Promtior

CHATBOT ASSISTANT + RAG

TECHNICAL TEST 2025

Prepared By:

NICOLAS FARIZANO

nicolasfarizano10@gmail.com

+54 9 379 493-3651
linkedin.com/in/nicolás-farizano-8534a0233/

Index

Promtior Chatbot Assistant Documentation

Introduction

Promtior's technical test evaluates a candidate's ability to design, develop, and deploy a chatbot assistant leveraging the Retrieval-Augmented Generation (RAG) architecture. This document provides a comprehensive overview of the development process for the chatbot, built with LangChain and LangServe libraries. The chatbot is enhanced using Ollama's pre-trained LLaMA2 model to ensure accurate and contextual responses.

Due to resource constraints encountered during deployment on cloud platforms such as Azure and Railway, the solution has been containerized for seamless local deployment using Docker. This approach ensures modularity, scalability, and ease of evaluation while providing a ready-to-run environment for future development or deployment efforts.

Project Objectives

1. **Develop a functional chatbot** capable of answering questions based on ***Promtior's*** website and PDF content, ensuring accuracy and relevance.
2. **Implement Retrieval-Augmented Generation (RAG) architecture** using ***LangChain*** and ***Ollama*** to enhance the chatbot's contextual understanding.
3. **Prepare a containerized solution** that can be deployed locally or adapted for other hosting platforms.

Architecture Overview

1. **Chatbot Service**:
   - Built with *FastAPI* and *LangChain*.
   - Handles user queries and integrates *Ollama's LLaMA2* model.
2. **Ollama Service**:
   - Provides the *LLaMA2* model for inference.
   - Preloaded in a *Docker* container for optimized performance.
3. **Local Deployment:**
   - Dockerized chatbot and Ollama services for local execution.
   - Ready-to-use Docker Compose configuration for streamlined setup.

Modular Architecture

- `main.py`: Orchestrates the application lifecycle and manages API routes using *FastAPI*.
- `model.py`: Encapsulates all logic for query embedding, retrieval, and language model interaction.
- `loaders.py`: Centralizes preprocessing logic for loading web and PDF data, ensuring extensibility for additional data sources.
- `config.py`: Manages environment-specific settings to simplify deployment and configuration changes.

Approach

To address this challenge, we followed a modular and scalable approach:

1. **Data Sources**
    - **Web Content**: Extracted relevant information from *Promtior's* website.
    - **PDF Content**: Parsed the "AI Engineer.pdf" document provided during the technical test.
2. **Preprocessing**

    `loaders.py` module:
    - Extracted data from both web and PDF sources.
    - Split the extracted text into manageable chunks using `RecursiveCharacterTextSplitter` for better embedding and retrieval performance.
3. **Vector Storage**
    - Embedded text chunks using `GPT4AllEmbeddings` from the LangChain library.
    - Leveraged **FAISS (Facebook AI Similarity Search)** to store these embeddings for efficient similarity searches.
4. **Query Processing**

    `model.py` module:
    - Handled incoming user queries by retrieving the most relevant text chunks from the *FAISS*-based vector store.
    - Integrated ***Ollama* LLM with *LLaMA2*** for language model inference.
    - Implemented a robust pipeline to ensure query relevance, avoiding unnecessary details.

5. **Response Generation**

Developed a custom **summarization method** in `model.py` module:

- Ensured responses remained concise, focused on user intent, and provided accurate information.
- Designed contextual prompts to guide the *LLaMA2* model toward generating precise answers.

Implementation

Development Process

1. Environment Setup

- **Technologies Used:**
  - **Python**: Core programming language for building and orchestrating the chatbot's logic.
  - **LangChain**: Implemented the Retrieval-Augmented Generation (RAG) pipeline.
  - **LangServe**: Simplified deployment of *LangChain* pipelines by exposing them as REST APIs.
  - **Docker**: Ensured containerization for both the chatbot and the *LLaMA2* model services.
  - **Railway**: Explored as an alternative cloud platform for hosting the Ollama service due to its flexible resource options.
  - **Azure App Services**: Initial deployment platform for the cloud-based chatbot and LLaMA2 services, later deprioritized due to resource constraints.

- **Setup Details:**
  - Defined and installed dependencies in `requirements.txt`.
  - Established a modular codebase (`main.py`, `model.py`, `loaders.py`, `config.py`) for better maintainability and scaling.
  - Updated `config.py` to handle containerized deployments and service-based URLs for inter-container communication.

2. Preloading the LLaMA2 Model

- Created a Docker image to preload LLaMA2 into an Ollama container.
- Steps:
  1. Pulled the base Ollama image.
  2. Used `ollama pull llama2` to download and configure the model.
  3. Tagged the preloaded image for deployment.

3. Integration of RAG Architecture with LangServe

- **Integrated LangServe:**
  - Used LangServe to expose the LangChain workflow (/invoke) as an API.
  - Configured LangServe to process user queries by chaining FAISS-based similarity search with LLaMA2 inference.

- **LangChain's RAG Pipeline**:
  - Employed RecursiveCharacterTextSplitter for preprocessing.
  - Used **GPT4AllEmbeddings** for generating vector embeddings.
  - Applied custom prompt engineering to improve response relevance.

4. Dockerization

- Built Docker images for both services:
  - **Ollama:** Image preloaded with LLaMA2.
  - **Chatbot:** Handles API requests and interacts with Ollama.

- Created a docker-compose.yml to orchestrate multi-container deployment, enabling developers to test the chatbot and Ollama integration seamlessly.

5. Automated Build with GitHub Actions

- Configured a GitHub Actions workflow to automate the creation and publishing of Docker images.
- Workflow triggers on every commit to the repository, ensuring the latest changes are built and pushed to Docker Hub.

Deployment with Docker on Local Machines

Prerequisites

- Ensure Docker and Docker Compose are installed on your machine.
- Verify that you have sufficient memory and CPU resources to run both the chatbot and the LLaMA2 model.

Steps to Deploy

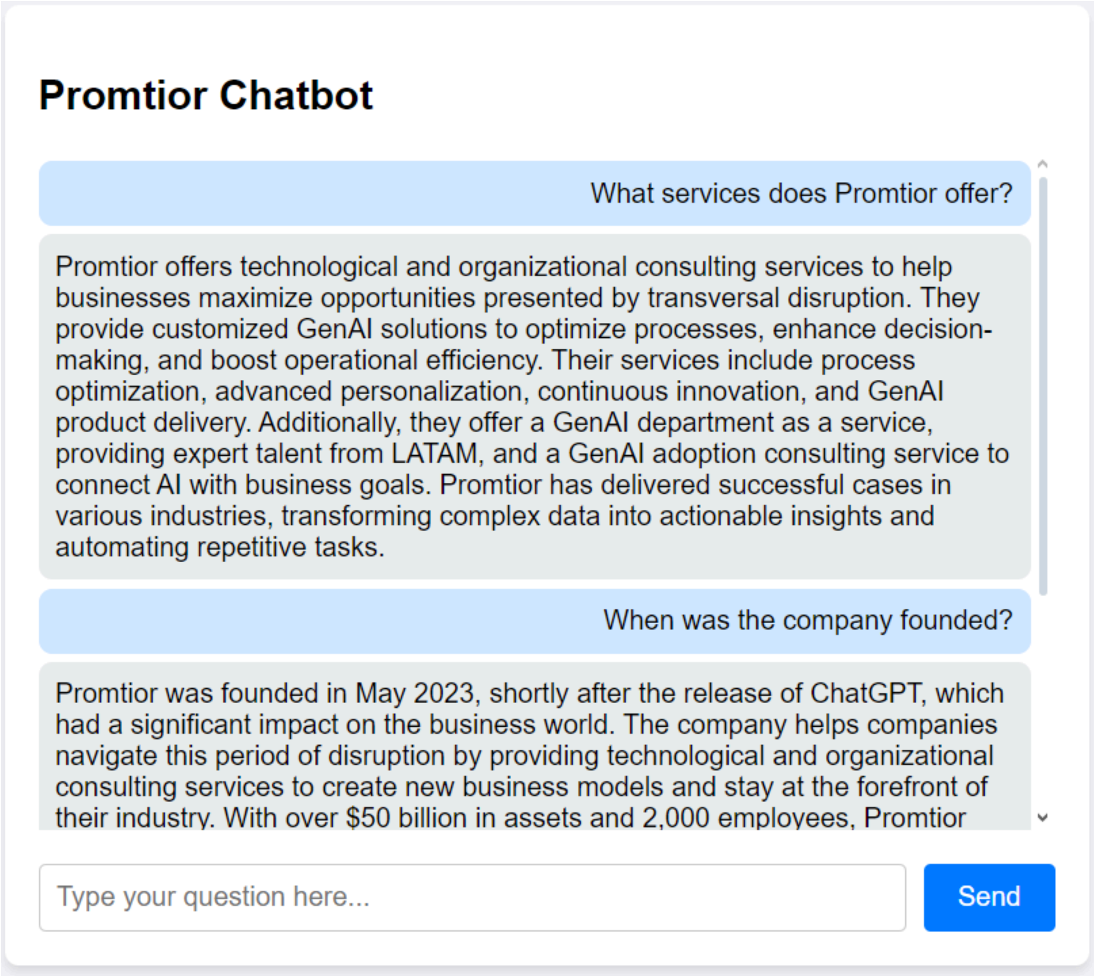
1. **Clone the Repository:** Clone the project repository to your local machine:
   - git clone https://github.com/nicofarizano/chatbot-RAG.git
   - cd <repository_directory>

2. **Start the Services:** Run the following command to start the containers using the provided docker-compose.yml file:
   - docker-compose up

3. **Access the Chatbot:** Once the containers are running, access the chatbot service at:
    - `http://localhost:11435`

Test and Verify

- Open a browser and navigate to the chatbot endpoint.
  Ask sample questions, such as:
    - `What services does Promtior offer?`
    - `When was the company founded?`



- **Shut Down the Services:** When you're done testing, stop and remove the containers using:
    - `docker-compose down`

Challenges Encountered

1. **No OpenAI Subscription:**
    - Initially, we considered using OpenAI's API; however, the lack of a subscription required us to pivot to alternative solutions.

- **Solution:** Adopted Ollama and Llama2, which provided robust language model capabilities without requiring a subscription.

2. **Python Version Compatibility and Dependency Conflicts:**
    - Several dependencies were incompatible with the latest version of Python, like chromadb and sentence-transformers, which caused significant installation issues.
    - **Solution:** Instead of downgrading Python, we opted for libraries and tools that supported the latest version, ensuring long-term maintainability. Switched to GPT4AllEmbeddings and ensured all packages were compatible.

3. **Limitation of Retrieved Documents**:
    - Retrieving too many documents caused irrelevant or excessive context to be included in the response.

```
HomeServicesCase StudiesAbout UsContact UsBlogAccelerate your GenAI adoptionWe boost operational efficiency in businesses with customized GenAI solutions, from discovery and development to implementation.



Our ClientsAbout Us Building Human-AI CollaborationWe help companies to achieve efficiency in the operative
business process with tailored GenAI solutions.Read more



Our method to Business Value with Gen AIIntegrate Generative AI across all operations, to lead the digital era. Our AI systems seamlessly optimize processes and decision-making, enhancing your organizatio
implementation, including predictive analytics,
intelligent automation,
among others.
 Read more
```

    - **Solution**: Limited the number of documents retrieved (k) to ensure only the most relevant information is processed.

```python
        # Recuperar los documentos más relevantes (limitar a 2)
        docs = vectorstore.similarity_search(question, k=2)
        print(f"\nDocumentos recuperados: {len(docs)}")
```

```
Escribe tu pregunta (o escribe 'salir' para terminar): What services does Promtioroffer?

Documentos recuperados: 2

Respuesta generada:
In November 2022, ChatGPT was released, causing a significant impact
and challenging previously unquestionable principles:
Creativity is an exclusively human trait.
The speed of these technological advancements has put
unprecedented pressure on leaders to incorporate AI into their
businesses.
In May 2023, Promtior was founded facing this context, where the key
question is: how to approach a scenario of transversal disruption and
maximize the opportunities it presents?
Through its technological and organizational consulting, Promtior offers
a way to generate new business models, answering this question and
bringing companies at the forefront of their sector.
About Promtior
https://promtior.ai/
Contact
Ignacio.acuna@promtior.ai
TECHNICAL TEST
```

4. **Summarization of Retrieved Text:**
    - Initially, the chatbot returned paragraphs verbatim from the sources, which were often too verbose or irrelevant like the response generated in the previous solution.
    - **New solution:** Implemented a summarization method to condense the retrieved text while retaining key information.

```python
# Resumir los documentos recuperados
print("\nRespuesta generada: ")
summarize_documents(llm, docs, question, max_words=200)
```

```python
def summarize_documents(llm, docs, question, max_words=200): # Resume los documentos recuperados utilizando el modelo de lenguaje, enfocado en la pregunta inicial.
    combined_text = " ".join([doc.page_content for doc in docs])
    summary_prompt = (
        f"A continuación, tienes un texto. Responde a la siguiente pregunta basándote en el texto. "
        f"Limita tu respuesta a no más de {max_words} palabras.\n\n"
        f"Pregunta: {question}\n\n"
        f"Texto:\n{combined_text}"
    )
    summary = llm.invoke(summary_prompt)
    return summary
```

```
Escribe tu pregunta (o escribe 'salir' para terminar): What services does Promtior offer?

Documentos recuperados: 3

Respuesta generada:
In November 2022, ChatGPT was released, revolutionizing the tech industry. Promtior was founded in May 2023 to address the challenge of integrating AI into businesses. Through technological and organization
al consulting, Promtior offers customized GenAI solutions to maximize opportunities and boost operational efficiency. They help clients achieve business value through seamless optimization of processes and
decision-making, enhancing organization capabilities in various areas. Their services include GenAI product delivery, department as a service, adoption consulting, and more.
```

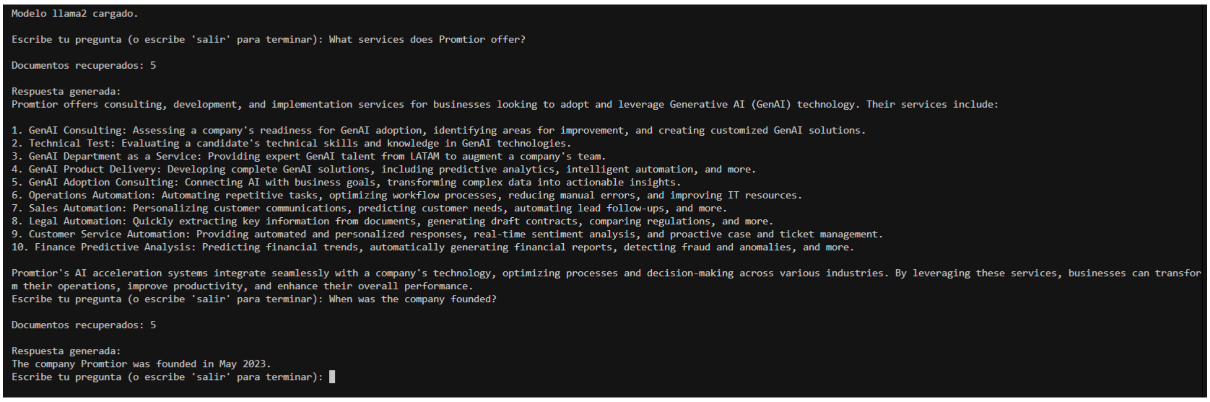
5. **Maintaining Focus on the Question:**
    - The chatbot sometimes lost focus and provided general summaries instead of directly addressing the user's question.
    - **Solution:** Tailored the language model's prompts to explicitly prioritize the question, ensuring answers were concise and directly relevant.

```python
def summarize_documents(llm, docs, question, max_words=200): # Resume los documentos recuperados utilizando el modelo de lenguaje, enfocado en la pregunta inicial.
    combined_text = " ".join([doc.page_content for doc in docs])
    summary_prompt = (
        f"A continuación, tienes un texto. Responde a la siguiente pregunta basándote en el texto. "
        f"Prioriza información relevante y no incluyas contenido que no esté relacionado con la pregunta. "
        f"Limita tu respuesta a no más de {max_words} palabras.\n\n"
        f"Pregunta: {question}\n\n"
        f"Texto:\n{combined_text}"
    )
    summary = llm.invoke(summary_prompt)
    return summary
```

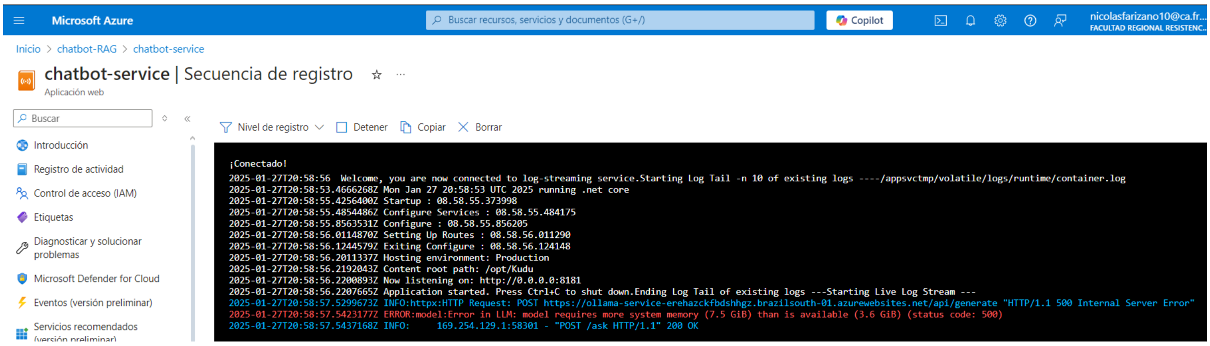
6. **Misidentification of Data:**
    - The chatbot initially confused the founding date of Promtior (May 2023) with the release date of ChatGPT (November 2022) due to overlapping context in the retrieved documents.

```
Escribe tu pregunta (o escribe 'salir' para terminar): When was the company founded?

Documentos recuperados: 4

Respuesta generada:
According to the text, Promtior was founded in November 2022.
Escribe tu pregunta (o escribe 'salir' para terminar): When was Promtior founded?

Documentos recuperados: 4

Respuesta generada:
According to the text, Promtior was founded in May 2023.
```

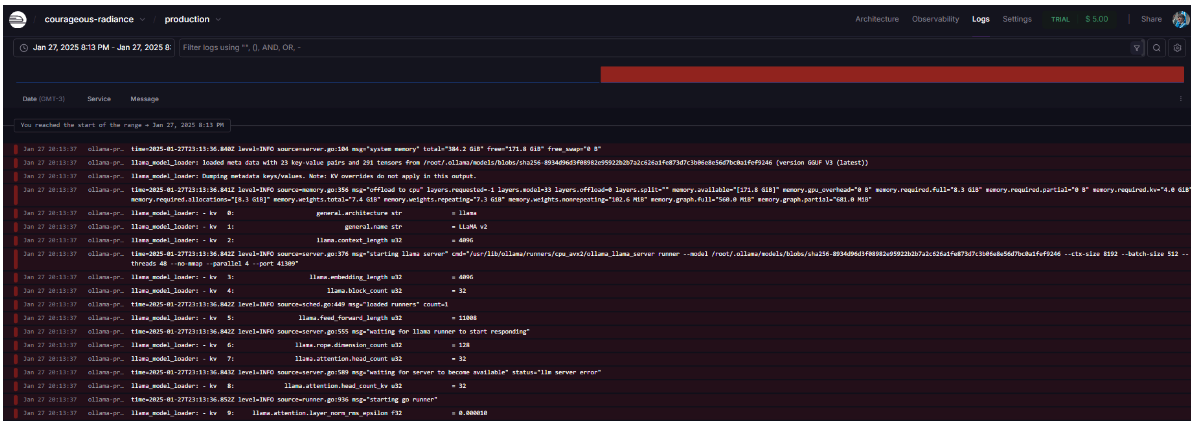
- **Solution**: Adjusted the k parameter (number of documents retrieved) during the similarity search to refine the context. By testing with various values, we identified the optimal k that provided accurate and consistent answers across multiple queries.



7. **Insufficient Resources on Azure and Railway:**
   - When deploying the Ollama service with the preloaded LLaMA2 model on Azure App Service, the instance lacked sufficient memory to load the model. The LLaMA2 model required at least 7.5 GiB of system memory, but the Azure App Service plan provided only 3.6 GiB. This caused the chatbot to fail when trying to generate responses, with errors indicating insufficient system memory.



   - Similarly, Railway's standard plans lacked the necessary resources for stable operation of the model, resulting in process termination.



- **Solution**: Given these constraints, I concluded that completing the challenge without investing additional resources in alternative technologies or services was not feasible. Instead, I ensured the entire project is fully containerized and prepared for local deployment using Docker. This approach maintains the modularity and scalability of the architecture, allowing future developers or teams to easily run and evaluate the solution locally or deploy it to a more suitable platform with sufficient resources.

- The Azure App Services and Railway deployments remain accessible for review. Below are the links to the services:
  - Azure App Services - Chatbot Service: https://chatbot-service-faf8h3dtdna8gcey.brazilsouth-01.azurewebsites.net
  - Azure App Services - Ollama Service: https://ollama-service-erehazckfbdshhgz.brazilsouth-01.azurewebsites.net
  - Railway - Promtior Chatbot: https://promtior-chatbot-production.up.railway.app
  - Railway - Ollama Preloaded: https://ollama-preloaded-production.up.railway.app

---

Component Diagram

Below is a detailed description and diagram illustrating the interactions between components in the solution:

Description

User Query:

The user sends a query to the chatbot API endpoint via HTTP POST requests.

Query Processing:

- The query is processed by the chatbot service.
- The FAISS vector store retrieves the most relevant text chunks based on query similarity.
- The retrieved chunks are passed to the LLaMA2 language model for further processing.

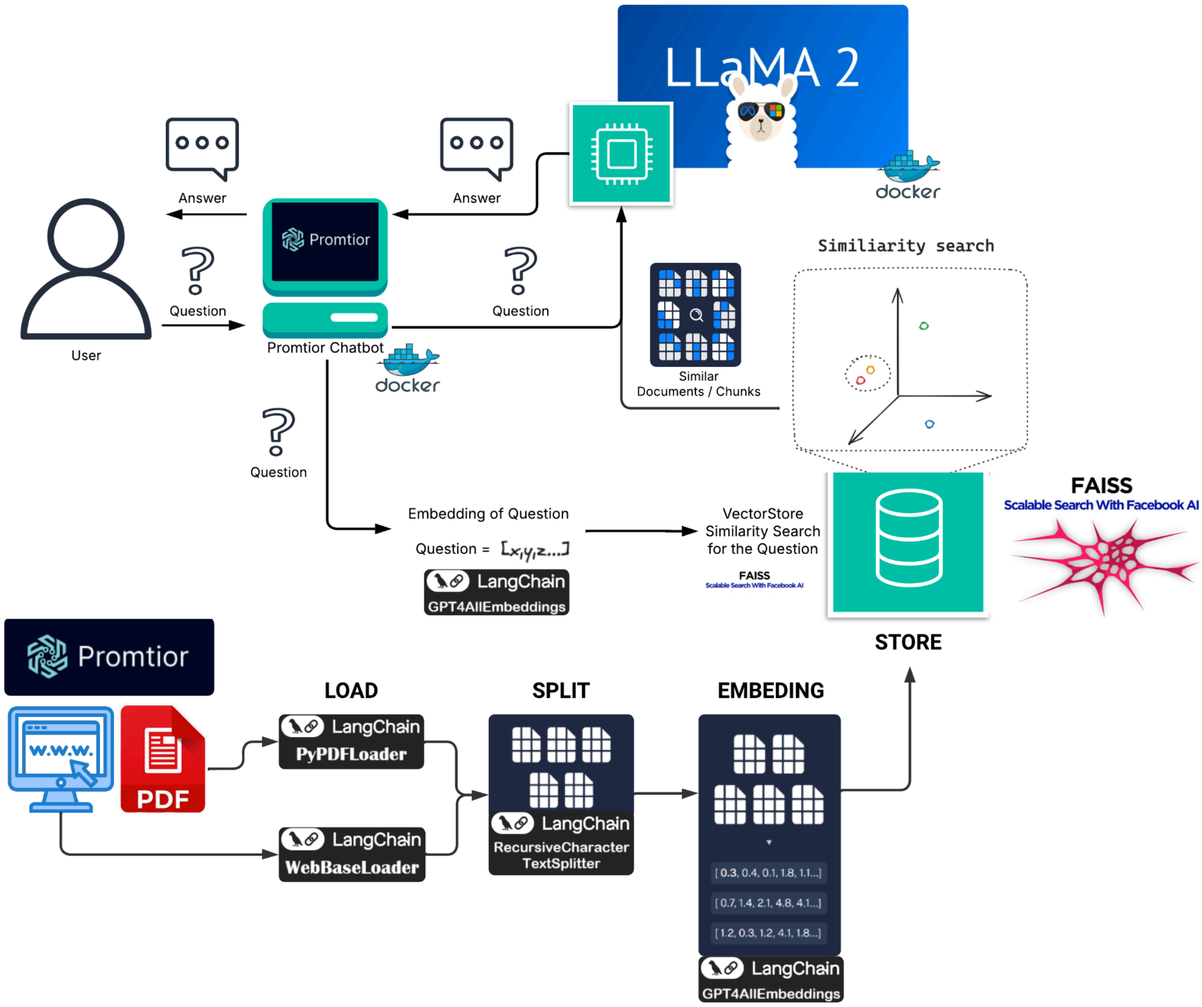
Response Generation:

The language model uses the query and retrieved context to generate a concise and accurate response.

Response Delivery:

The chatbot service formats the generated response and sends it back to the user via the API.

Diagram



Conclusion

This project showcases the development of a modular, containerized chatbot using the Retrieval-Augmented Generation (RAG) architecture. By leveraging Docker, we ensured that the solution can be deployed easily on local machines or cloud platforms. While Azure and Railway were explored for cloud deployment, technical limitations and resource constraints made local Docker deployment the most practical option for this project.

Despite not achieving deployment on Azure or Railway, the process of seeking alternatives allowed me to deepen and refine my knowledge of containerization, cloud infrastructure, and deployment strategies. These learnings have enhanced my technical expertise and will undoubtedly contribute to the success of future projects.

References

- LangChain Documentation: For implementing the RAG pipeline and vector store integration.
- FAISS Documentation: For efficient similarity searches.
- Ollama Documentation: For leveraging the LLaMA2 language model.
- Docker Documentation: For containerizing and orchestrating services.