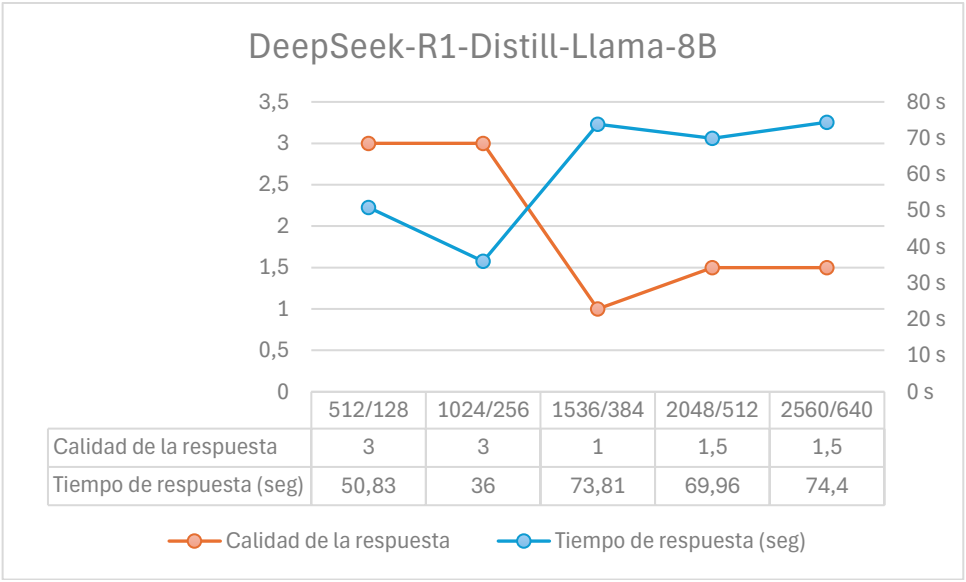
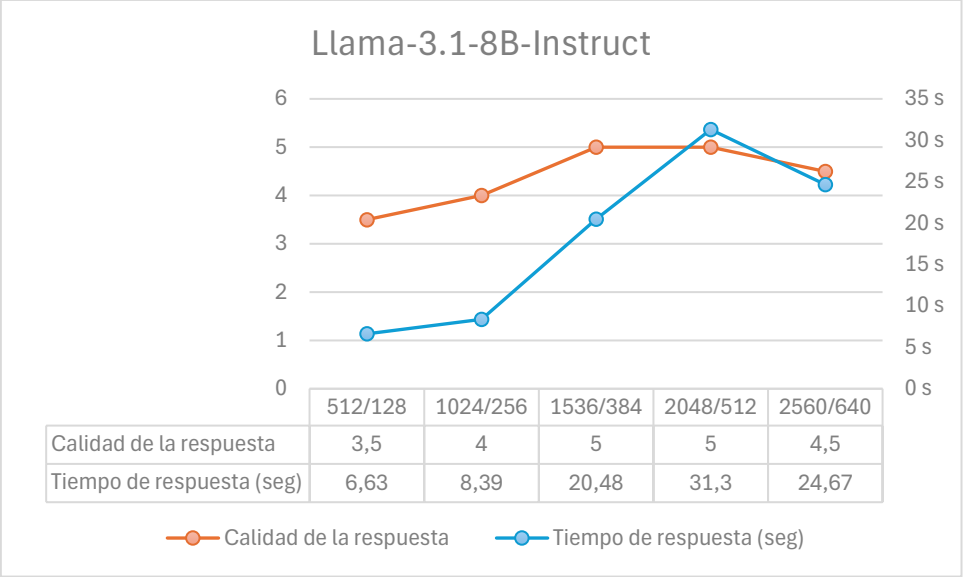


Figura 12 Análisis rendimiento DeepSeek-R1-Llama-8B

En la figura 12, el modelo evaluado fue DeepSeek-R1-Distill-Llama-8B, una versión distilada[11] basada en uno de los modelos previamente probados. Si bien presenta una arquitectura optimizada, durante las pruebas se observó un comportamiento particular: el modelo tiende a elaborar extensamente sus respuestas, realizando un razonamiento detallado incluso para preguntas simples.

Como consecuencia, los tiempos de respuesta se incrementan considerablemente, y las respuestas generadas suelen ser excesivamente largas y, en algunos casos, innecesariamente complejas para el tipo de consulta esperada por los operarios. Esta tendencia a sobre-explicar cada respuesta lo convierte en una opción menos adecuada para un entorno donde se priorizan respuestas concisas, claras y rápidas.



	512/128	1024/256	1536/384	2048/512	2560/640
Calidad de la respuesta	3,5	4	5	5	4,5
Tiempo de respuesta (seg)	6,63	8,39	20,48	31,3	24,67

Figura 13 Llama-3.1-8B-Instruct

En la comparación entre modelos de tamaño similar, destaca el caso de LLaMA-3.1-8B (Figura 13) frente a Mistral-7B. Aunque ambos presentan características técnicas comparables, las respuestas generadas por LLaMA-3.1-8B fueron ligeramente inferiores en cuanto a calidad y mostraron una mayor latencia. Este desequilibrio en la relación calidad-tiempo fue determinante a la hora de tomar la decisión final, optándose por Mistral-7B como modelo preferente, al ofrecer un rendimiento más sólido y eficiente dentro de su categoría.

[11] Knowledge distillation is a machine learning technique that aims to transfer the learnings of a large pre-trained model, the “teacher model,” to a smaller “student model.”(IBM, 2024)

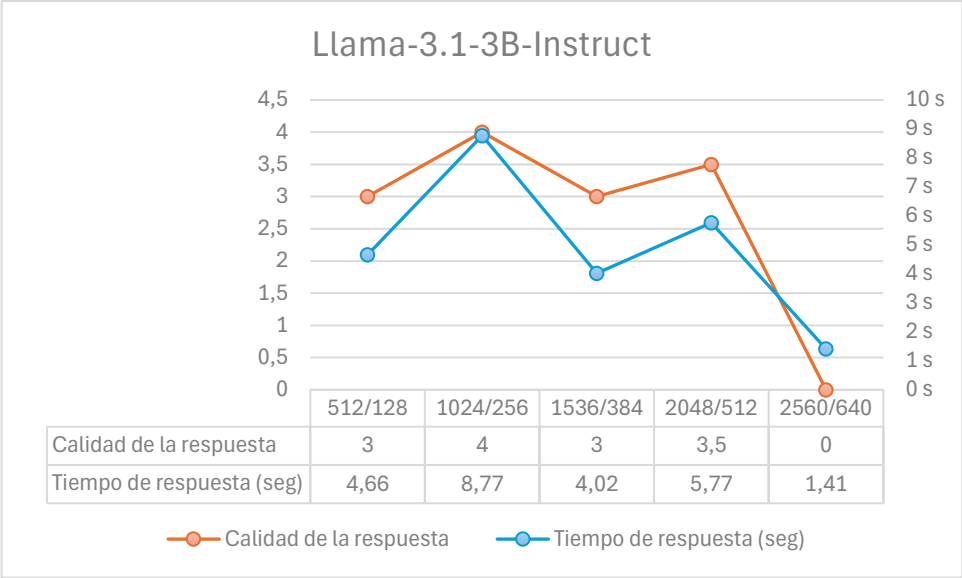


Figura 14 Llama-3.1-3B-Instruct

Por último, se analiza el modelo Llama-3.1-3B, representado en la figura 14. Al tratarse del modelo más pequeño entre los evaluados, presenta limitaciones evidentes en su capacidad de procesamiento, lo que afecta directamente a la calidad de las respuestas generadas. En situaciones donde se le proporciona una cantidad elevada de información, el modelo puede omitir partes relevantes del texto o incluso no generar respuesta alguna, al no ser capaz de gestionar correctamente toda la entrada.

A pesar de estas limitaciones, su rendimiento es aceptable dentro de su categoría, mostrando una eficiencia razonable si se lo compara con otros modelos de tamaño similar. No obstante, no puede competir en calidad ni en consistencia con modelos de mayor capacidad, que, aunque más exigentes en recursos, ofrecen resultados notablemente superiores.

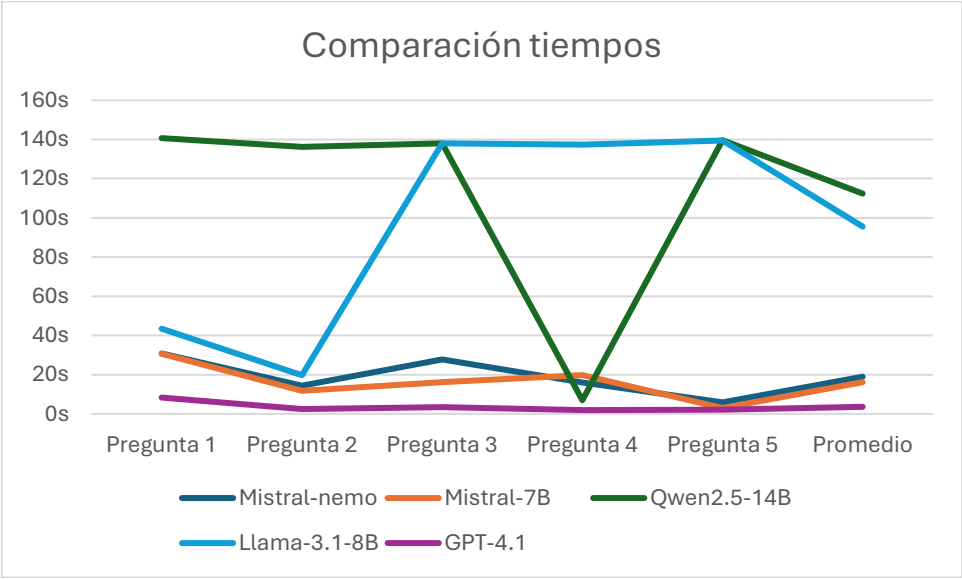


Figura 15 Comparación de tiempos de respuesta

En la figura 15 se puede observar los tiempos de respuesta a diferentes cuestiones con diferentes tipos de exigencias. Por simplicidad, solo se han tomado muestras de aquellos modelos con mayor rendimiento. Añadido al resto de modelos probados, se ha implementado una versión con la API de OpenAI dado su bajo coste y alto rendimiento es una de las mejores opciones. El resto de los modelos no pueden compararse a este debido tanto a la diferencia en parámetros como a la cantidad de datos con los que han sido entrenados. Por otra parte, de los modelos restantes, destacan los Mistral, ambos con buenos resultados en lo que a tiempo respecta, pero, en cambio, la calidad de esta es la que marca la diferencia.

6.5 Limitaciones actuales

A pesar del desarrollo satisfactorio del sistema y los buenos resultados obtenidos en las pruebas, existen una serie de limitaciones que afectan directamente a su implementación práctica, escalabilidad y mantenimiento a medio y largo plazo.

6.5.1 Costes de infraestructura

Una de las principales limitaciones es el elevado coste de desplegar modelos de lenguaje de gran tamaño, especialmente en entornos industriales que requieren soluciones locales por motivos de privacidad, seguridad o disponibilidad. Ejecutar modelos en local exige equipamiento especializado, siendo imprescindible contar con tarjetas gráficas de alto rendimiento capaces de manejar grandes cantidades de parámetros en tiempos razonables. Esta infraestructura supone una inversión económica considerable que no todas las empresas están dispuestas o pueden asumir, especialmente las pequeñas y medianas.

En entornos en la nube, como GCE, aunque se elimina la necesidad de adquirir hardware propio, los costes son recurrentes y dependen del uso del sistema. Cuanto mayor sea la demanda o el número de usuarios simultáneos, mayor será el coste mensual, lo que complica la previsión presupuestaria y puede encarecer el sistema a largo plazo.

En general, los modelos de alto rendimiento disponibles actualmente no son accesibles económicamente para la mayoría de los entornos industriales sin un respaldo financiero sólido.

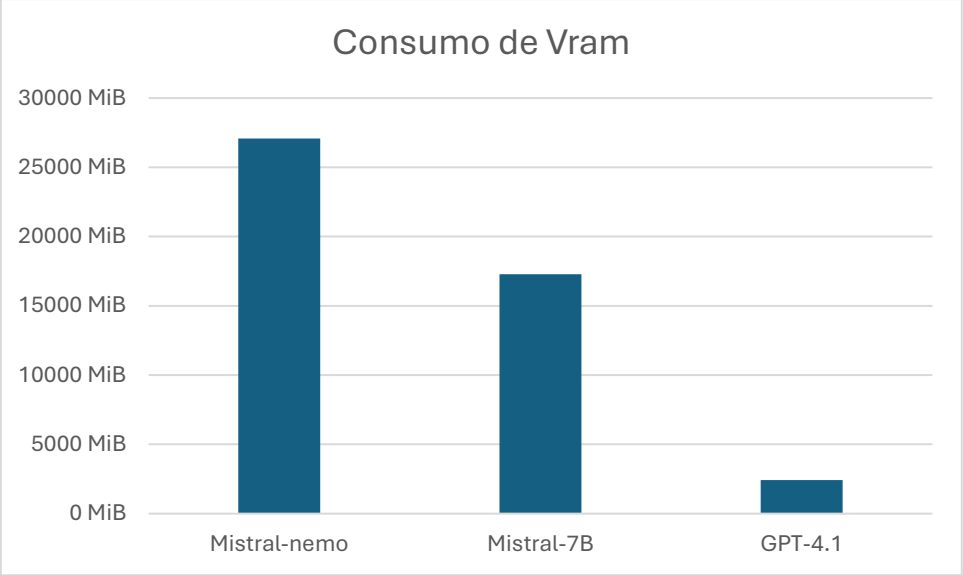


Figura 16 Consumo de los modelos

En la Figura 16 se presenta el consumo de tres modelos, cada uno seleccionado como representativo de un grupo según su cantidad aproximada de parámetros. Los grupos considerados son: modelos con alrededor de 14 mil millones de parámetros, modelos con aproximadamente 7 mil millones, y GPT-4.1, que se accede a través de la API de OpenAI. Este último muestra un consumo considerablemente menor, ya que no se ejecuta localmente, sino que los procesamientos se realizan mediante llamadas a la API.

6.5.2 Limitaciones del uso local

Aunque existen soluciones externas como la API de OpenAI, que permiten acceder a modelos de gran calidad a través de internet, esta opción queda descartada si se desea un despliegue completamente local. Muchas plantas industriales no permiten acceso a internet por razones de seguridad o confidencialidad, lo que limita el uso de servicios cloud o APIs externas. En este contexto, las restricciones de conectividad imponen una fuerte limitación técnica, obligando a que todo el sistema, incluidos los modelos LLM, se ejecuten en máquinas locales.

6.5.3 Complejidad en la evaluación del feedback

Otro aspecto crítico es la dificultad para evaluar automáticamente el feedback de los operarios. El sistema implementa un enfoque de tipo RAG, lo que implica que las respuestas generadas dependen tanto de los documentos recuperados como del modelo generador. Esto puede dar lugar a respuestas complejas o ambiguas, que pueden ser interpretadas de forma distinta por diferentes usuarios.

En situaciones en las que dos operarios emiten evaluaciones opuestas sobre una misma respuesta, el sistema no tiene capacidad para determinar de forma autónoma cuál de las valoraciones es correcta. La única forma de validar este tipo de conflictos es

mediante revisión manual, lo cual no es escalable y supone una carga operativa que difícilmente puede mantenerse de forma continuada en un entorno real.

6.5.4 Rendimiento y tiempos de respuesta

Aunque los modelos utilizados ofrecen resultados satisfactorios, el rendimiento en términos de tiempo de respuesta no puede igualar al de soluciones comerciales más avanzadas y optimizadas. Se ha trabajado en distintas estrategias de mejora, incluyendo:

- Ajustes de los parámetros del sistema.
- Técnicas de prompt engineering.
- Optimización del sistema de recuperación de información (RAG).
- Pruebas con distintos tamaños de split de los documentos y configuraciones de embedding.

A pesar de estas optimizaciones, no ha sido posible reducir significativamente los tiempos sin comprometer la calidad de las respuestas. Aunque el sistema ofrece tiempos aceptables para la mayoría de los casos, en preguntas complejas puede superar los 25-30 segundos, lo que podría afectar la experiencia del usuario si no se gestiona adecuadamente.

6.5.5 Imposibilidad de aplicar *fine-tuning*

Durante el desarrollo se valoró la posibilidad de aplicar *fine-tuning* a los modelos utilizados, lo que permitiría mejorar su rendimiento en tareas concretas relacionadas con el dominio industrial específico de la fábrica. Sin embargo, el entrenamiento personalizado de modelos LLM requiere recursos computacionales significativos y una cantidad considerable de datos etiquetados, además de tiempo y experiencia técnica.
El coste económico de llevar a cabo este proceso es elevado y supera el presupuesto disponible para este proyecto, lo que hizo que esta alternativa fuera descartada. A corto plazo, la única opción viable es seguir utilizando modelos pre entrenados y optimizar su uso mediante técnicas ligeras como *prompt engineering* y *retrieval tuning.*

7. Conclusiones y líneas de trabajo a futuro

El desarrollo del *chatbot* presentado en este trabajo ha sido capaz de cumplir todos los objetivos propuestos.

A lo largo de la investigación, las pruebas y el testeo realizados, se ha llegado a la conclusión de que desarrollar un *chatbot* que funcione completamente de forma local con un rendimiento competitivo implica costes económicos elevados y presenta serias limitaciones técnicas. Por un coste significativamente menor, tanto en términos de hardware como de software, es posible acceder a soluciones no locales que ofrecen un rendimiento mucho más elevado, sin comprometer necesariamente la privacidad de los datos. Sin embargo, esta solución no siempre es viable para empresas que manejan información sensible y que requieren garantizar la confidencialidad de sus datos, lo cual puede conllevar a descartar estas opciones.

Esta problemática plantea un desafío clave: lograr un equilibrio entre rendimiento, privacidad y coste.

A partir de este punto, las líneas de trabajo futuras que se proponen son las siguientes:

- **Evaluar e incorporar modelos más potentes**: Analizar nuevas arquitecturas o versiones de modelos de lenguaje que, si bien pueden tener un coste mayor, proporcionen una mejora significativa en el rendimiento general del *chatbot*.

- **Mejorar sistema de *feedback***: Actualmente, el modelo es capaz de almacenar y utilizar las reseñas de los usuarios. Uno de los objetivos siguientes sería desarrollar un sistema capaz de filtrar reseñas no validas o aquellas que se contradigan, además de requerir un cierto número de reseñas para que estas se tengan en cuenta.

- **Realizar pruebas piloto en entornos reales**: Implementar el *chatbot* en empresas seleccionadas a modo de prueba piloto, identificando patrones comunes y necesidades específicas, lo que permitirá ajustar funcionalidades clave antes de una posible implementación más amplia.

- **Explorar soluciones híbridas**: Investigar el uso de modelos locales para el tratamiento de información sensible, combinados con servicios en la nube para consultas generales, con el objetivo de mantener la privacidad sin sacrificar el rendimiento.

En conclusión, aunque el proyecto ha sido completado con éxito, todavía hay aspectos en los que se puede continuar desarrollando un sistema para que sea más robusto, adaptable y eficiente. La comprensión de los desafíos técnicos, económicos y éticos asociados al uso de IAs en contextos empresariales ha sido clave para trazar una hoja de ruta realista y con potencial impacto a futuro.

La realización de este TFG me ha permitido poner en práctica los conocimientos adquiridos durante el grado. Además, me ha permitido introducirme en tecnologías punteras, como lo pueden ser GCE, modelos LLM, además de herramientas ampliamente utilizadas en los entornos profesionales como Docker o Kubernetes. También he trabajado con bases de datos MySQL y PostgreSQL, y *frameworks* de desarrollo como Flask o FastAPI. Todo ello me ha otorgado una visión más realista y actual del ambiente en el que trabajan las empresas hoy en día.

8. Bibliografía

[1]. Chen, L. (2021, July 16). *Intro to natural language processing – how to encode meaning of a word?* Towards Data Science. https://towardsdatascience.com/intro-to-natural-language-processing-how-to-encode-meaning-of-a-word-b1742d4beca2/

[2]. Cornejo, P. (2019, junio 7). *Arquitectura de un chatbot*. Medium. https://medium.com/@patcornejo/arquitectura-de-un-chatbot-cb2d1c5f86c7

[3]. Datacamp. (2023). *What is BERT? An intro to BERT models*. https://www.datacamp.com/es/blog/what-is-bert-an-intro-to-bert-models

[4]. GeeksforGeeks (2025) Software engineering: prototyping model. GeeksforGeeks. https://www.geeksforgeeks.org/software-engineering-prototyping-model/

[5]. Google Cloud. (s.f.). *Productos de computación: máquinas virtuales para cualquier carga de trabajo*. https://cloud.google.com/products/compute#virtual-machines-for-any-workload

[6]. IBM (2025) Knowledge distillation. IBM Think. https://www.ibm.com/think/topics/knowledge-distillation

[7]. IBM Research. (2023, 22 de agosto). *What is retrieval-augmented generation (RAG)?*. IBM Research. https://research.ibm.com/blog/retrieval-augmented-generation-RAG

[8]. Kiangala KS, Wang Z (2024) An experimental hybrid customized AI and generative AI chatbot human machine interface to improve a factory troubleshooting downtime in the context of Industry 5.0. *Int J Adv Manuf Technol* 132(5):2715–2733. https://doi.org/10.1007/s00170-024-13492-0

[9]. Lasi, H., Fettke, P., Kemper, H.-G., Feld, T., & Hoffmann, M. (2014). Industry 4.0. *Business & Information Systems Engineering, 6*(4), 239–242. https://doi.org/10.1007/s12599-014-0334-4

[10]. Mehta, A. (2024). *Artificial intelligence chatbots and sustainable supply chain optimization in manufacturing: Examining the role of transparency, innovativeness, and Industry 4.0 advancements*. ResearchGate. https://www.researchgate.net/publication/387131680

[11]. Mleczko, K. (2021). *Chatbot as a tool for knowledge sharing in the maintenance and repair processes*. Multidisciplinary Aspects of Production Engineering, 4(1), 499–508. https://www.researchgate.net/publication/355177200_Chatbot_as_a_Tool_for_Knowledge_Sharing_in_the_Maintenance_and_Repair_Processes

[12]. NVIDIA. (s.f.). *GPU NVIDIA L4 Tensor Core para centros de datos.* Recuperado el 14 de mayo de 2025, de https://www.nvidia.com/es-es/data-center/l4/

[13]. Refactoring.fm. (2023). *Meeting buffers, naming files, and how not to be boring*. https://refactoring.fm/p/meeting-buffers-naming-files-and

[14]. Soydaner, D. (2022). *Attention mechanism in neural networks: Where it comes and where it goes*. Neural Computing and Applications, 34, 13371–13385. https://doi.org/10.1007/s00521-022-07366-3

[15]. Tang, T., Lu, K., Song, X., Zhang, X., Gong, Z., Zhao, Z., ... & He, X. (2023). *A survey of large language models*. ResearchGate. https://www.researchgate.net/publication/369740832_A_Survey_of_Large_Language_Models

[16]. Towards Data Science. (s.f.). *Intro to Natural Language Processing: How to encode meaning of a word* [Imagen]. https://towardsdatascience.com/intro-to-natural-language-processing-how-to-encode-meaning-of-a-word-b1742d4beca2/

[17]. Tunstall, L., von Werra, L., & Wolf, T. (2022). *Natural Language Processing with Transformers: Building Language Applications with Hugging Face* (Revised ed.). O'Reilly Media.

[18]. Wolters Kluwer. (s.f.). How natural language processing impacts professions. https://www.wolterskluwer.com/en/expert-insights/how-natural-language-processing-impacts-professions

[19]. Yang, Y., Yang, H., Xu, Y., & Xie, L. (2024). *Foundation models and the risk of AI-generated hallucinations: A survey and opinion*. Patterns, 5(4), 100978. https://www.sciencedirect.com/science/article/pii/S266729522400014X]

[20]. Amazon Web Services. (s.f.). *¿Qué son los modelos autoregresivos?* https://aws.amazon.com/es/what-is/autoregressive-models/