Building RAG Chatbots for Technical Documentation

Advanced Topics on Machine Learning 2024/2025

Group P3_E

Athos Freitas – up202108792

Félix Martins – up202108837

Luís Du – up202105385

Master in Artificial Intelligence

Professor: Pedro Gabriel Dias Ferreira

3rd November, 2024

Introduction

This report aims to explore the potential of LLMs on the analysis of data and information, specifically through Retrieval Augmented Generation (RAG) using *LangChain* to create a chatbot for answering questions about medical documentation.

The project consists of the following stages:

1. **Context Analysis:** Before initiating indexing, a comprehensive analysis of the document will be conducted to establish an understanding of content structure, key entities and relationships.
2. **Indexing:** A robust pipeline will be established for data ingestion and indexing, which includes document splitting, embedding generation and storage.
3. **Retrieval and Generation:** The core RAG chain, combining real-time retrieval of relevant data from the index, then passes that to the model. The steps include: retriever creation, LLM initialization and prompting, defining and invoking the RAG Chain.
4. **Chat history:** The prompt can also include the chat history of the conversation, providing more context to the LLM.
5. **Improving basic RAG architecture:** In basic RAG, the user question is used to directly query the database. However, an intermediate step via the LLM before querying the database can improve the performance of the retriever and therefore the RAG model.
6. **Interface:** A chatbot interface for the end-user using the *Chainlit* library. Features include showing the sources retrieved, and changing model configuration settings.
7. **Evaluation:** Finally, appropriate evaluation metrics will be implemented to assess the effectiveness of the chatbot.

Tools and resources

The development of this project relies on several key tools and resources.

- **Programming language:** *Python* was chosen for its versatility, strong support for machine learning and natural language processing, making it well-suited to build the RAG architecture.
- **Framework:** *LangChain* is used as the main framework, offering essential components for creating the RAG pipeline. *LangChain* provides tools to facilitate the RAG architecture, including utilities for document chunking, embedding storage, and retrieval mechanisms. Additionally, *LangChain* includes evaluators and benchmarking tools to measure the performance and effectiveness of the chatbot, allowing for continuous improvements based on metrics such as accuracy and relevance.
- **LLM Provider:** The project uses *Gemini* as the LLM provider, accessing the *"Gemini 1.5-flash"* language model through its API. This model generates responses based on the retrieved content, making it integral to the generation aspect of the RAG pipeline. For the

embedding stage, the *"Text Embeddings 004"* model from Gemini is employed, which generates high-quality vector embeddings of the document segments to facilitate accurate and contextually relevant retrieval.

- **Interface framework**: *Chainlit* is used to develop the interface. It provides an easy-to-use event-driven design with integrations with multiple LLM Orchestration Frameworks.

1. Context Analysis

The chosen document, **Regulation (EU) 2017/745 of the European Parliament and of the Council**, outlines crucial regulations governing medical devices within the European Union.

A key focus is the placement of medical devices on the market, where the regulation establishes stringent requirements that manufacturers must meet to legally distribute their products in EU member states. Furthermore, another central theme is the standards for quality and safety in device development, which sets out comprehensive criteria for risk management, clinical evaluation, and post-market surveillance.

Regulation (EU) 2017/745 is an extensive document, comprising 232 pages and approximately 100,000 words. Its structured layout facilitates a detailed exploration of the requirements for medical devices within the European Union: it is divided into 10 chapters, each addressing specific aspects of medical device regulation. Within these chapters are 123 articles, which provide detailed guidelines and legal requirements, complemented by 17 annexes, offering further clarification and technical specifications.

2. Indexing

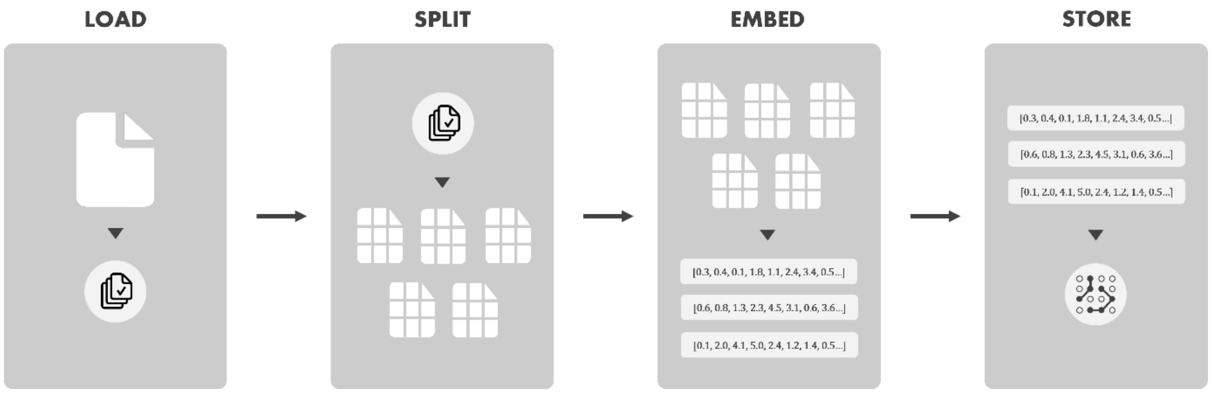


Figure 1. Indexing flow

2.1. Document Splitting

Since the source document is in pdf format, the **PDFPlumberLoader** was utilized for document splitting in combination with the **RecursiveCharacterTextSplitter**. A default *chunk_size* of 1000 characters was set with a *chunk_overlap* of 200 characters to ensure a more efficient context window, maintaining continuity between chunks. Additionally, document metadata were extracted by enabling the *add_start_index* flag.

2.2. Generate and store the embeddings

As previously mentioned, the *"Text Embeddings 004"* model from Gemini is employed for the embedding generation, with the *task_type* set to "retrieval_document" to optimize embeddings for retrieval tasks.

The embeddings are then stored in vector format using **Chroma** library, creating an efficient index that enables quick and relevant document retrieval during the query process.

3. Retrieval and Generation

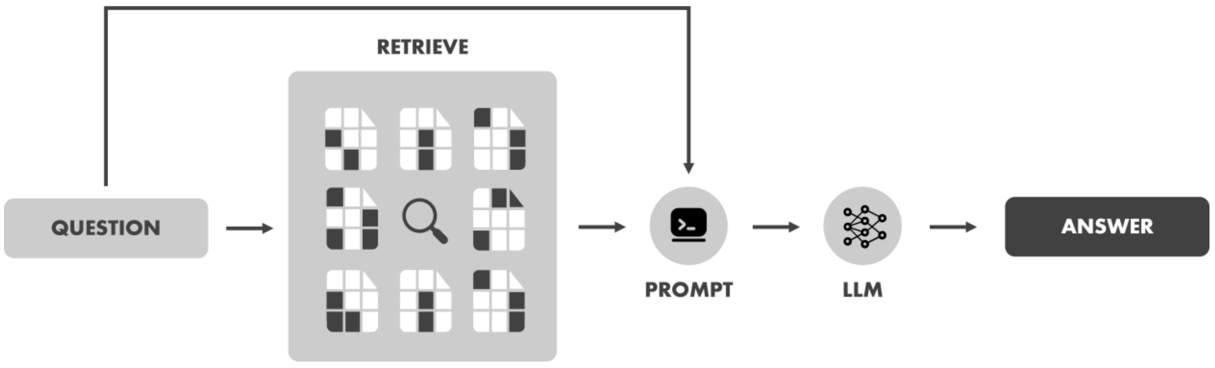


Figure 2. Rag Chain

3.1. Retrieve

The retriever component enhances the input prompt by adding relevant context before passing it to the LLM. It does this by querying the vector database for embeddings that are semantically similar to the input prompt.

Based on the similarity results, the **top *k* most relevant** text segments from the document are retrieved, ensuring that the LLM receives contextually accurate information to generate a more precise and informed response.

3.2. Prompt

Using the **PromptTemplate** library from *LangChain*, we can define a structured format for the prompts sent to the LLM. This allows us to tailor the input to include essential context and instructions via placeholders, ensuring that the model understands the specific requirements of the task.

3.3. LLM

The LLM utilized in this project is **Gemini 1.5 Flash**, recognized as Google Gemini's fastest multimodal model. It boasts an impressive context window of 1 million tokens, allowing for comprehensive understanding and processing of extensive inputs.

The model operates at a speed of 1 million tokens per minute (TPM) and can handle up to 15 requests per minute (RPM), with a daily limit of 1,500 requests per day (RPD). This highperformance capability ensures efficient processing and rapid response generation, making it well-suited for applications that require quick and reliable interactions.

3.4. Generate

Putting it all together, we can create a chain that takes a question, retrieves relevant documents, constructs a prompt, passes it into a model, and parses the output.

```
rag_chain = (
    {"context": retriever | format_docs, "question": RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)
```

4. Chat History

LangChain's memory management capabilities were used to implement the chat history. This history includes the full conversation between the user and the chatbot previous to the current question. It is sent to the LLM along with the context to generate the next answer.

However, the history does not include the context retrieved from the previous questions, only the actual questions from the user and the answers from the LLM.

5. Improving basic RAG architecture

Basic RAG architecture does not allow questions that do not directly reference the document. This means that questions referencing, for example, the chat history or the number of pages, will result in the LLM responding that the context does not provide information about the question. We can improve this by asking the LLM to create a question for searching the database, based on the user question and the chat history. Additionally, using SelfQueryRetrieval, we can even use the LLM to filter specific chunks of the documents based on their metadata, in this case the page number.

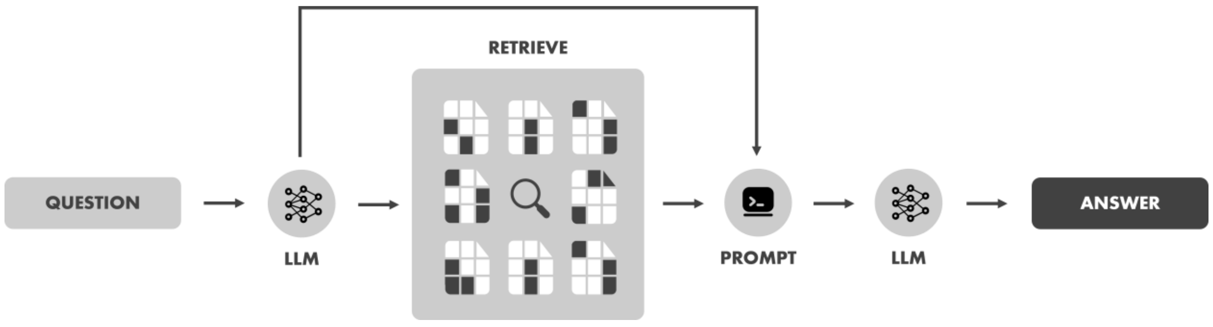


Figure 3. Improved RAG architecture

6. Interface

The chosen library for the interface and interaction with end-users is *Chainlit*. It provides an easy-to-use chatbot interface and integrations with multiple LLM Orchestration Frameworks, namely *LangChain*. It is based on an event driven paradigm, allowing us to specify functions when specific events happen. This library requires .py files, so the code had to be adapted to suit this organization. This library supports multiple useful features for interfacing with chatbots. It also includes a small README that the users can access to get more information about the project.

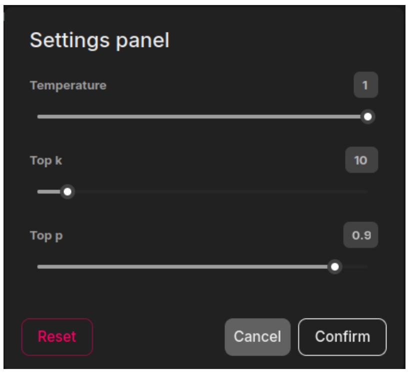
6.1. Multiple users

Since multiple users could use this application at the same time, some information needs to be stored across distinct user sessions. The library allows this by setting variables for each user session. For example:

```
cl.user_session.set("question", Model())
```

6.2. Settings

We have added some configuration settings that the user can customize, specifically for the model's generation process.


Settings panel

Temperature 1

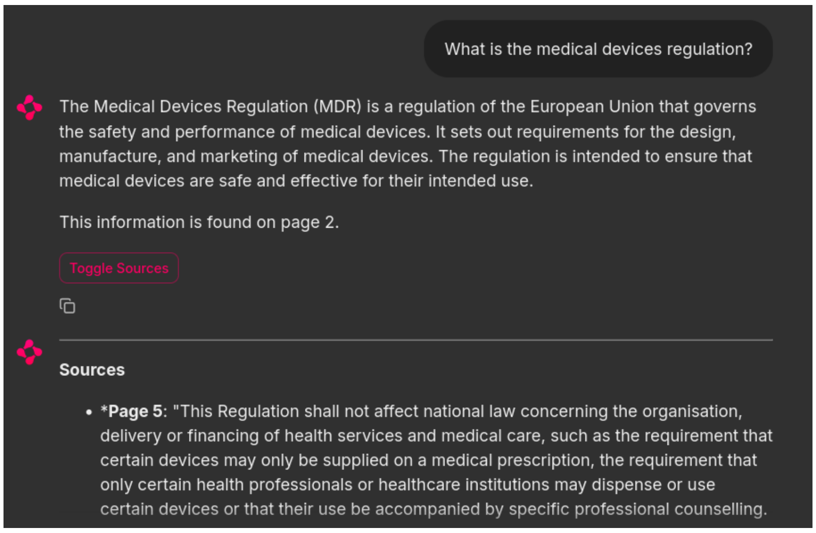
Top k 10

Top p 0.9

Figure 4. Configuration settings

6.3. Referencing sources

Retrieval Augmented Generation is heavily based on the document retrieval process. Since the retrieved information is so important, we allow the users to toggle showing the sources used by the model, for each question.


What is the medical devices regulation?

The Medical Devices Regulation (MDR) is a regulation of the European Union that governs the safety and performance of medical devices. It sets out requirements for the design, manufacture, and marketing of medical devices. The regulation is intended to ensure that medical devices are safe and effective for their intended use.

This information is found on page 2.

Sources

- ***Page 5**: "This Regulation shall not affect national law concerning the organisation, delivery or financing of health services and medical care, such as the requirement that certain devices may only be supplied on a medical prescription, the requirement that only certain health professionals or healthcare institutions may dispense or use certain devices or that their use be accompanied by specific professional counselling.

Figure 5. Referencing sources

7. Evaluation Metrics

Benchmarks would provide the most accurate and objective way to assess the system's performance, allowing for a direct comparison of generated outputs against established standards. However, in the absence of such benchmarks, we explored alternative evaluation approaches:

- **Parameter Tuning:** adjusting parameters such as *temperature* and *top-p* showed minimal impact on the quality of generated outputs, with only extreme variations in *temperature* producing noticeable changes. This suggests that fine-tuning parameters alone may not significantly enhance response quality.
- **Model Comparison:** When using the GPT-2 model, responses were largely nonsensical and lacked relevance, indicating that the Gemini 1.5 Flash model performs significantly better in this context.
- **Prompt Tuning:** Finally, we experimented with different prompt formulations, by removing restrictions that prevent the LLM from answering when the answer is not found in the document. This led to the model generating responses regardless of document content, increasing the risk of hallucinations. These findings suggest that carefully designed prompts are crucial to limiting hallucinations and guiding the LLM toward accurate, context-based responses.

Conclusions

This project demonstrated the potential of using Retrieval-Augmented Generation to build a chatbot capable of answering complex, document-based questions about a medical article. During the project, we explored each of the steps required to build a RAG model, from document loading and splitting to embedding generation, retrieval and response generation.

Ultimately, the integration of Retrieval Augmented Generation enables more focused prompts and reduces the risk of hallucinations, which is crucial for technical documentation.

Bibliography

[1] The European Union Medical Device Regulation - Regulation (EU) 2017/745 (EU MDR). [Online; accessed 3rd October 2024].
https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A32017R0745

[2] How to build LLM Application with LangChain. [Online; continuously accessed throughout the project].
https://www.datacamp.com/tutorial/how-to-build-llm-applications-with-langchain

[3] Build a Retrieval Augmented Generation App. [Online; continuously accessed throughout the project].
https://python.langchain.com/docs/tutorials/rag/#retrieval-and-generation-generate

[4] Embeddings in Gemini API. [Online; continuously accessed throughout the project].
https://ai.google.dev/gemini-api/docs/embeddings?hl=pt-br

[5] Google Deepmind Gemini 1.5 flash. [Online; continuously accessed throughout the project].
https://deepmind.google/technologies/gemini/flash

[6] PDFPlumberLoader documentation. [Online; continuously accessed throughout the project].
https://api.python.langchain.com/en/latest/document_loaders/langchain_community.document_loaders.pdf.PDFPlumberLoader.html

[7] Chainlit documentation. [Online; continuously accessed throughout the project].
https://docs.chainlit.io/get-started/overview