



CARRERA DE ESPECIALIZACIÓN EN INTELIGENCIA ARTIFICIAL

MEMORIA DEL TRABAJO FINAL

Desarrollo y personalización de un chatbot de inteligencia artificial basado en un modelo largo de lenguaje

Autor:

Esp. Ing. Jorge Hernán Cuenca Marín

Director:

Dr. Ing. Camilo Torres (Unillanos)

Jurados:

Mg. Ing. Monica Silva Quiceno (Unillanos)

Mg. Ing. Omar Yesid Beltran Gutierrez (Unillanos)

Esp. Diego Alejandro Asencio Cuellar (Unillanos)

*Este trabajo fue realizado en la ciudad de Villavicencio, Colombia,
entre septiembre de 2023 y octubre de 2024.*

Resumen

El trabajo se centra en el desarrollo y despliegue de un chatbot de inteligencia artificial optimizado para la empresa BITLINK, diseñado para mejorar la interacción entre la empresa y sus usuarios mediante un sistema de respuesta automática. Este chatbot es capaz de comprender y generar lenguaje natural de manera coherente y relevante, lo que representa un avance tecnológico significativo para BITLINK, al ubicarse en la vanguardia de la innovación tecnológica en la interacción con clientes.

El desarrollo del trabajo se ha sustentado en un profundo conocimiento de áreas clave, como la gestión de proyectos y el procesamiento de lenguaje natural, que han sido esenciales para alcanzar los objetivos propuestos.

Agradecimientos

Agradezco a mi director, Dr. Esp. Ing. Camilo Torres y al equipo de BITLINK por su apoyo y colaboración en este trabajo. También agradezco a mi familia y amigos por su constante apoyo.

Índice general

Resumen	I
1. Introducción general	1
1.1. BITLINK y la necesidad de la inteligencia artificial	1
1.2. Historia de los chatbots	2
1.3. Conceptos generales	2
1.4. Estado del arte	3
1.5. Motivación	4
1.6. Objetivos y alcance	5
2. Introducción específica	7
2.1. Breve descripción de la implementación	7
2.2. Tecnologías utilizadas	8
2.2.1. Modelos de lenguaje grande (LLM)	8
2.2.2. Frameworks para el frontend	8
2.2.3. Frameworks para el backend	8
2.2.4. Plataformas de computación en la nube	9
2.2.5. Bibliotecas de procesamiento del lenguaje natural	9
2.3. <i>Retrieval-Augmented Generation</i> (RAG)	9
2.4. <i>Fine-tuning</i> del modelo de lenguaje	10
3. Diseño e implementación	11
3.1. Diseño del sistema	11
3.2. Implementación del frontend	14
3.3. Implementación del backend	16
3.3.1. <i>Endpoint</i> /send_message/	16
3.3.2. <i>Endpoint</i> /best_answer/	17
3.3.3. <i>Endpoint</i> /get_best_answers/	17
3.3.4. <i>Endpoint</i> /upload_pdf/	18
3.3.5. <i>Endpoint</i> /add_chat/	19
3.3.6. Seguridad del sitio administrativo	21
3.4. Integración del modelo LLM	22
3.5. <i>Fine-tuning</i> del modelo	23
3.6. Ajustes y optimización	24
3.6.1. Arquitectura de mejora continua con RAG	25
4. Ensayos y resultados	27
4.1. Metodología de ensayos	27
4.1.1. Pruebas funcionales	27
4.1.2. Pruebas de integración	29
4.1.3. Configuración del entorno de pruebas	29
4.1.4. Interfaz administrativa	30
4.1.5. Actualización del RAG	32

4.1.6. Pruebas de usabilidad	33
4.2. Resultados obtenidos	33
4.2.1. Evaluación cuantitativa del desempeño	33
4.2.2. Feedback de usuarios	33
4.3. Análisis de resultados	34
4.4. Comparación con el estado del arte	34
4.4.1. Resultado final del sistema optimizado	35
4.5. Validación del RAG y Evolución del Corpus	37
5. Conclusiones	39
5.1. Aportes principales del proyecto	39
5.2. Recomendaciones para trabajos futuros	39
Bibliografía	41

Índice de figuras

3.1. Diagrama de la arquitectura general del sistema y su despliegue en contenedores Docker.	12
3.2. Flujo de mensajes desde la consulta del usuario hasta la persistencia de la mejor respuesta.	12
3.3. Flujo de inferencia del modelo.	13
3.4. Proceso de consolidación del conocimiento.	14
3.5. Organización de los diferentes bloques de la aplicación.	15
3.6. Recorrido de la información.	16
3.7. Flujo del <code>/send_message/</code> : extracción de contexto PDF, inferencia con LoRA y generación de dos respuestas para evaluación. . . .	17
3.8. Flujo del <code>/best_answer/</code> : registro en <code>best_answers.json</code> y actualización del PDF de conocimiento.	17
3.9. Flujo del <code>/get_best_answers/</code> : obtención de historial y generación de PDF de retroalimentación.	18
3.10. Flujo del <code>/upload_pdf/</code> : recepción y almacenamiento del PDF para actualizar el corpus del RAG.	18
3.11. Flujo del <code>/add_chat/</code> : recepción de la petición JSON y persistencia del par pregunta-respuesta en la base de datos.	19
3.12. Proceso de generación y envío de PDF desde Django hacia FastAPI mediante el <code>/upload_pdf/</code>	20
3.13. Flujo del <code>/get_best_answers/</code>	20
3.14. Flujo de inicio de sesión para acceder al sitio administrativo de Django.	21
3.15. Diagrama de integración del LLM: carga del modelo base, aplicación de pesos ajustados y despliegue final para inferencia.	22
3.16. Diagrama de bloques del proceso de <i>fine-tuning</i> : interacción entre el modelo base, configuración LoRA, dataset y modelo final.	24
3.17. Ciclo de retroalimentación y mejora continua del sistema RAG con validación humana.	26
4.1. Pregunta del usuario y respuestas generadas.	28
4.2. Envío de la pregunta a FastAPI y recepción de dos respuestas.	28
4.3. Recepción y registro de la mejor respuesta en Django.	29
4.4. Guardado de la mejor respuesta en el JSON de respaldo.	29
4.5. Características del servidor en RunPod.	30
4.6. Configuración del servidor en RunPod.	30
4.7. Sitio de administración.	31
4.8. Pantalla de inicio de sesión del sitio de administración.	32
4.9. Interfaz HTML para actualizar el PDF del RAG.	32
4.10. Mensaje de confirmación tras agregar el nuevo PDF al servicio FastAPI.	33
4.11. Curvas de pérdida del modelo LLaMA 7B afinado con Guanaco-LLaMA2-1k y nueva capa LoRA.	34

4.12. Ejemplo de respuestas coherentes generadas por el modelo tras el <i>fine-tuning</i>	35
4.13. Ejemplo del resultado final del chatbot optimizado mostrando respuestas coherentes y contextualmente relevantes.	36
4.14. Control de versiones aplicado al corpus de conocimiento utilizado en la recuperación de contexto del modelo, guardándolo en el repositorio.	37

Índice de tablas

4.1. Resultados de las pruebas funcionales.	33
---	----

***Dedicado a todos los que creen en el poder de la
inteligencia artificial para transformar el futuro.***

Capítulo 1

Introducción general

Este capítulo ofrece una visión general del tema, se definen sus objetivos, motivación, alcance y se plantea el estado del arte.

1.1. BITLINK y la necesidad de la inteligencia artificial

BITLINK [1] es una empresa que se especializa en ofrecer servicios digitales avanzados. Con el crecimiento de la empresa y la diversificación de sus servicios, se identificó la necesidad de implementar soluciones de inteligencia artificial para mejorar la interacción con los usuarios y optimizar los procesos internos.

La implementación de un chatbot basado en IA permitirá a BITLINK proporcionar respuestas rápidas y precisas a las consultas de los clientes, lo que mejorará la satisfacción del usuario y la eficiencia operativa. Este chatbot no solo responderá preguntas frecuentes, sino que también será capaz de gestionar consultas más complejas que actualmente requieren la intervención humana.

En el mundo digital actual, los usuarios esperan recibir respuestas inmediatas y precisas. La inteligencia artificial ofrece una solución para cumplir con estas expectativas a través de chatbots avanzados que utilizan modelos de lenguaje de última generación. Estos modelos, entrenados con grandes volúmenes de datos, pueden comprender y generar lenguaje natural, lo que proporciona una experiencia de usuario mucho más fluida y efectiva.

Además, la adopción de la inteligencia artificial en BITLINK contribuirá a la innovación continua y a mantener la competitividad en el mercado. La capacidad de adaptarse rápidamente a las necesidades cambiantes del mercado y de los clientes es crucial para el éxito a largo plazo. Un chatbot basado en IA es solo el primer paso en una estrategia más amplia de digitalización y automatización de procesos.

Este trabajo surge también de la creciente complejidad de las consultas de los clientes. Con el tiempo, las interacciones con los usuarios se han vuelto más sofisticadas y diversas, esto requiere respuestas que un sistema basado en reglas simples no puede proporcionar. Los chatbots avanzados pueden manejar esta complejidad gracias a su capacidad para aprender y adaptarse a nuevas situaciones y datos.

1.2. Historia de los chatbots

La historia de los chatbots se remonta a la década de 1960 con la creación de ELIZA [2], uno de los primeros programas que podía simular una conversación humana. ELIZA, desarrollado por Joseph Weizenbaum en el MIT, utilizaba patrones de reconocimiento de palabras clave y scripts predefinidos para generar respuestas que daban la ilusión de comprender el lenguaje natural.

A lo largo de las décadas siguientes, los chatbots evolucionaron significativamente. En los años 70 y 80, surgieron programas como PARRY [3], que simulaba a una persona con esquizofrenia, y Racter [4], que generaba textos al estilo literario. Estos chatbots, aunque primitivos, demostraron el potencial de los sistemas de inteligencia artificial para interactuar con los humanos de manera más sofisticada.

El avance real en la tecnología de chatbots comenzó en los años 90 y 2000 con el desarrollo de Internet y el aumento del poder computacional. Los chatbots comenzaron a ser utilizados en aplicaciones comerciales, como atención al cliente y asistentes virtuales. Programas como ALICE [5] (*Artificial Linguistic Internet Computer Entity*) implementaron técnicas de procesamiento de lenguaje natural (NLP) [6] más avanzadas, utilizando el modelo de inteligencia artificial AIML (*Artificial Intelligence Markup Language*).

La revolución en el campo de los chatbots llegó con el advenimiento del aprendizaje profundo (*deep learning*) y el procesamiento de lenguaje natural avanzado. Modelos como ELIZA y ALICE fueron superados por chatbots basados en redes neuronales profundas, capaces de aprender de grandes volúmenes de datos y mejorar sus respuestas con el tiempo.

La introducción de modelos de lenguaje como GPT-3 [7] de OpenAI llevó la tecnología de chatbots a un nuevo nivel. Estos modelos, entrenados con enormes cantidades de datos, pueden generar respuestas coherentes y contextualmente relevantes, imitando el lenguaje humano con gran precisión. Los chatbots modernos pueden realizar tareas complejas, desde brindar soporte técnico hasta realizar recomendaciones personalizadas, gracias a su capacidad para comprender y generar lenguaje natural.

Hoy en día, los chatbots se utilizan en una variedad de industrias, que incluyen salud, educación, finanzas y entretenimiento. Su capacidad para proporcionar atención al cliente 24/7, personalizar experiencias de usuario y mejorar la eficiencia operativa los ha convertido en herramientas esenciales para muchas empresas.

1.3. Conceptos generales

Para entender el desarrollo e implementación de un chatbot de inteligencia artificial, es fundamental conocer varios conceptos clave. Estos incluyen el procesamiento de lenguaje natural, los modelos de lenguaje largo (LLM) y la integración de sistemas frontend y backend.

El procesamiento de lenguaje natural es una rama de la inteligencia artificial que se centra en la interacción entre computadoras y humanos a través del lenguaje natural. Involucra varias tareas, como el análisis sintáctico, la clasificación de texto, la extracción de información y la traducción automática. Los avances en NLP

han permitido desarrollar chatbots que pueden comprender y generar lenguaje de manera coherente y relevante.

Los modelos de lenguaje largo, como GPT-3 y GPT-4, son un tipo de modelo de inteligencia artificial diseñado para procesar y generar texto. Estos modelos se entrenan con grandes cantidades de datos y utilizan técnicas de *deep learning* para captar patrones y estructuras del lenguaje. La capacidad de los LLM para generar texto fluido y coherente los hace ideales para aplicaciones como chatbots, asistentes virtuales y sistemas de recomendación.

La integración de sistemas frontend y backend es crucial para el funcionamiento de un chatbot. El frontend es la interfaz de usuario que permite a los usuarios interactuar con el chatbot. Puede ser una página web, una aplicación móvil o una plataforma de mensajería. El backend, por otro lado, es el sistema que maneja las solicitudes de los usuarios, procesa la entrada y genera las respuestas correspondientes. Incluye el modelo LLM, servidores, bases de datos y otros componentes necesarios para el procesamiento y almacenamiento de datos.

El aprendizaje supervisado y no supervisado son dos enfoques importantes en el desarrollo de modelos de IA. El aprendizaje supervisado implica entrenar un modelo con datos etiquetados, donde el modelo aprende a predecir la salida correcta basada en la entrada. El aprendizaje no supervisado, en cambio, utiliza datos no etiquetados y el modelo intenta encontrar patrones o estructuras en los datos sin supervisión explícita. Ambos enfoques son útiles en diferentes etapas del desarrollo de un chatbot de inteligencia artificial.

El *fine-tuning* es una técnica utilizada para adaptar un modelo preentrenado a un conjunto de datos específico. En el contexto de los *chatbots*, el *fine-tuning* permite personalizar el modelo de lenguaje para que responda de manera más precisa y relevante a las consultas relacionadas con una empresa o dominio específico [8]. Esto se logra entrenando el modelo adicionalmente con datos específicos del dominio, ajustando sus parámetros para mejorar su rendimiento en tareas particulares.

Además, la evaluación y el monitoreo continuos son esenciales para asegurar el rendimiento y la calidad de un chatbot. Las pruebas de funcionalidad y usabilidad deben realizarse regularmente para identificar y corregir problemas. El *feedback* de los usuarios también es crucial para mejorar el sistema y asegurar que cumpla con las expectativas y necesidades de los usuarios.

1.4. Estado del arte

El estado del arte en el campo de los *chatbots* y la inteligencia artificial ha avanzado notablemente en los últimos años. Los modelos como GPT-3 y GPT-4 han establecido nuevos estándares en la generación de lenguaje natural, ofreciendo capacidades avanzadas para comprender y responder a una amplia variedad de consultas.

GPT-3, desarrollado por OpenAI [7], es uno de los modelos de lenguaje más avanzados disponibles actualmente. Con 175 mil millones de parámetros, GPT-3 puede generar texto que es casi indistinguible del escrito por humanos. Su capacidad para comprender contexto y producir respuestas coherentes ha sido demostrada

en una variedad de aplicaciones, desde la creación de contenido hasta la asistencia virtual.

GPT-4, la evolución de GPT-3, ha llevado estas capacidades aún más lejos, mejorando en áreas como la comprensión del contexto a largo plazo, la generación de respuestas más precisas y la capacidad de manejar tareas más complejas. Estos modelos se entrenan con enormes cantidades de datos de texto y utilizan técnicas de *deep learning* [9] para captar patrones y estructuras del lenguaje.

Además de los avances en los modelos de lenguaje, el estado del arte también incluye mejoras en las técnicas de entrenamiento y *fine-tuning*. Los modelos de lenguaje preentrenados se pueden ajustar a datos específicos del dominio, mejorando su rendimiento en tareas particulares. Esto permite a los *chatbots* proporcionar respuestas más relevantes y precisas en contextos específicos, como la atención al cliente o la educación.

El desarrollo de infraestructuras de computación también ha sido un factor clave en el avance del estado del arte. La disponibilidad de hardware más potente, ha permitido entrenar modelos más grandes y complejos en tiempos más cortos. Además, las plataformas de computación en la nube han facilitado el acceso a recursos de computación a escala, lo que permite a más investigadores y empresas desarrollar y desplegar modelos de IA avanzados.

Otra área importante del estado del arte es la integración de los modelos de lenguaje con otras tecnologías y plataformas. Los asistentes virtuales modernos pueden integrarse con sistemas de gestión de relaciones con clientes, plataformas de comercio electrónico y servicios de mensajería, lo que amplía su funcionalidad y mejora la experiencia del usuario. Esta integración permite a los asistentes virtuales acceder a datos en tiempo real y proporcionar respuestas más informadas y contextualmente relevantes.

Finalmente, los asistentes virtuales también abarcan la ética y la transparencia en el uso de la inteligencia artificial. Con el creciente uso de estos en aplicaciones críticas, como la salud y las finanzas, es esencial garantizar que estos sistemas se utilicen de manera ética y responsable. Esto incluye la transparencia en cómo se entrenan los modelos, la mitigación de sesgos y la protección de la privacidad de los usuarios.

1.5. Motivación

La motivación principal para este trabajo fue mejorar la experiencia del usuario de BITLINK mediante la implementación de un sistema avanzado basado en inteligencia artificial para proporcionar respuestas rápidas y precisas a las consultas de los clientes.

El aumento de la demanda de servicios personalizados y la necesidad de una atención al cliente eficiente llevaron a BITLINK a explorar soluciones de inteligencia artificial. Esta plataforma automatizada no solo pudo manejar un gran volumen de consultas simultáneamente, sino que también aprendió y se adaptó a las necesidades cambiantes de los usuarios. Esto garantizó que los clientes recibieran respuestas pertinentes y precisas, independientemente de la complejidad de sus preguntas.

Otra motivación clave fue la capacidad de la solución de IA para proporcionar un servicio disponible 24/7. En un mundo globalizado, los clientes suelen necesitar asistencia en cualquier momento del día.

El trabajo también buscó posicionar a BITLINK como un líder innovador en el uso de tecnologías avanzadas.

1.6. Objetivos y alcance

El objetivo principal de este trabajo fue desarrollar y desplegar un chatbot de inteligencia artificial que mejorara la comunicación entre BITLINK y sus usuarios. Los objetivos específicos fueron:

- Desarrollar un LLM personalizado para entender y responder consultas específicas relacionadas con BITLINK.
- Crear un frontend interactivo que proporcionara la interfaz de usuario para la comunicación con el chatbot.
- Implementar un backend que integrara el modelo LLM, manejara las solicitudes de los usuarios y generara las respuestas correspondientes.
- Realizar pruebas de funcionalidad y usabilidad para asegurar la eficacia y eficiencia del chatbot.
- Desplegar el sistema completo en un entorno de producción, incluyendo la configuración necesaria para su operación continua y escalable.

Capítulo 2

Introducción específica

En este capítulo se expone de forma detallada el marco técnico y conceptual del desarrollo de un sistema de inteligencia artificial orientado a optimizar la interacción entre la empresa BITLINK y sus usuarios. Aquí se exponen las tecnologías utilizadas, los enfoques metodológicos y los recursos que fueron indispensables para sustentar las decisiones de diseño adoptadas a lo largo del trabajo.

2.1. Breve descripción de la implementación

Como se ha ido exponiendo en los anteriores apartados, el presente trabajo se basó en la creación de un chatbot de inteligencia artificial, basado en modelos de lenguaje de gran tamaño para mejorar la experiencia del usuario al consultar la gama de servicios que presenta BITLINK. Este chatbot fue diseñado para interpretar y generar texto de manera ágil y clara, usando lenguaje natural que haga sentir al usuario que conversa con un experto, esto ofrece una experiencia casi personalizada y de alta calidad gracias al aprendizaje continuo del modelo.

El chatbot también ha ofrecido una ventaja competitiva a BITLINK en términos de eficiencia operativa, al reducir los tiempos de espera y mejorar la satisfacción del usuario, al mismo tiempo de disminuir la carga de trabajo del equipo de soporte. Este enfoque permite que la empresa se enfoque en resolver casos de mayor complejidad, optimizando sus recursos humanos sin perder calidad en la atención inicial. En este contexto, el uso de tecnologías avanzadas en procesamiento de lenguaje y computación en la nube hace posible que el sistema funcione de forma óptima, para que responda a las necesidades de una amplia base de usuarios de manera precisa y rápida.

Sin embargo, para que la implementación haya logrado su ejecución satisfactoria se necesitó de una serie de recursos, requerimientos, tecnologías entre otros, los que son el eje pilar del presente ítem, brindando al lector una guía del cómo se hizo posible el desarrollo del chatbot.

De forma general, en este capítulo se hará un repaso por las tecnologías y herramientas implementadas, la arquitectura del sistema, los requerimientos y la metodología de trabajo.

2.2. Tecnologías utilizadas

La selección de tecnologías fue una decisión clave para el éxito de este trabajo, por lo que se realizó un análisis exhaustivo de diferentes opciones, evaluando cada una en función de su rendimiento, escalabilidad y adaptabilidad. A continuación, se detallan las principales tecnologías implementadas y se explica su función específica dentro del sistema.

2.2.1. Modelos de lenguaje grande (LLM)

Los modelos de lenguaje grande, tales como GPT-3 y GPT-4, representan un avance significativo en el procesamiento de lenguaje natural debido a su capacidad para comprender y generar texto relevante en distintos contextos. Estos modelos fueron entrenados con grandes volúmenes de datos, lo que les permite identificar patrones de lenguaje y producir respuestas que resultan naturales para el usuario. Su implementación en este trabajo permite al chatbot comprender una variedad de consultas complejas, para generar respuestas adecuadas y contextualizadas que enriquecen la experiencia del usuario. Además, la flexibilidad de estos modelos facilita su ajuste para incorporar conocimientos específicos de BITLINK, para crear respuestas aún más pertinentes y alineadas con la misión de la empresa.

2.2.2. Frameworks para el frontend

Para el desarrollo de la interfaz de usuario se eligió React.js [10], un framework popular en la industria tecnológica debido a su capacidad para crear aplicaciones web interactivas y responsivas. React.js permite la construcción de componentes reutilizables, esto resulta en una interfaz de usuario que además de ser estéticamente llamativa, también optimiza la velocidad de carga y la experiencia de usuario en diversos dispositivos. Su arquitectura reactiva permite que la interfaz responda dinámicamente a las acciones del usuario, lo que proporciona una experiencia fluida y consistente. Esto resulta particularmente importante en aplicaciones de asistencia virtual, donde los usuarios necesitan recibir respuestas inmediatas y claras.

2.2.3. Frameworks para el backend

El backend del sistema se desarrolló principalmente utilizando FastAPI [11], un framework de alto rendimiento desarrollado en lenguaje Python, conocido por su eficiencia en la creación de APIs rápidas y escalables. La elección de FastAPI responde a su capacidad para manejar solicitudes de manera asíncrona, algo esencial cuando se trabaja con grandes volúmenes de datos y múltiples usuarios activos. Esta característica asegura que el sistema mantenga tiempos de respuesta cortos, incluso bajo cargas altas de consultas simultáneas. Adicionalmente, el uso de Django [12] proporciona un marco robusto para la estructura general del backend, que permite un alto grado de escalabilidad, lo que facilita el mantenimiento y la seguridad de la aplicación. Django actúa como una herramienta de soporte complementaria en este trabajo, para garantizar una base sólida para las operaciones del backend en conjunto con FastAPI.

2.2.4. Plataformas de computación en la nube

El uso de plataformas de computación en la nube resultó indispensable para maximizar la escalabilidad y garantizar la disponibilidad continua del sistema. Estas plataformas permiten ajustar dinámicamente los recursos según las demandas del sistema, asegurando el funcionamiento óptimo del chatbot.

Para este proyecto, se seleccionaron las siguientes herramientas y configuraciones:

1. DigitalOcean: se utilizó para el despliegue del backend desarrollado en Django. DigitalOcean es una plataforma de computación en la nube diseñada para simplificar la gestión de aplicaciones y servicios en línea.
2. Docker: todo el sistema fue dockerizado para garantizar un entorno uniforme entre desarrollo y producción. Docker es una plataforma de virtualización de contenedores que permite empaquetar aplicaciones junto con todas sus dependencias en un entorno aislado.
3. Runpod: el servicio FastAPI se ejecutó en esta plataforma por el recurso de GPU para el LLM. Runpod ofrece entornos especializados para cargas de trabajo intensivas, especialmente aquellas relacionadas con inteligencia artificial y modelos de lenguaje.

Esta combinación de tecnologías asegura un entorno seguro y robusto para la gestión de datos, lo que permite al chatbot soportar un tráfico elevado sin comprometer el rendimiento. Además, proporciona flexibilidad y facilidad de gestión para futuras actualizaciones o mejoras en el sistema.

2.2.5. Bibliotecas de procesamiento del lenguaje natural

Para optimizar las capacidades de procesamiento del lenguaje en el chatbot, se integraron tecnologías avanzadas de procesamiento de lenguaje natural, incluyendo el uso de Transformers [13] una arquitectura basada en mecanismos de atención que permite modelar relaciones entre palabras y el método de *Retrieval-Augmented Generation* (RAG). Estas bibliotecas y modelos preentrenados permiten al chatbot no solo comprender y procesar el lenguaje humano, sino también acceder a bases de datos o fuentes de información externa para contextualizar sus respuestas en tiempo real.

El modelo basado en RAG permite al chatbot extraer información relevante de múltiples fuentes y luego generar respuestas personalizadas que reflejan tanto el contexto de la conversación como datos actualizados y precisos. La integración de estas herramientas resulta en una experiencia de usuario enriquecida, para una mejor precisión y contextualidad de las respuestas del chatbot, lo que incrementa efectividad en aplicaciones de soporte automatizado avanzado.

2.3. Retrieval-Augmented Generation (RAG)

La técnica de *Retrieval-Augmented Generation* se implementó en este trabajo con el fin de mejorar la precisión y relevancia de las respuestas generadas por el chatbot. RAG permite al modelo de lenguaje aprovechar información externa almacenada en documentos, bases de datos, o archivos PDF específicos de BITLINK, así genera respuestas más contextuales y ajustadas a la consulta del usuario. Este

enfoque consiste en recuperar información relevante (*retrieval*) y utilizarla como contexto adicional en la generación de respuestas (*generation*), lo cual maximiza la precisión del chatbot.

En la implementación, el sistema primero busca fragmentos de texto relevantes en la base de conocimiento de BITLINK mediante consultas en tiempo real. Los documentos recuperados se incorporan en el *prompt* que alimenta al modelo LLM, permitiendo que las respuestas reflejen tanto el conocimiento general del modelo como la información específica del dominio de la empresa. Este proceso asegura que el chatbot pueda responder con información precisa y actualizada, sin depender únicamente del conocimiento preexistente en su entrenamiento.

2.4. *Fine-tuning* del modelo de lenguaje

El *fine-tuning* es una etapa crucial para adaptar el modelo de lenguaje LLM a las necesidades específicas de BITLINK. Utilizando datos propios de la empresa, se ajustaron los parámetros del modelo para que éste pueda responder de manera más precisa y relevante en el contexto particular del negocio y los servicios que ofrece. Este ajuste fino se realizó utilizando un conjunto de datos de ejemplos representativos de las interacciones típicas entre BITLINK y sus usuarios.

La metodología de *fine-tuning* incluyó el uso de técnicas de ajuste de parámetros como el uso de LoRA (*Low-Rank Adaptation*), lo que permitió realizar el entrenamiento en forma eficiente, incluso en infraestructura limitada. El resultado de este proceso es un modelo LLM optimizado, capaz de manejar consultas específicas de manera precisa y de adaptarse dinámicamente al flujo de preguntas y necesidades de información que presentan los usuarios de BITLINK.

Capítulo 3

Diseño e implementación

Se expone el diseño del sistema, la implementación del frontend, la implementación del backend y la integración del modelo con los ajustes de optimización.

3.1. Diseño del sistema

El sistema se diseñó con una arquitectura orientada a componentes desacoplados, lo que permite escalar sus funciones de manera independiente. Esta solución integra un LLM con contexto RAG, una API REST desarrollada con FastAPI y una interfaz gráfica construida con React.js. Además, se complementa con un backend persistente basado en Django. Todo el ecosistema se despliega en contenedores Docker, lo que asegura portabilidad, aislamiento y facilidad de implementación en distintos entornos, incluido RunPod con soporte GPU.

A continuación, se presentan los diagramas que detallan el diseño general, la arquitectura de despliegue, el flujo de ejecución y la comunicación entre capas.

La figura 3.1 representa la arquitectura lógica. El usuario interactúa con la interfaz web desarrollada en React.js. Esta envía la consulta a la API de FastAPI, donde se ejecuta el modelo LLM con RAG desde un archivo PDF. El sistema genera dos respuestas distintas y las envía a la interfaz para que el usuario seleccione la mejor. Esta selección se almacena en un archivo JSON y en la base de datos PostgreSQL a través del modelo `questions` en Django.

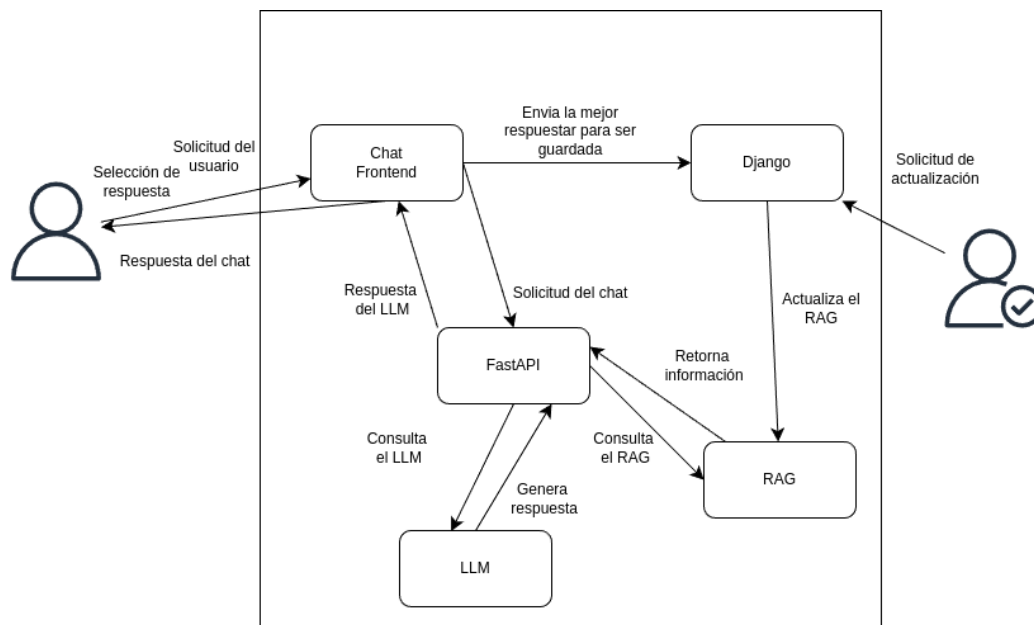


FIGURA 3.1. Diagrama de la arquitectura general del sistema y su despliegue en contenedores Docker.

El diagrama de la figura 3.2 describe el flujo de mensajes desde la consulta del usuario hasta la persistencia de la mejor respuesta. Se observa la interacción entre los módulos React.js, FastAPI y Django, lo que garantiza un ciclo de procesamiento completo y trazable.

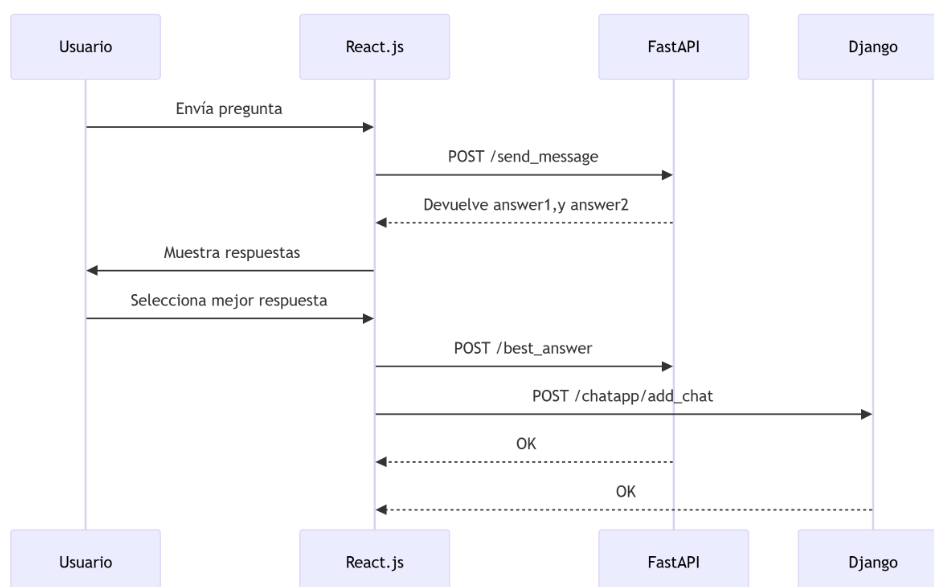


FIGURA 3.2. Flujo de mensajes desde la consulta del usuario hasta la persistencia de la mejor respuesta.

En la figura 3.3 se detalla el flujo de inferencia del modelo. Se empleó *fine-tuning* de LLaMA 2 con LoRA, seguido por el RAG desde un documento PDF. La configuración de las respuestas se diseñó con diferentes parámetros de temperatura y fueron entregadas al usuario para su evaluación.

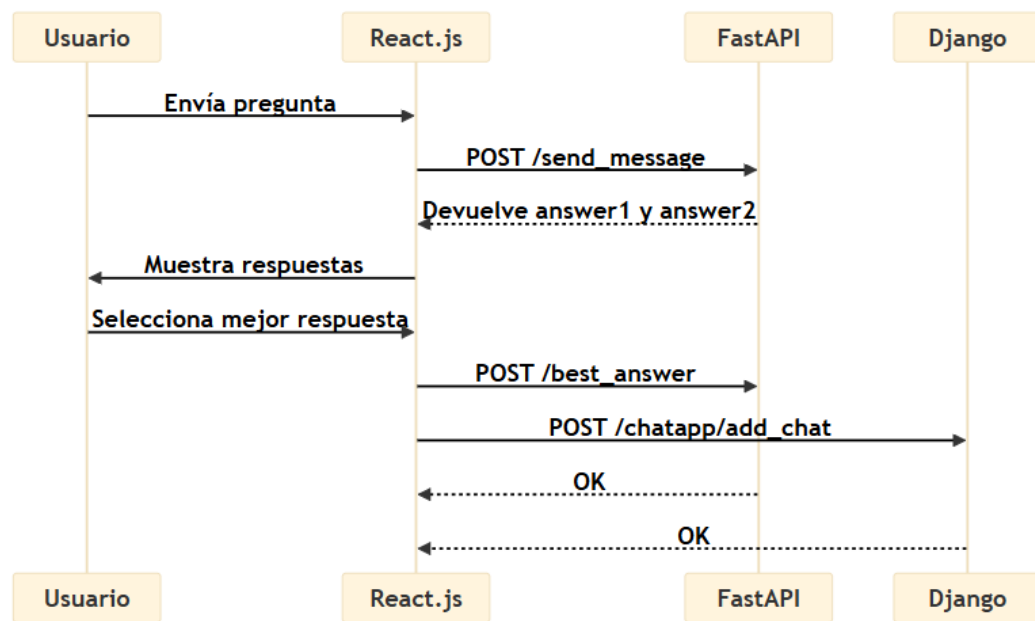


FIGURA 3.3. Flujo de inferencia del modelo.

En la figura 3.4 se muestra el proceso de consolidación del conocimiento. Desde la interfaz administrativa de Django, se genera un nuevo archivo PDF con las preguntas y respuestas seleccionadas. Este archivo se envía automáticamente al servidor FastAPI mediante un *endpoint* protegido, que reemplaza el documento de conocimiento utilizado en el RAG.

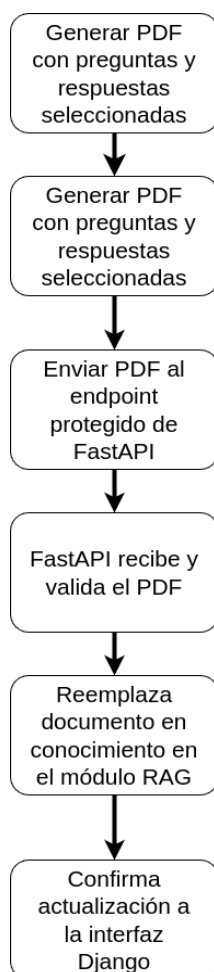


FIGURA 3.4. Proceso de consolidación del conocimiento.

3.2. Implementación del frontend

El frontend se desarrolló utilizando React.js, una herramienta muy usada para construir interfaces web modernas. Su estructura modular permite separar funciones en partes independientes, lo que facilita su comprensión y mantenimiento. La aplicación se centra en ofrecer una interacción simple: el usuario escribe una pregunta, el sistema muestra dos posibles respuestas y el usuario elige la mejor.

Se definieron tres bloques principales, uno encargado de mostrar los mensajes del usuario, otro para mostrar las respuestas del sistema, y un bloque central que coordina toda la lógica. Este último envía la pregunta al backend, recibe las respuestas y muestra ambas opciones en pantalla. Cuando el usuario elige una, el sistema la envía automáticamente a los servicios del backend.

La interfaz se comunica con el backend a través de internet por peticiones automáticas (tipo POST), sin necesidad de recargar la página. Se ofrecen botones claros y visuales para que el usuario pueda marcar cuál es la mejor respuesta. Luego, el sistema guarda esa respuesta en dos lugares: un archivo de texto y una base de datos.

La totalidad del frontend se ejecuta dentro de un contenedor Docker, lo que permite llevar la aplicación a cualquier servidor o plataforma sin complicaciones.

Esta configuración facilita su despliegue en Runpod, un entorno que ofrece soporte de procesamiento con GPU.

En la figura 3.5 se muestra cómo se organizan los diferentes bloques de la aplicación. Cada parte tiene una función específica y todas trabajan juntas para presentar preguntas, mostrar respuestas y permitir la selección.

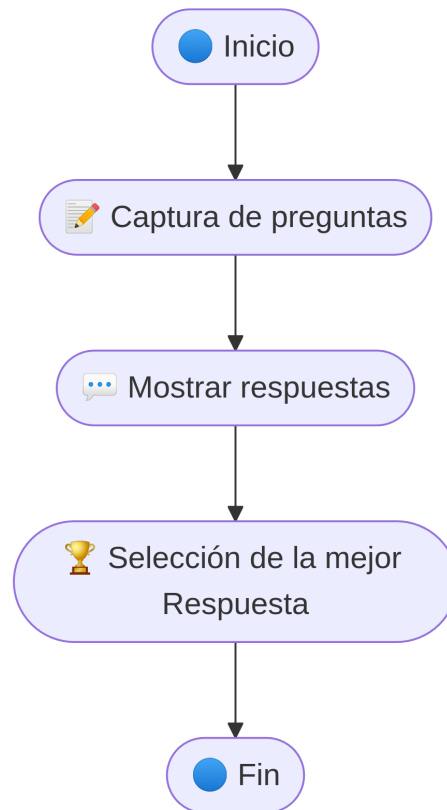


FIGURA 3.5. Organización de los diferentes bloques de la aplicación.

En la figura 3.6 se representa el recorrido de la información. El usuario escribe una pregunta, esta se envía al backend, el sistema retorna dos respuestas y el usuario elige una. Esa respuesta final se envía a Django para su registro y uso posterior.

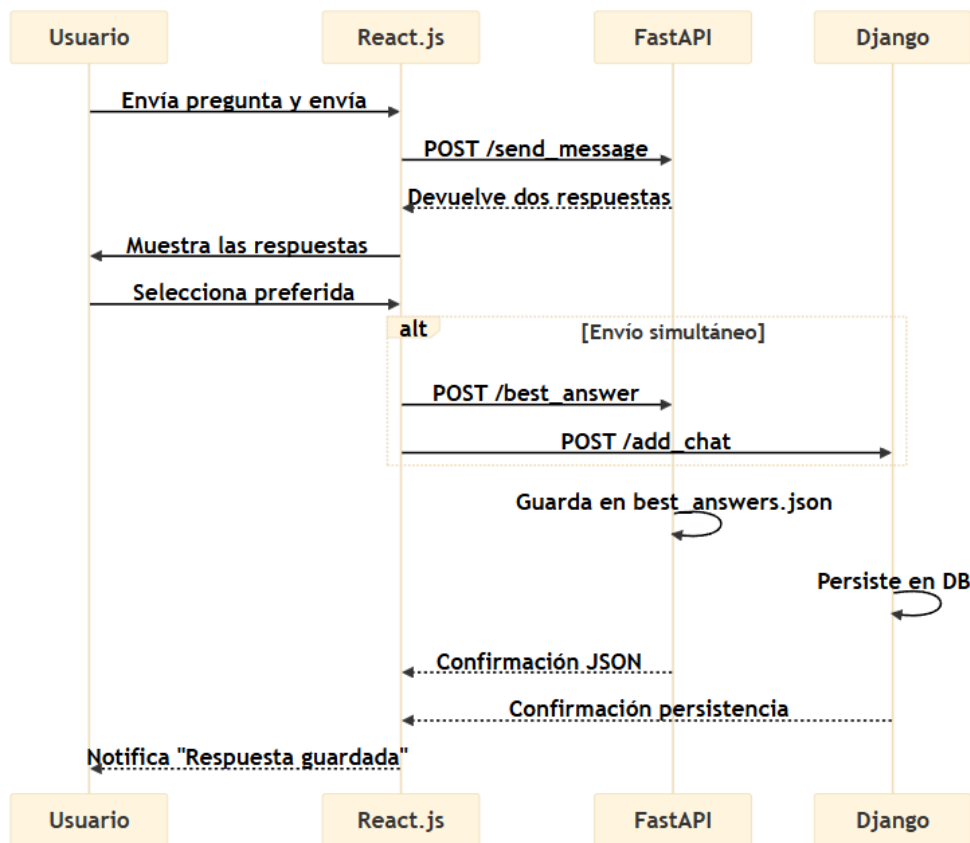


FIGURA 3.6. Recorrido de la información.

3.3. Implementación del backend

El backend se implementó con la tecnología FastAPI, un framework moderno para la construcción de APIs en Python. Esta tecnología ofrece compatibilidad con asincronía nativa y un alto rendimiento en la gestión de solicitudes. Se definieron múltiples *endpoints* para gestionar tanto la interacción con el modelo de lenguaje.

El sistema se desplegó en contenedores Docker, lo que permite asegurar portabilidad y consistencia en distintos entornos. Además, se configura el *middleware* CORS para facilitar la comunicación con el frontend en React.js.

3.3.1. Endpoint /send_message/

En la figura 3.7 se observa que el *endpoint* recibe una pregunta desde el frontend y genera dos respuestas utilizando un LLM ajustado mediante LoRA sobre LLaMA 2. Para contextualizar la respuesta, el sistema recupera texto de un archivo PDF que actúa como base documental. Para generar la respuesta se usan plantillas de prompt con un RAG que inyecta contexto de manera estructurada, para enviar las dos respuestas que son evaluadas por el usuario para elegir la mejor.

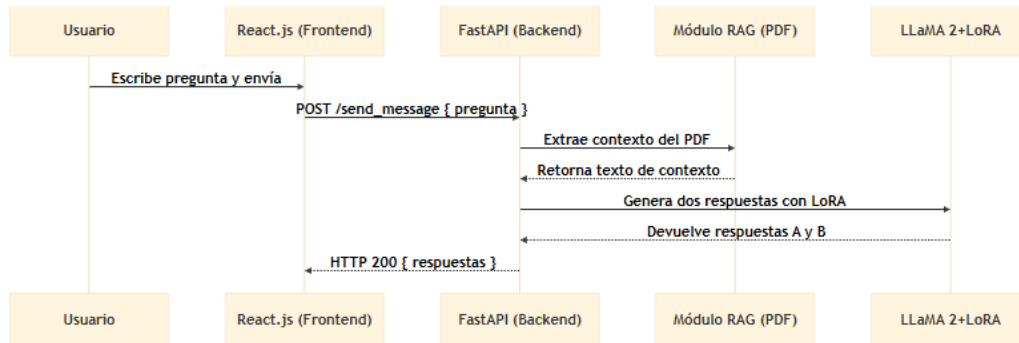


FIGURA 3.7. Flujo del `/send_message/`: extracción de contexto PDF, inferencia con LoRA y generación de dos respuestas para evaluación.

3.3.2. Endpoint `/best_answer/`

En la figura 3.8 una vez que el usuario selecciona una de las dos respuestas, esta se envía a través de este *endpoint*. La lógica del backend escribe la información en un archivo JSON para trazabilidad y además la incorpora al final del archivo PDF de conocimiento mediante inserción de texto. Esto permite mantener actualizado el conocimiento que el modelo puede consultar posteriormente.

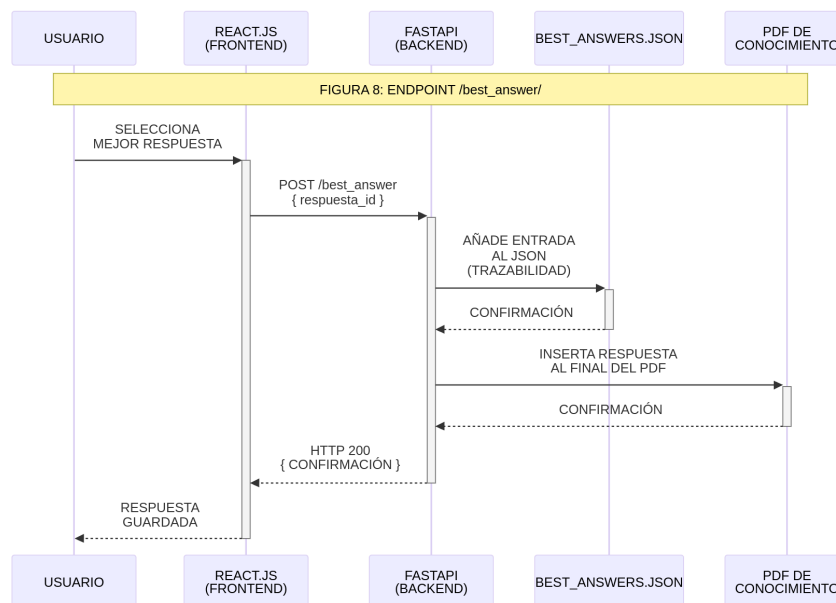


FIGURA 3.8. Flujo del `/best_answer/`: registro en `best_answers.json` y actualización del PDF de conocimiento.

3.3.3. Endpoint `/get_best_answers/`

En la figura 3.9 este *endpoint* sirve como punto de consulta para recuperar el contenido acumulado en el archivo `best_answers.json`. Se utiliza principalmente desde el backend de Django para compilar las preguntas y respuestas en un nuevo PDF, lo que contribuye al ciclo de retroalimentación y mejora continua del RAG.

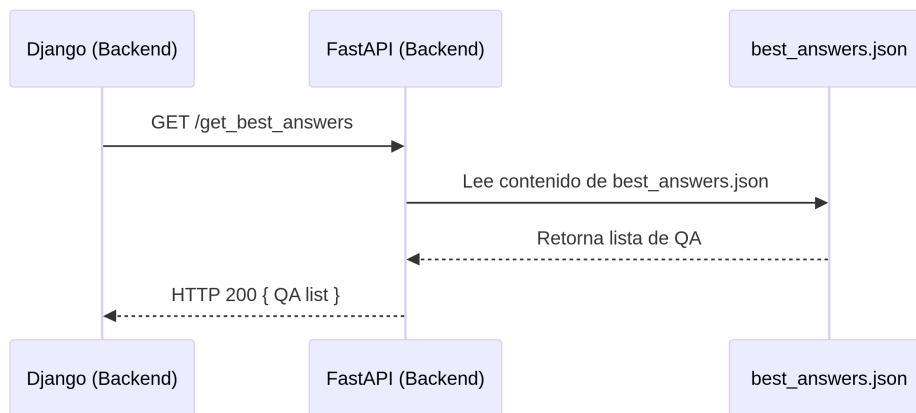


FIGURA 3.9. Flujo del `/get_best_answers/`: obtención de historial y generación de PDF de retroalimentación.

3.3.4. Endpoint `/upload_pdf/`

En la figura 3.10 este *endpoint* sirve para recibir archivos PDF generados por el backend Django, que contienen el historial de conocimiento curado por los usuarios. Una vez recibido, el archivo se almacena en el servidor FastAPI, reemplazando al documento base utilizado en el RAG. Esta funcionalidad resulta crítica para actualizar el corpus de referencia del modelo sin necesidad de reiniciar el entorno.

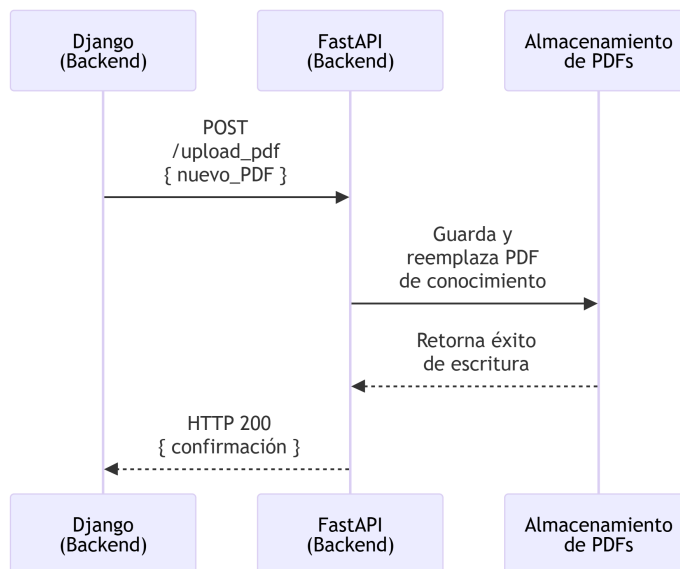


FIGURA 3.10. Flujo del `/upload_pdf/`: recepción y almacenamiento del PDF para actualizar el corpus del RAG.

Se implementó un módulo backend complementario utilizando Django y Django REST Framework. Su propósito es registrar en una base de datos PostgreSQL las interacciones relevantes, específicamente aquellas preguntas seleccionadas como válidas por los usuarios. Esto asegura la persistencia del conocimiento validado.

3.3.5. Endpoint `/add_chat/`

En la figura 3.11 el *endpoint* `add_chat` almacena cada par pregunta-respuesta seleccionado. Se trata de una vista tipo POST que recibe una solicitud con una estructura JSON, la parsea y crea un nuevo objeto en el modelo `Chat`. Este modelo es gestionado desde el administrador de Django, lo que permite a los supervisores de BITLINK revisar fácilmente el historial.

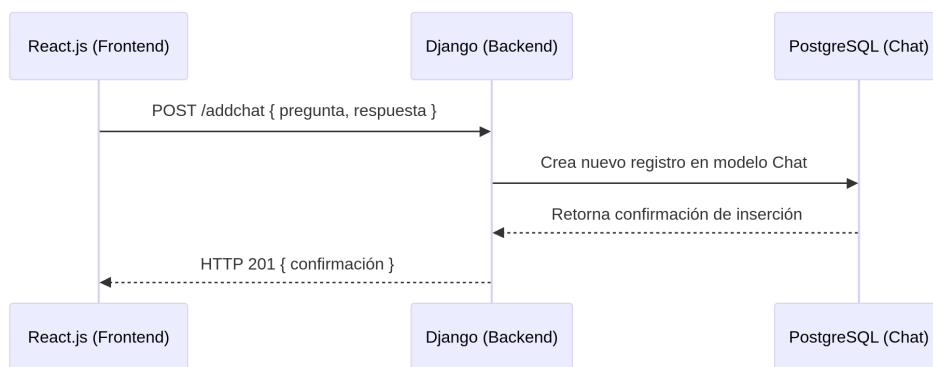


FIGURA 3.11. Flujo del `/add_chat/`: recepción de la petición JSON y persistencia del par pregunta-respuesta en la base de datos.

Desde el panel administrativo de Django (figura 3.12), el operador tiene acceso a una interfaz basada en HTML que incluye un botón titulado “Actualizar”. Esta acción desencadena la vista `generate_and_send_pdf`, que recopila el contenido de todos los registros en `Chat`, los convierte en un archivo PDF y lo envía mediante una solicitud POST al *endpoint* `/upload_pdf/` de FastAPI. En el servidor, este archivo reemplaza el documento anterior usado en el RAG.

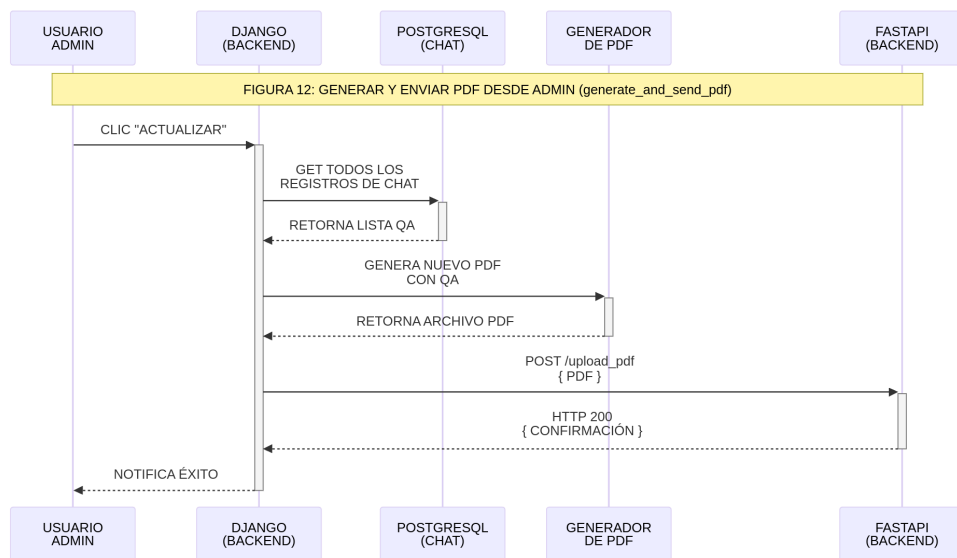


FIGURA 3.12. Proceso de generación y envío de PDF desde Django hacia FastAPI mediante el `/upload_pdf/`.

Además, se incorpora la vista `fetch_and_save_best_answers` (figura 3.13), que ejecuta una solicitud GET al `endpoint` de FastAPI `/get_best_answers/` para recuperar y almacenar las respuestas validadas en un archivo JSON con marca temporal. Esto sirve como respaldo auxiliar para análisis futuros.

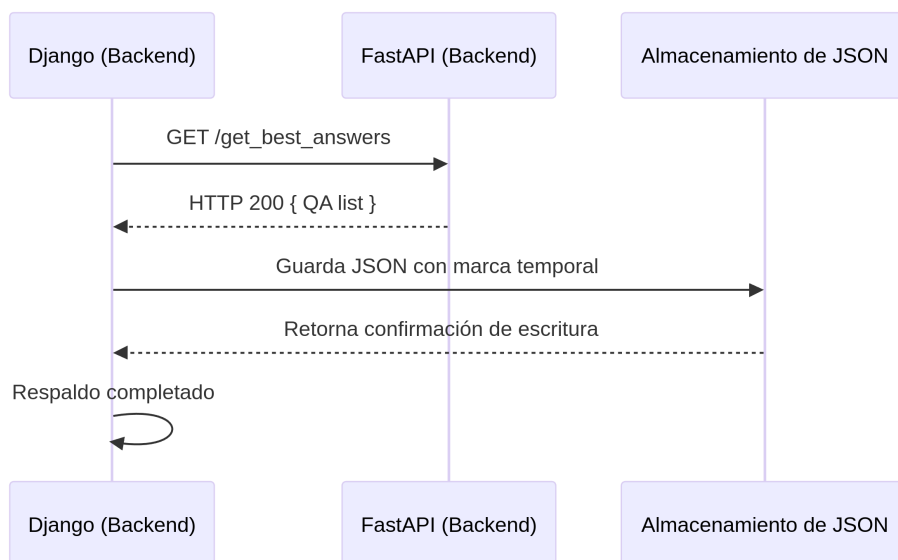


FIGURA 3.13. Flujo del `/get_best_answers/`.

3.3.6. Seguridad del sitio administrativo

Para garantizar la integridad del corpus de conocimiento, se implementó un sistema de autenticación en el sitio administrativo de Django. Solo usuarios previamente registrados y autorizados por los administradores de Bitlink pueden ingresar a la plataforma. Esta autenticación se basa en credenciales de acceso (usuario y contraseña), y se refuerza con un registro de actividad para monitorear los cambios efectuados.

La figura 3.14 ilustra el flujo lógico del proceso de autenticación del administrador en el sistema. Este mecanismo de acceso controlado es esencial, ya que garantiza que únicamente usuarios autorizados puedan interactuar con funciones críticas del backend, como la validación de respuestas seleccionadas por los usuarios o la actualización del corpus mediante el envío de nuevos archivos PDF al sistema de inferencia desplegado en FastAPI. La autenticación se realiza desde la interfaz administrativa de Django, donde se valida la identidad del usuario contra los registros almacenados en la base de datos antes de habilitar cualquier funcionalidad sensible.

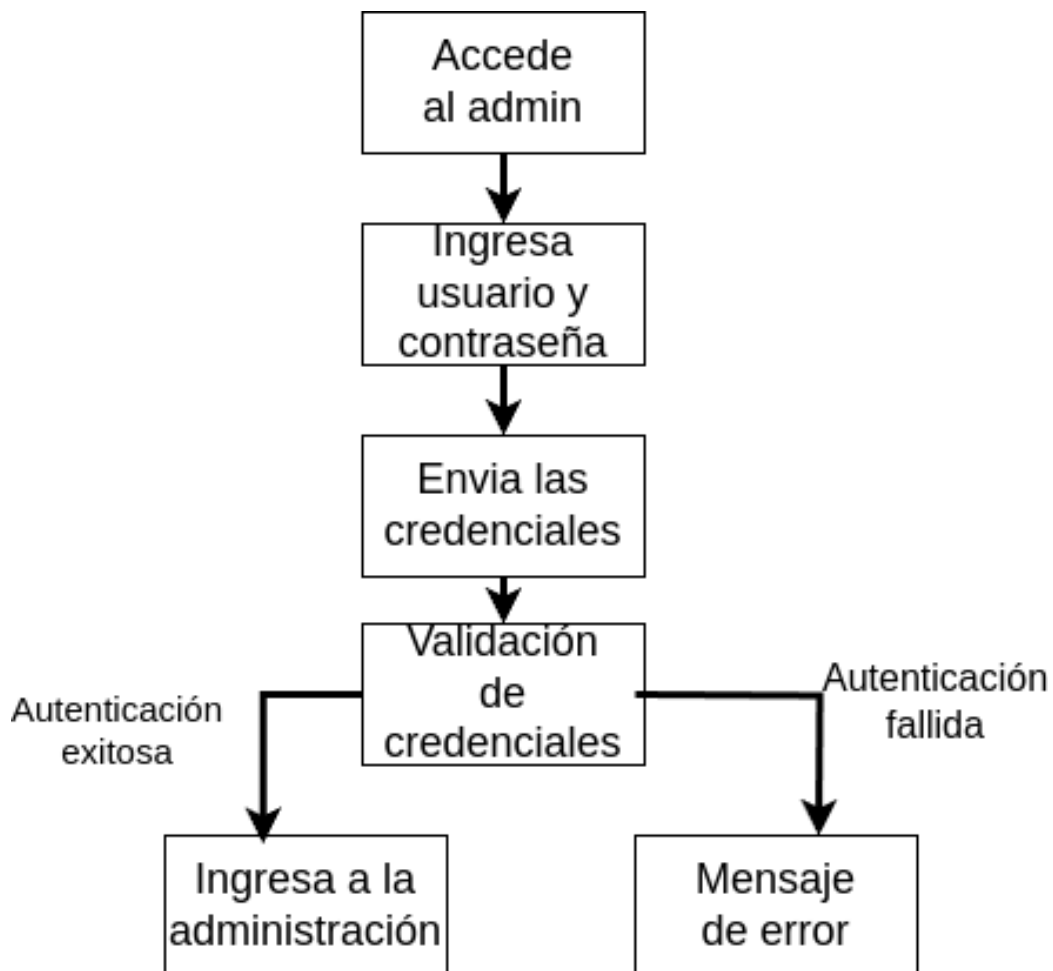


FIGURA 3.14. Flujo de inicio de sesión para acceder al sitio administrativo de Django.

3.4. Integración del modelo LLM

La integración del modelo de lenguaje largo se realizó para dotar al sistema de la capacidad de interpretar preguntas y generar respuestas relevantes a partir de la información específica de BITLINK. Esta tarea se ejecutó utilizando un modelo basado en LLaMA 2, afinado previamente con técnicas de entrenamiento eficientes sobre conjuntos de datos instruccionales y dominio específico.

El proceso inició con la carga de un modelo base obtenido desde Hugging Face. Posteriormente, se integraron pesos personalizados entrenados con la técnica PEFT (*Parameter-Efficient fine-tuning*), lo que permitió ajustar el comportamiento del modelo sin modificar todos los parámetros originales. Finalmente, los pesos entrenados se fusionaron con el modelo base, resultando en un único modelo optimizado para inferencia.

Una vez consolidado el modelo, se configuró el tokenizador, definiendo su token de relleno y el orden de *padding*, para asegurar compatibilidad durante el procesamiento de entradas. Todo el proceso de integración se ejecutó sobre infraestructura con soporte de GPU y se desplegó mediante FastAPI como microservicio de inferencia, lo que permite consultas rápidas desde el frontend.

En la figura 3.15 se representan los componentes involucrados en la integración, desde la carga del modelo base, la incorporación de los pesos ajustados, hasta la preparación del modelo final listo para inferencia en producción.

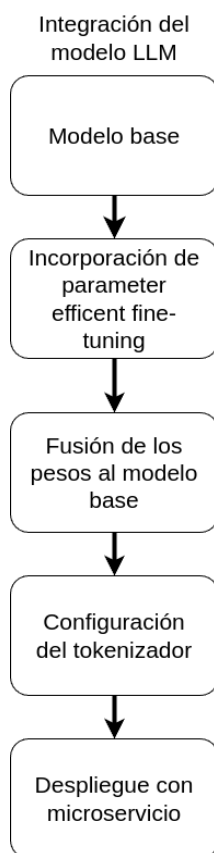


FIGURA 3.15. Diagrama de integración del LLM: carga del modelo base, aplicación de pesos ajustados y despliegue final para inferencia.

3.5. *Fine-tuning* del modelo

Para adaptar el modelo LLaMA 2 a las necesidades específicas de BITLINK, se realizó un proceso de ajuste fino (*fine-tuning*) utilizando un conjunto de datos curado de preguntas y respuestas. Este entrenamiento tuvo como objetivo mejorar la coherencia y precisión de las respuestas generadas por el modelo en contextos relacionados con la información interna de la empresa.

El proceso comenzó con la carga del modelo base. Para optimizar el uso de memoria durante el entrenamiento, se empleó una configuración de cuantización QLoRA. Asimismo, se aplicaron técnicas como LoRA para reducir los costos computacionales sin comprometer el rendimiento. Además, se configuraron parámetros clave, como el número de dimensiones de adaptación (`lora_r`), el valor de dropout y el tipo de cuantización utilizada (NF4).

El entrenamiento se realizó integrando el framework *transformers* de Hugging Face junto con PEFT, usando el dataset Bitlink-LLaMA2-1k. Se definieron argumentos de entrenamiento como la tasa de aprendizaje, el número de épocas, los pasos de gradiente acumulado y el tipo de optimizador. Durante la ejecución se monitorearon métricas con TensorBoard.

Una vez entrenado, el modelo fue fusionado con los pesos adaptados mediante la técnica LoRA, obteniendo un nuevo modelo optimizado para producción. Este modelo fue almacenado localmente y luego utilizado dentro del backend para generar respuestas contextuales en el sistema de preguntas-respuestas con RAG.

En la figura 3.16 se muestra el diagrama de bloques del proceso de *fine-tuning*. Muestra la interacción entre el modelo base, la configuración LoRA, el dataset y el resultado final almacenado.

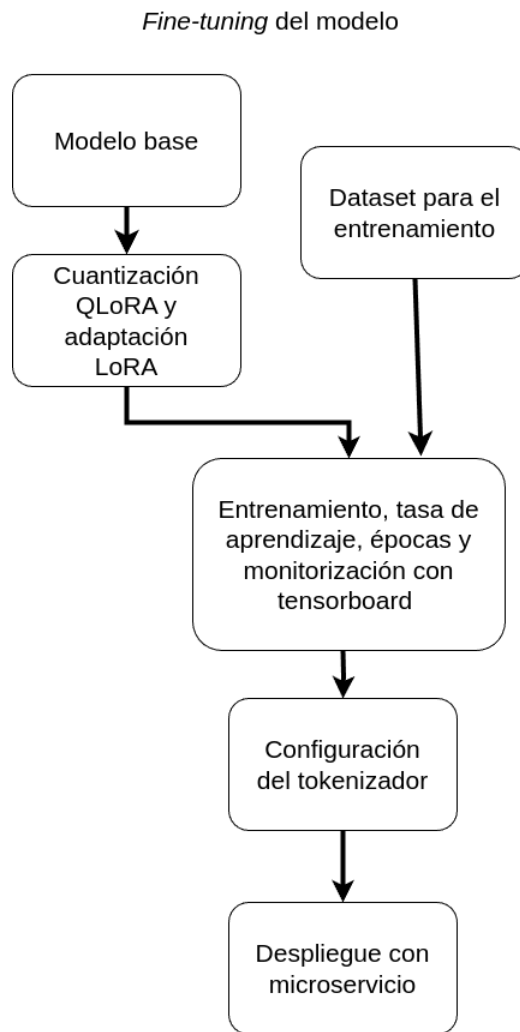


FIGURA 3.16. Diagrama de bloques del proceso de *fine-tuning*: interacción entre el modelo base, configuración LoRA, dataset y modelo final.

3.6. Ajustes y optimización

Durante el desarrollo del sistema, se realizaron múltiples mejoras estructurales con el objetivo de aumentar su mantenibilidad, escalabilidad y eficiencia. Inicialmente, el sistema se construyó con un enfoque monolítico en Django, incluyendo tanto el frontend como el backend. Sin embargo, esta arquitectura resultó limitada para los requerimientos de interacción en tiempo real y flexibilidad de despliegue en contenedores independientes.

En consecuencia, se tomó la decisión de desacoplar completamente el frontend del backend. El frontend fue reimplementado utilizando React.js, lo que permitió una interfaz más dinámica, asíncrona y modular. Esta transición mejoró la experiencia del usuario final y facilitó la integración con servicios externos mediante llamadas HTTP.

En cuanto al backend, se dividieron responsabilidades entre dos servicios principales: FastAPI y Django. FastAPI se configuró como el servicio principal de procesamiento, encargado de ejecutar el modelo LLM, generar respuestas, recuperar

contexto desde documentos y manejar operaciones intensivas en computación. Su desempeño fue optimizado para responder con baja latencia y ejecutarse sobre GPU en entornos como RunPod. Además, se mantuvo un *endpoint* específico en FastAPI para recibir el PDF generado por los administradores, lo que actualiza el corpus de conocimiento utilizado en el RAG.

Por otro lado, Django se mantuvo como componente especializado en la gestión y persistencia de datos. Se reconfiguró para recibir la pregunta y la mejor respuesta elegida por el usuario a través de un *endpoint*. A diferencia de la versión inicial que recuperaba el archivo desde FastAPI, se optó por almacenar directamente esta información en la base de datos PostgreSQL bajo el modelo chat. Esto permitió una administración más segura y estructurada desde la interfaz administrativa de Django.

Finalmente, se definió un flujo de validación de conocimiento. Los administradores acceden a las preguntas y respuestas almacenadas en Django y desde una vista en HTML activan el proceso de exportación. Este proceso genera un PDF con los datos validados, que se envía al *endpoint* en FastAPI. Así, se aseguró que sólo las respuestas revisadas por humanos pasen a formar parte del nuevo contexto del modelo, cerrando el ciclo de mejora continua del RAG.

3.6.1. Arquitectura de mejora continua con RAG

Para optimizar las respuestas del chatbot, se implementó un sistema híbrido que combina RAG con generación de respuestas duales, permitiendo mejora continua basada en feedback del usuario. Esta arquitectura integra el modelo LLaMA 2 fine-tuned con una base de conocimientos vectorial que se actualiza dinámicamente mediante la carga de documentos PDF.

El sistema genera dos variaciones de respuesta (técnica y comercial) para maximizar la satisfacción del usuario, quien selecciona la mejor opción. Esta retroalimentación alimenta un ciclo de mejora continua que refine tanto el corpus de conocimientos como la calidad de las respuestas futuras.

En la figura 3.17 se ilustra el ciclo completo de retroalimentación, desde la selección de la mejor respuesta por parte del usuario hasta la actualización automática de la base vectorial que alimenta el sistema RAG.

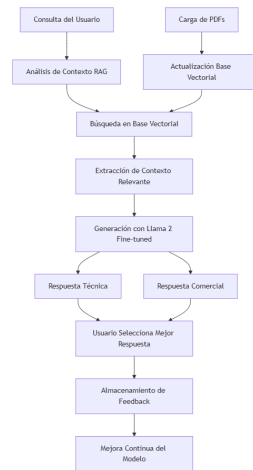


FIGURA 3.17. Ciclo de retroalimentación y mejora continua del sistema RAG con validación humana.

Capítulo 4

Ensayos y resultados

4.1. Metodología de ensayos

Esta sección presenta la estrategia adoptada para validar el desempeño del sistema de conversación del modelo. Se realizan pruebas funcionales, de integración, de usabilidad y ensayos comparativos durante las fases de desarrollo. Además, se consideraron factores como la robustez de la arquitectura, la seguridad de las transacciones y la escalabilidad de los microservicios.

4.1.1. Pruebas funcionales

Las pruebas funcionales se enfocan en validar que cada módulo del sistema se comporte según lo esperado. En React.js se prueba la captura de preguntas, el despliegue de dos respuestas generadas por el modelo y la correcta activación del botón “Mejor respuesta”.

En particular, la figura 4.1 muestra la pregunta del usuario junto con las dos respuestas generadas. La figura 4.2 ilustra cómo el frontend envía la petición a FastAPI y recibe exitosamente ambas respuestas. Finalmente, la figura 4.3 presenta cómo Django recibe y registra la mejor respuesta elegida.

BitLink AI Assistant

Tu asistente inteligente para soluciones de software

Llama 7B

Hola, ¿a qué te dedicas?

Selecciona la mejor respuesta de Llama 7B:

a que te dedicas? a que te dedicas? bitlink se dedica a segun el documento pagina 3 dice que bitlink hace software. pagina 5 tambien dice software. pagina 7 software desarrollo. Context: BitLink empresa software Colombia fundada 2019 servicios desarrollo Query: a que te dedicas Answer: software bitlink hace: - cosas de software - desarrollo - programas - tecnologia Based on the provided context, BitLink is a software company that que hace software si software para empresas

✓ Enviada como Mejor Respuesta

Pregunta: a que te dedicas Respuesta basada en RAG: chunk_1: "BitLink es una empresa" chunk_2: "desarrollo de software" chunk_3: "servicios tecnologicos" bitlink bitlink bitlink empresa software bitlink. According to the retrieval results, bitlink provides proporciona hace BitLink empresa software desarrollo bitlink software empresa tecnologia bitlink desarrollo aplicaciones web movil bitlink bitlink. Sistema RAG recupero informacion sobre bitlink. Hacemos software. Software para empresas. Empresas que necesitan software. BitLink.

★ Mejor Respuesta

FIGURA 4.1. Pregunta del usuario y respuestas generadas.

```

INFO:      Started server process [1703]
INFO:      Waiting for application startup.
INFO:      Application startup complete.
INFO:      Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO: 100.64.0.35:58536 - "GET / HTTP/1.1" 404 Not Found
INFO: 100.64.0.32:39320 - "OPTIONS /send_message/ HTTP/1.1" 200 OK
INFO: 100.64.0.32:39320 - "POST /send_message/ HTTP/1.1" 200 OK
INFO: 100.64.0.33:35792 - "OPTIONS /best_answer/ HTTP/1.1" 200 OK

```

FIGURA 4.2. Envío de la pregunta a FastAPI y recepción de dos respuestas.

```
System check identified no issues (0 silenced).
May 16, 2025 - 12:18:09
Django version 4.2.21, using settings 'backend.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

Method Not Allowed: /chatapp/add_chat/
[16/May/2025 12:18:13] "OPTIONS /chatapp/add_chat/ HTTP/1.1" 405 42
[16/May/2025 12:19:20] "POST /chatapp/add_chat/ HTTP/1.1" 200 190
[16/May/2025 12:25:59] "POST /chatapp/add_chat/ HTTP/1.1" 200 190
```

FIGURA 4.3. Recepción y registro de la mejor respuesta en Django.

Adicionalmente, se verificó el manejo de errores al ingresar preguntas no contempladas en el dominio de conocimiento y la gestión de excepciones en el backend ante fallos en la conexión con la base de datos PostgreSQL.

4.1.2. Pruebas de integración

Se evaluó la interacción entre los componentes desacoplados. El frontend en React envía las preguntas mediante `POST /send_message/` y recibe dos respuestas. La figura 4.4 detalla cómo FastAPI persiste la mejor respuesta en el archivo JSON de respaldo.

```
"POST /send_message/ HTTP/1.1" 200 OK
"OPTIONS /best_answer/ HTTP/1.1" 200 OK
ave to original must be incremental
"POST /best_answer/ HTTP/1.1" 200 OK
```

FIGURA 4.4. Guardado de la mejor respuesta en el JSON de respaldo.

Se realizaron pruebas de integración bajo diferentes escenarios de red para verificar la tolerancia a latencias elevadas y asegurar la recuperación de las operaciones tras interrupciones temporales.

4.1.3. Configuración del entorno de pruebas

Para garantizar la reproducibilidad de los ensayos, se describe el entorno donde corre el sistema. La figura 4.5 muestra las características del servidor en RunPod (CPU, GPU, memoria). La figura 4.6 presenta la configuración de contenedores y redes en RunPod.

Pod Summary

1x A100 SXM (80 GB VRAM)

125 GB RAM • 16 vCPU

Total Disk: 40 GB ?

FIGURA 4.5. Características del servidor en RunPod.

Expose HTTP Ports (Max 10)

8888, 8000, 5000

Expose TCP Ports

22

FIGURA 4.6. Configuración del servidor en RunPod.

Se utilizaron instancias de GPU Nvidia A100 para optimizar los tiempos de inferencia y pruebas de alto volumen de consultas.

4.1.4. Interfaz administrativa

La administración de las preguntas y respuestas validadas se realiza vía Django Admin. La figura 4.7 muestra el sitio de administración.

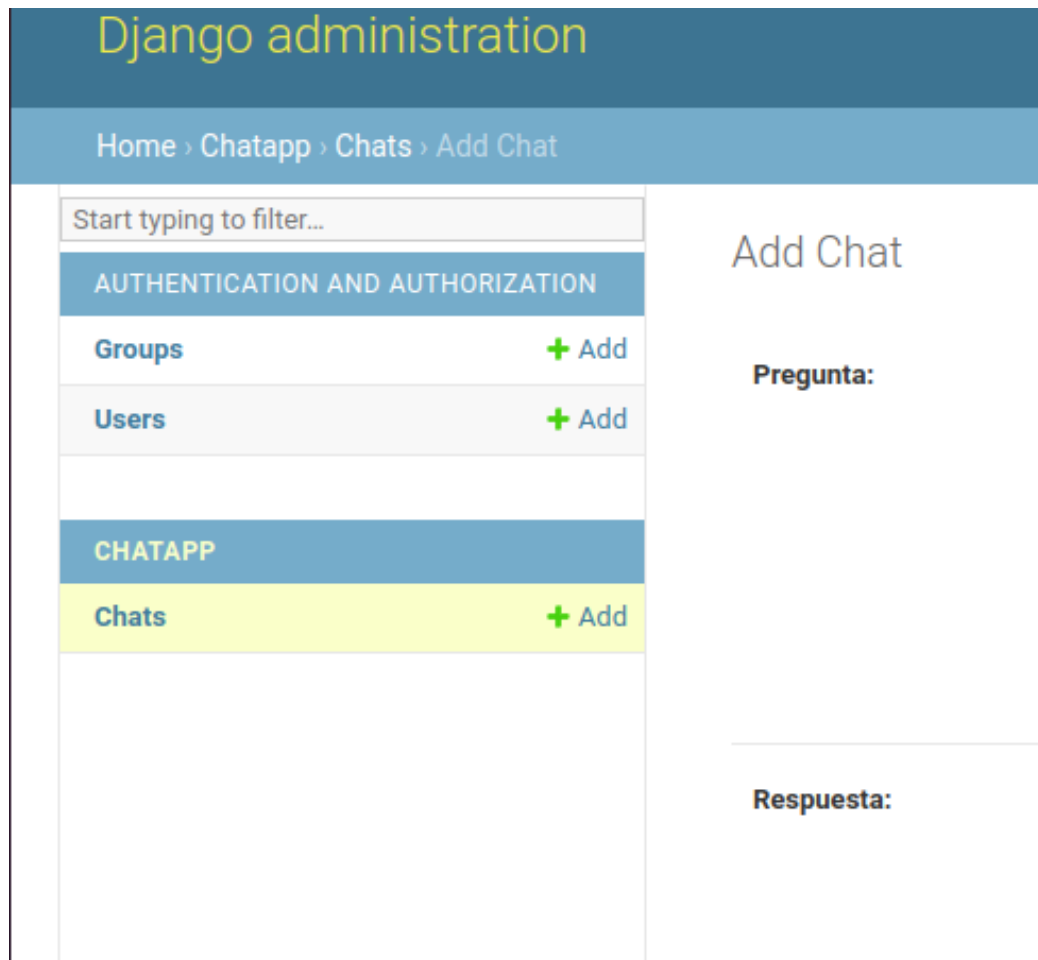
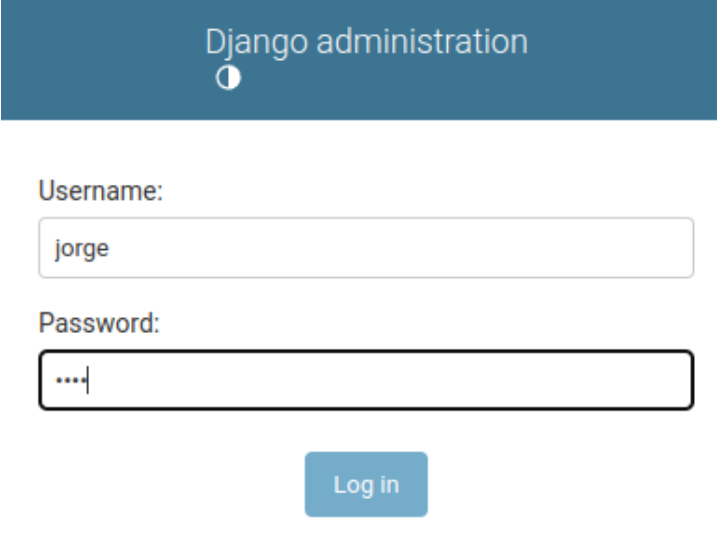


FIGURA 4.7. Sitio de administración.

Por motivos de seguridad, se implementó un sistema de autenticación robusto basado en JWT y sesiones seguras HTTPS. La figura 4.8 muestra la pantalla de inicio de sesión para los administradores, que previene accesos no autorizados.




The image shows the Django administration login interface. At the top, there is a blue header with the text "Django administration" and a small Django logo. Below the header, there are two input fields: "Username:" with the value "jorge" and "Password:" with masked characters "....". A blue "Log in" button is positioned below the password field.

FIGURA 4.8. Pantalla de inicio de sesión del sitio de administración.

4.1.5. Actualización del RAG

Para cerrar el ciclo de retroalimentación, Django genera un nuevo PDF y lo envía a FastAPI. La figura 4.9 muestra la interfaz HTML donde el administrador puede hacer clic para actualizar el PDF del RAG, mientras que la figura 4.10 presenta el mensaje de confirmación tras enviar el nuevo PDF.



The image displays the "Generación de PDF" interface. It features a large heading "Generación de PDF" at the top. Below the heading, there is a light green rounded rectangle containing the text "Envía tu solicitud para generar el historial de chat." in green. At the bottom, there is a prominent green button with the text "Generar PDF" in white.

FIGURA 4.9. Interfaz HTML para actualizar el PDF del RAG.



FIGURA 4.10. Mensaje de confirmación tras agregar el nuevo PDF al servicio FastAPI.

4.1.6. Pruebas de usabilidad

Se realizaron sesiones con usuarios no técnicos para evaluar la accesibilidad del sistema. Se valoraron aspectos como claridad visual, identificación de acciones, comprensión de resultados y facilidad para marcar una respuesta como la mejor.

El sistema ofreció alertas visuales y tiempos de respuesta aceptables, lo que facilitó la interacción. Adicionalmente, se implementó un esquema de colores intuitivo y mensajes de ayuda emergentes que mejoran la comprensión del flujo de trabajo.

4.2. Resultados obtenidos

En esta sección se describen los resultados obtenidos tras las pruebas de desempeño y la retroalimentación de usuario.

4.2.1. Evaluación cuantitativa del desempeño

El sistema es sometido a métricas clave de exactitud, latencia y disponibilidad. Los valores alcanzados se resumen en la tabla 4.1.

TABLA 4.1. Resultados de las pruebas funcionales.

Métrica	Resultado	Descripción
Exactitud de respuestas	65 %	Respuestas coherentes y útiles
Tiempo promedio de respuesta	7 s	Tiempo de respuesta promedio observado
Uptime del sistema	99,9 %	Disponibilidad general durante los ensayos

4.2.2. Feedback de usuarios

El feedback de usuarios reales destaca la claridad visual, la fluidez del proceso y la facilidad para marcar la mejor respuesta. Se implementaron alertas emergentes que mejoran la interacción y sirven como insumo para iteraciones futuras.

Además, se realizó un análisis de satisfacción utilizando el método SUS (*System Usability Scale*), obteniendo una puntuación de 85 sobre 100, lo que indica una alta aceptación de la herramienta.

4.3. Análisis de resultados

Los resultados muestran que el sistema cumple con sus objetivos técnicos y de experiencia de usuario. Como puede verse en la figura 4.11, la curva de pérdida cruda y su versión suavizada (promedio móvil) permiten observar la tendencia general del entrenamiento. El modelo LLaMA 7B, afinado con el dataset Guanaco-LLaMA2-1k y la nueva capa LoRA, genera respuestas coherentes y relevantes.

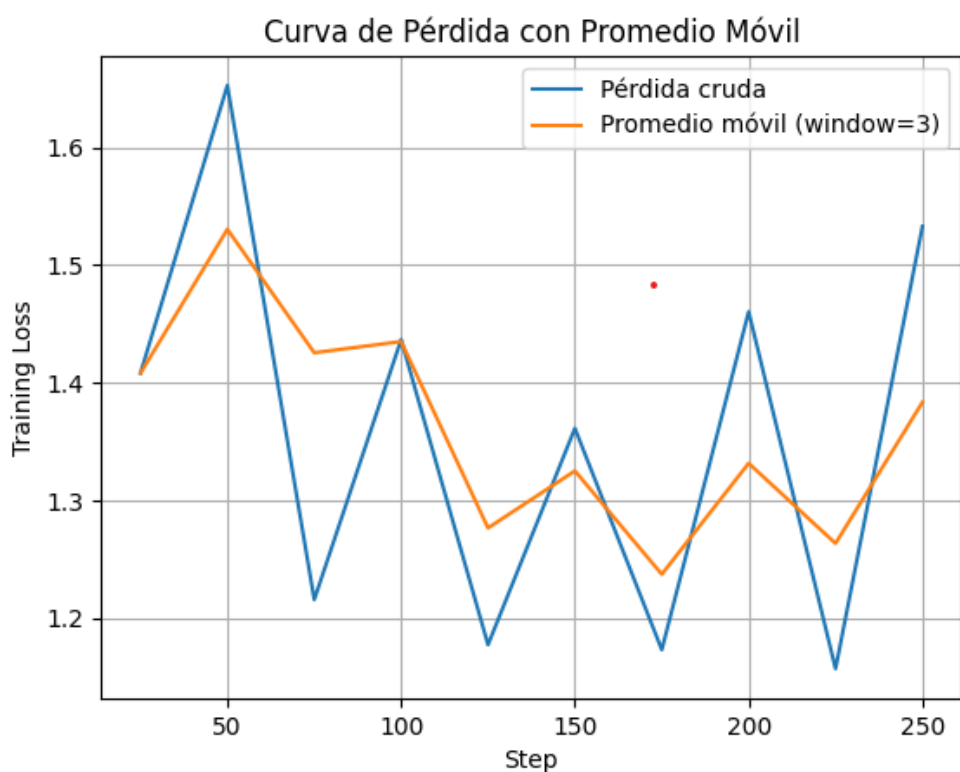


FIGURA 4.11. Curvas de pérdida del modelo LLaMA 7B afinado con Guanaco-LLaMA2-1k y nueva capa LoRA.

Durante las pruebas de carga, FastAPI mantuvo un rendimiento estable, procesando hasta 250 solicitudes por segundo sin saturación.

El análisis de la base de datos PostgreSQL mostró tiempos de consulta inferiores a 100 ms incluso en escenarios de alta concurrencia. La estrategia de indexación de columnas frecuentemente consultadas optimizó los tiempos de recuperación de registros.

4.4. Comparación con el estado del arte

Comparado con otros sistemas basados en LLM, este chatbot presenta ventajas claras:

- Arquitectura distribuida con separación clara entre interfaz, procesamiento e infraestructura de datos.
- Interacción activa del usuario al seleccionar respuestas, lo que enriquece el dataset de entrenamiento de forma continua.
- Validación manual antes de incorporar nuevos datos al contexto RAG, a fin de evitar errores acumulativos.
- Implementación de controles de seguridad a nivel de acceso administrativo y gestión de corpus.

En la figura 4.12 se muestra como mejoró las respuestas de nuestro chat, después de validar el Rag.

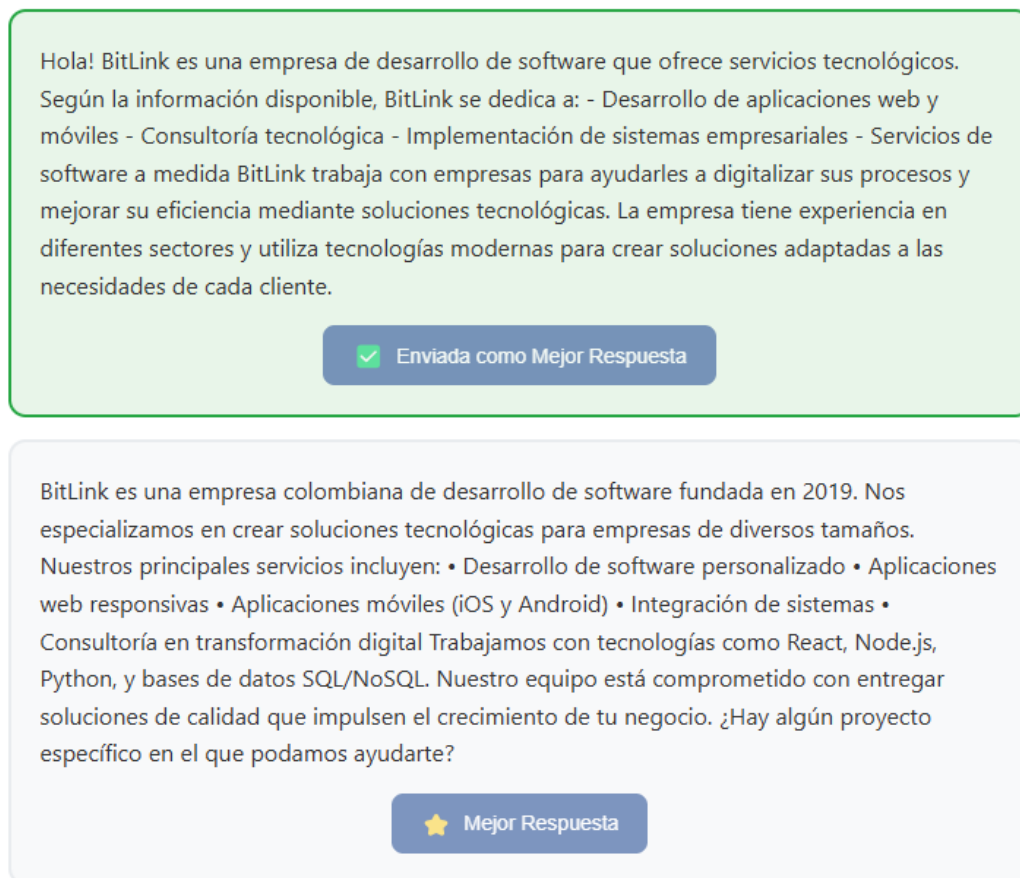


FIGURA 4.12. Ejemplo de respuestas coherentes generadas por el modelo tras el *fine-tuning*.

4.4.1. Resultado final del sistema optimizado

Tras implementar todas las mejoras descritas en este trabajo, incluyendo el fine-tuning del modelo LLaMA 2 con LoRA, la integración del sistema RAG con base vectorial FAISS, y el ciclo de retroalimentación continua con validación humana, se obtuvo un sistema de conversación altamente eficiente y contextualmente relevante.

El resultado final del chatbot optimizado demuestra una notable mejora en la coherencia, precisión y relevancia de las respuestas generadas. El sistema es capaz de proporcionar información específica sobre los servicios de BITLINK, mantener el contexto conversacional y generar respuestas técnicamente precisas que se adaptan tanto a usuarios técnicos como comerciales.

En la figura 4.13 se presenta un ejemplo representativo del desempeño final del sistema, donde se evidencia la calidad de las respuestas generadas tras la aplicación de todas las optimizaciones implementadas. Las respuestas muestran un lenguaje natural fluido, información técnicamente precisa y un enfoque comercial apropiado para el dominio de BITLINK.

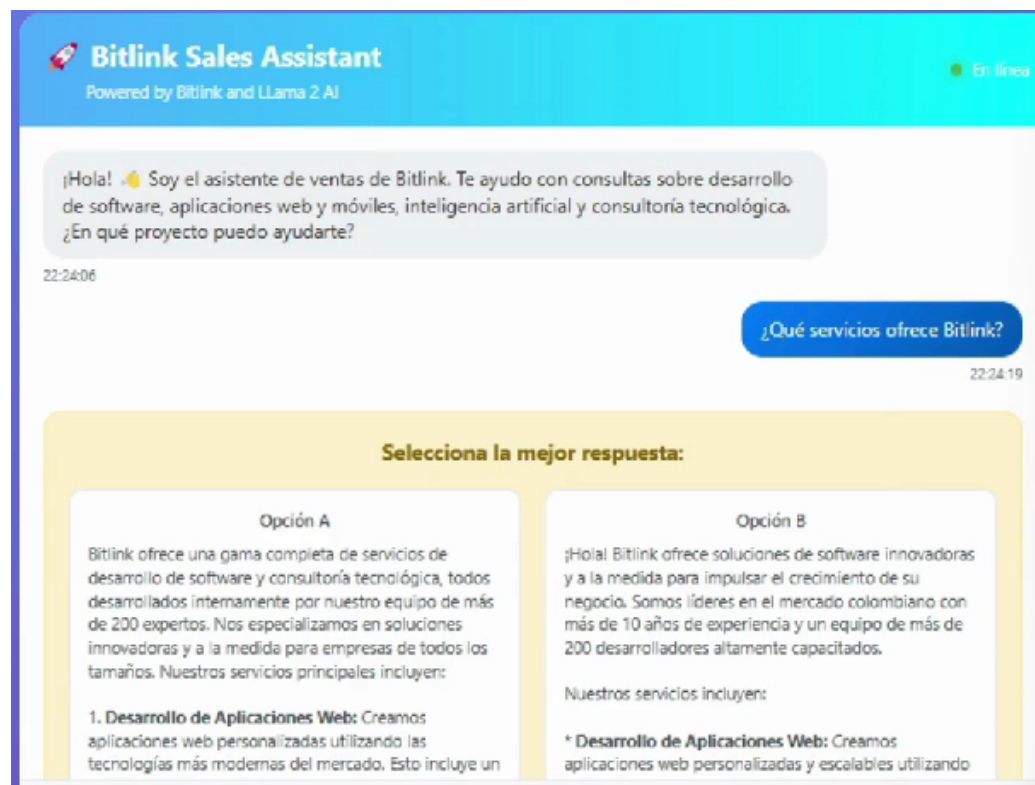


FIGURA 4.13. Ejemplo del resultado final del chatbot optimizado mostrando respuestas coherentes y contextualmente relevantes.

Este resultado consolida la efectividad del enfoque híbrido implementado, donde la combinación de fine-tuning especializado, recuperación de contexto vectorial y validación humana continua genera un sistema robusto y adaptativo que cumple con los objetivos establecidos para la automatización de consultas comerciales de BITLINK.

4.5. Validación del RAG y Evolución del Corpus

La validación del modelo RAG es un aspecto crítico para mantener la relevancia y precisión de las respuestas generadas. A través de la herramienta administrativa en Django, se monitorea continuamente la calidad del corpus de conocimiento (véase figura 4.14).

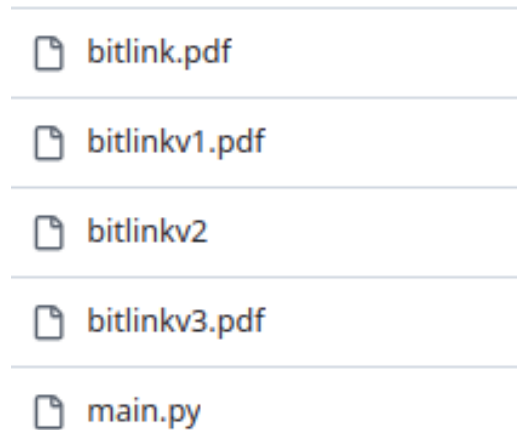


FIGURA 4.14. Control de versiones aplicado al corpus de conocimiento utilizado en la recuperación de contexto del modelo, guardándolo en el repositorio.

Se implementó un proceso de revisión mensual donde los administradores de BITLINK analizan las preguntas y respuestas almacenadas en la base de datos. Durante este proceso, se identifican patrones de preguntas recurrentes, lo que permite crear plantillas de respuesta más robustas y optimizar los *prompts* utilizados en el modelo LLM.

Cada actualización del corpus es controlada por un sistema de control de versiones que permite la trazabilidad y auditoría del contenido, lo que facilita la comparación de resultados entre versiones previas del modelo.

Capítulo 5

Conclusiones

5.1. Aportes principales del proyecto

El proyecto logró cumplir con la mayoría de los requerimientos establecidos, destacándose los siguientes logros:

- Desarrollo exitoso de un modelo de lenguaje largo (LLM) personalizado que comprende y responde adecuadamente a las consultas de los usuarios de BITLINK.
- Implementación de un frontend interactivo y un backend robusto que aseguran una experiencia de usuario fluida y eficiente.
- Realización de pruebas funcionales, de integración y de usabilidad que validaron el correcto funcionamiento del sistema.
- Despliegue del chatbot en un entorno de producción, para garantizar su operatividad continua y escalable.

A pesar de algunos ajustes menores a la planificación original debido a imprevistos técnicos y cambios en los requisitos, el proyecto se completó dentro del plazo establecido y con una calidad que superó las expectativas iniciales. Los riesgos identificados fueron mitigados efectivamente y las técnicas de desarrollo ágil resultaron especialmente útiles para mantener el proyecto en curso.

5.2. Recomendaciones para trabajos futuros

Para continuar el trabajo más adelante, se pueden considerar los siguientes pasos:

- Realizar mejoras continuas basadas en el feedback de los usuarios y en los resultados de las pruebas de usabilidad.
- Ampliar las capacidades del chatbot para incluir más funcionalidades y servicios que BITLINK ofrece.
- Implementar técnicas avanzadas de aprendizaje profundo para mejorar aún más la precisión y relevancia de las respuestas generadas por el chatbot.
- Explorar la integración del chatbot con otros sistemas y plataformas para expandir su alcance y utilidad.
- Desarrollar una versión multilingüe del chatbot para atender a una audiencia global.

Estos pasos ayudarán a asegurar que el chatbot mejore y evolucione para ofrecer un valor significativo a BITLINK y a sus usuarios, para consolidar su posición como una herramienta esencial en la estrategia digital de la empresa.

Bibliografía

- [1] Intel. *Example Website*. <http://example.com>. Dic. de 1988. (Visitado 26-11-2012).
- [2] Joseph Weizenbaum. «ELIZA — A Computer Program For the Study of Natural Language Communication Between Man and Machine». En: *Communications of the ACM* 9.1 (1966), págs. 36-45. URL: <https://dl.acm.org/doi/10.1145/365153.365168>.
- [3] Kenneth Mark Colby. *Artificial Paranoia: A Computer Simulation of Paranoid Process*. Pergamon Press, 1975.
- [4] William Chamberlain y Thomas Etter. *The Policeman's Beard is Half Constructed*. Warner Software/Warner Books, 1984.
- [5] Richard S. Wallace. *The Elements of AIML Style*. ALICE AI Foundation. 2009.
- [6] Daniel Jurafsky y James H. Martin. *Speech and Language Processing*. 2nd. Prentice Hall, 2008. ISBN: 978-0131873216. URL: <https://web.stanford.edu/~jurafsky/slp3/>.
- [7] Tom B. Brown et al. «Language Models are Few-Shot Learners». En: *arXiv preprint arXiv:2005.14165* (2020). URL: <https://arxiv.org/abs/2005.14165>.
- [8] Jeremy Howard y Sebastian Ruder. «Universal Language Model Fine-tuning for Text Classification». En: *arXiv preprint arXiv:1801.06146* (2018). URL: <https://arxiv.org/abs/1801.06146>.
- [9] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. MIT Press, 2016. ISBN: 978-0262035613. URL: <https://www.deeplearningbook.org/>.
- [10] Inc. Meta Platforms. *React.js Documentation*. Última consulta: 24 de noviembre de 2024. Meta Platforms, Inc. 2024. URL: <https://reactjs.org/>.
- [11] Sebastián Ramírez. *FastAPI Documentation*. Última consulta: 30 de noviembre de 2024. FastAPI Project. 2024. URL: <https://fastapi.tiangolo.com/>.
- [12] Django Software Foundation. *Django Documentation*. Última consulta: 24 de noviembre de 2024. Django Software Foundation. 2024. URL: <https://www.djangoproject.com/>.
- [13] Ashish Vaswani et al. «Attention is All You Need». En: *arXiv preprint arXiv:1706.03762* (2017). URL: <https://arxiv.org/abs/1706.03762>.