Building RAG Chatbot for Technical Documentation

Guilherme Vaz, Sebastião Santos Lessa
Group $P3_C$
Faculdade de Ciências da Universidade do Porto
Faculdade de Engenharia da Universidade do Porto

November 2024

1 Introduction

The evolution of large language models (LLMs) has opened new possibilities in natural language processing (NLP), particularly with the advent of retrieval-augmented generation (RAG). This technique enhances the model's ability to retrieve relevant information from a database and generate informed, context-rich responses. The integration of retrieval mechanisms and generative models provides a significant leap in creating applications that are not only responsive but also contextually aware.

This project centers on utilizing the Haystack framework, a flexible and comprehensive open-source library in Python, renowned for its capabilities in constructing RAG pipelines and combining model retrieval with generation functionalities. Haystack's modular architecture supports seamless integration with other technologies, such as those offered by Hugging Face, enabling the implementation of sophisticated search and generation processes. To enhance model interactions, the Hugging Face inference client was incorporated, improving API communication and supporting efficient interaction between Python applications and hosted models. For testing, we used the complete text of the `European Union Medical Device Regulation - Regulation (EU) 2017/745 (EU MDR)` to evaluate the system's effectiveness and accuracy in handling complex, regulatory content.

The core LLM used is `meta-llama/Llama-3.2-1B-Instruct`, known for its instruction-following proficiency and balanced performance, offering an optimal blend of response quality and computational efficiency. Additionally, we used Few-Shot Example Prompts within system prompts to guide the model's responses. This technique involved providing multiple examples within the system prompts, allowing the LLM to better understand the expected response format and improve contextual accuracy. Nuanced prompt refinements were also added in the final user prompt to increase relevance and depth in responses.

To demonstrate the practical utility of these components, we developed a full-stack web application using Flask for server-side operations, paired with HTML and CSS to create a user-friendly browser interface. This structure enabled real-time user interaction with the RAG system, highlighting the project's potential in practical deployment scenarios and showcasing the effective combination of modern frameworks for local, accessible solutions.

2 Methodology

2.1 Processing Documents Pipeline

The document processing pipeline is designed to optimize the way documents are ingested, segmented, and stored for efficient retrieval and analysis. The pipeline operates as follows:

- **Document Segmentation Using LLM Sherpa**: Unlike the default document splitting methods used by frameworks like Haystack or LangChain, which may randomly split documents or rely on fixed-size chunking, this pipeline leverages the LLM Sherpa model. This model identifies and segments the text from one header to the next, preserving the document's logical structure and ensuring that each section is contextually coherent.
- **NER Classification for Metadata Generation**: Once the document is divided into sections, each segment undergoes Named Entity Recognition (NER) using a SpaCy model. This step extracts entities and classifies them to enrich the content with relevant metadata. These metadata tags enhance the searchability and contextual understanding of the document, enabling more refined and precise search results.
- **Keyword-Based Storage**: The sections are stored in a database indexed by keywords, supporting more efficient retrieval based on thematic relevance and allowing for targeted search queries that yield high-value responses.
- **Segmenting Sections into Sentences**: To enhance the granularity and effectiveness of the vector database, each section is split into individual sentences. Storing large sections as a single vector representation can dilute semantic specificity and reduce search precision. By breaking sections down into sentences, the pipeline ensures that each entry in the vector database maintains fine-grained semantic resolution.
- **Storage in Vector Database**: The processed sentences, enriched with metadata and keywords, are stored in the vector database. This enables semantic search capabilities, where the retrieval process leverages the embeddings of each sentence to find matches based on conceptual similarity rather than exact keyword matching, resulting in nuanced and contextually relevant query responses.

Contextual Note: Initially, an intention classification step was part of the pipeline, designed to categorize sections based on intent. However, after comparing results, it was determined that this step did not add significant value. Therefore, it was removed to streamline the process and avoid unnecessary complexity.

The final pipeline architecture is depicted in Figure 1, which illustrates the flow from document input through segmentation, classification, and storage in both keyword and vector databases.

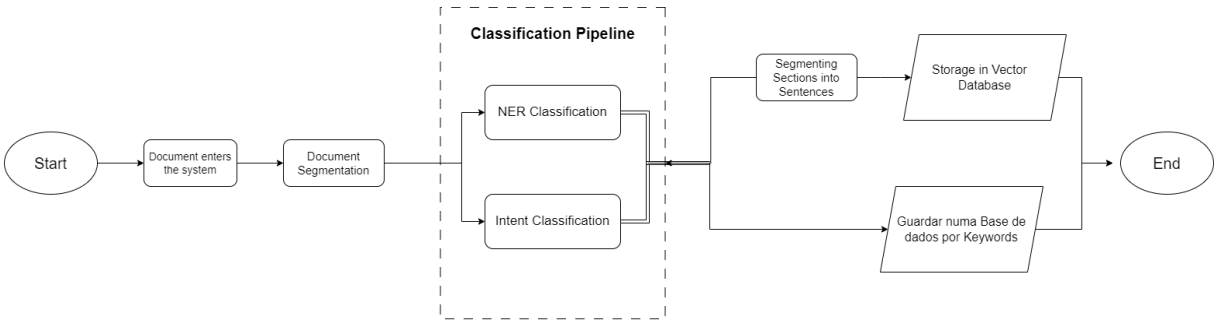

Figure 1: Processing Documents Pipeline.

2.2 Prompt Re-engineering

The pipeline modifies the user prompt, adapting it into specific prompts optimized for different types of searches, including keyword-based and vector-based searches. After these prompts are created, the pipeline performs the search and retrieves the most relevant sections to answer the query. The process follows these steps:

1. **User Prompt Transformation**:

   When a user prompt is received, it is processed by the LLM, which then generates specific multi sub-prompts tailored for both vector and keyword-based searches. This bifurcated approach ensures that the prompts aligns with the requirements of each search method and maximizes the retrieval capabilities of the system.

   1.1 **Prompts for Vector Database Search**: The generated prompts is sent to the vector database, where a semantic search is conducted. This type of search evaluates the conceptual similarity between the prompts and stored data, providing results that are contextually relevant.

   1.2 **Prompts for Keyword Database Search**: In parallel, the LLM also generates prompts suited for keyword-based searching. This search method looks for exact or thematic keyword matches in the keyword database, providing high-precision results for direct term correlations.

2. **Dual Search Execution**: Using these specialized prompts, the pipeline conducts multiple searches in both the keyword-based and vector-based databases. The keyword searches focus on finding documents through direct term matches, while vector searches retrieve contextually aligned results by assessing semantic similarities.

3. **Result Concatenation**: After completing both searches, the pipeline concatenates the results from the keyword and vector searches. This combined output leverages the precision of keyword matching and the depth of semantic understanding from vector search, providing a comprehensive set of results. The concatenation process is enhanced using *'JoinDocuments(join mode="distribution based rank fusion")'*. This method merges the outputs by evaluating the distribution of document ranks from both searches, ensuring that the final result set reflects a balanced contribution from each method. By considering the rank positions and their distribution, this approach maximizes the relevance and diversity of the final response, yielding richer and more contextually accurate outputs.

The overall architecture of this querying process is shown in Figure 3, illustrating the flow from the initial user query to the final result compilation.

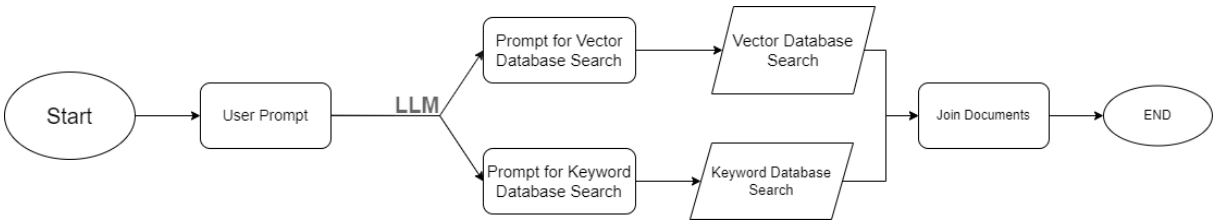

Figure 2: Prompt Re-engineering

2.3 Querying the LLM

This stage of the pipeline involves the simplest yet critical task: submitting the processed user prompt and the retrieved document sections to the LLM to generate a comprehensive and contextually relevant response.

After the user prompt has been refined and the relevant document sections have been obtained through keyword and vector-based searches, the system forwards these elements to the LLM. The LLM, equipped with advanced language understanding and generation capabilities, synthesizes an output that integrates the user's query with the contextual information provided by the retrieved sections. This ensures that the response is not only accurate but also context-aware, leveraging the depth of information found in the database. To maintain efficiency and prevent overwhelming the model with irrelevant information, we provide context up to a limit of 1000 tokens.

The key to this stage is the interaction between the LLM and the curated input data. By aligning the user's intent with supporting sections, the LLM can produce responses that are richer and more precise, demonstrating an effective combination of retrieval and generation.

This approach highlights the simplicity of the final query phase while showcasing the effectiveness of the prior steps. The resulting output reflects the LLM's ability to draw from pre-retrieved, contextually relevant data to enhance response accuracy and informativeness.

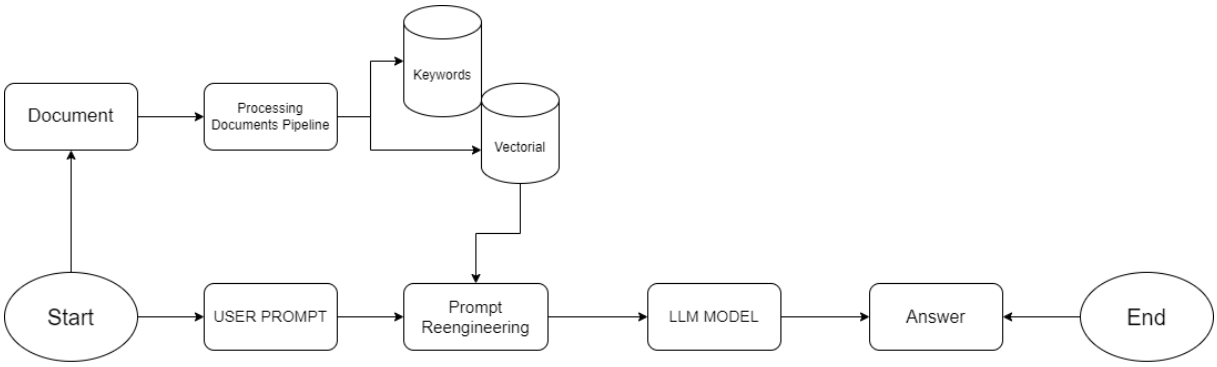

Figure 3: Our system architecture.

3 Conclusion

In developing this RAG-based chatbot, we demonstrated how structured document processing, the use of few-shot example prompts, and dual-database querying can result in high-quality, contextually relevant responses, even for dense regulatory documents like the European Union Medical Device Regulation (EU MDR). The few-shot examples within system prompts were particularly effective in refining the LLM's output by providing illustrative guidance for response structure and detail. This approach enabled the model to respond accurately to complex queries, showing a sophisticated understanding of nuanced regulatory content. With this adaptable architecture, the chatbot is well-suited for a variety of applications, from regulatory compliance to technical documentation, effectively enhancing accessibility to intricate information in real time.