Mr. HelpMateAI

A robust generative search system capable of effectively and accurately answering questions from a policy document using the RAG framework.

Project Background

Retrieval-augmented generation (RAG) is an AI framework for improving the quality of LLM generated responses by grounding the model on external sources of knowledge to supplement the LLM's internal representation of information. Implementing RAG in an LLM based question answering system has two main benefits: It ensures that the model has access to the most current, reliable facts, and that users have access to the model's sources, ensuring that its claims can be checked for accuracy and ultimately trusted.

Objective

The goal of the project is to build a robust generative search system capable of effectively and accurately answering questions from a policy document using the RAG framework described above

Dataset

The data used for the project is a single long life insurance policy document

Design

The project is built using three layers namely:

- Embedding Layer
- Search layer
- Generation Layer

Implementation

Embedding Layer

Here the PDF document needs to be effectively processed, cleaned, and chunked for the embeddings. We have used the pdfplumber library to extract text/tables etc from multiple pdfs. In the pdf pre-processing we have removed the pages where text length is less than 10. The chunking strategy used is the Page-wise chunking as we can observe that mostly the pages contain few hundred words, maximum going upto 5000. So, we don't need to chunk the documents further; we can perform the embeddings on individual pages. This strategy makes sense for 2 reasons:

1. The way insurance documents are generally structured, we will not have a lot of extraneous information in a page, and all the text pieces in that page will likely be interrelated.
2. We want to have larger chunk sizes to be able to pass appropriate context to the LLM during the generation layer.

Search Layer

Here we have followed the following steps:

- Generate and Store Embeddings using OpenAI and ChromaDB
- We have embedded the pages in the dataframe through OpenAI's text-embeddingada-002 model, and store them in a ChromaDB collection.
- We have implemented the cache mechanism for faster retrieval of queries
- We have implemented the re-ranking block which enhances the quality of search results. Re-ranking the results obtained from our semantic search can sometime significantly improve the relevance of the retrieved results. This is often done by passing the query paired with each of the retrieved responses into a cross-encoder to score the relevance of the response with respect to the query.
- We have used the 'cross-encoder/ms-marco-MiniLM-L-12-v2' cross encoder from sentence-transformer library

Generation Layer

Here we followed the following steps:

- Once we have the final top search results, we can pass it to an GPT 3.5 along with the user query and a well-engineered prompt, to generate a direct answer to the query along with citations, rather than returning whole pages/chunks.
- We have written an exhaustive prompt with clear instructions and passed the user query and top searched results to the LLM and generated a response.

Testing of the RAG system with three user designed queries.

The three queries used to test the system are as follows:

1. What is the name of the policyholder and when was this policy issued?
2. What are the premium rates for the members insured?
3. What are the conditions for premium rate changes?

Screenshots for the above user queries from the Search and generation layer are attached in a separate document.

Challenges Faced

- **Data quality and preprocessing:** Extracting relevant information from complex insurance documents proved challenging due to varied text structures.
- **Chunking strategies:** Optimising chunk size to maintain context without losing coherence was critical but difficult.

- **Query understanding and matching:** Designing relevant queries that required sophisticated understanding and reasoning posed a significant challenge.

Lessons learned

- **Efficient document processing**: Utilizing tools like pdfplumber is crucial for handling complex PDF documents efficiently.
- **Cache management:** Implementing the efficient caching strategy significantly improves system performance.