Mr.HELPMATE AI PROJECT REPORT

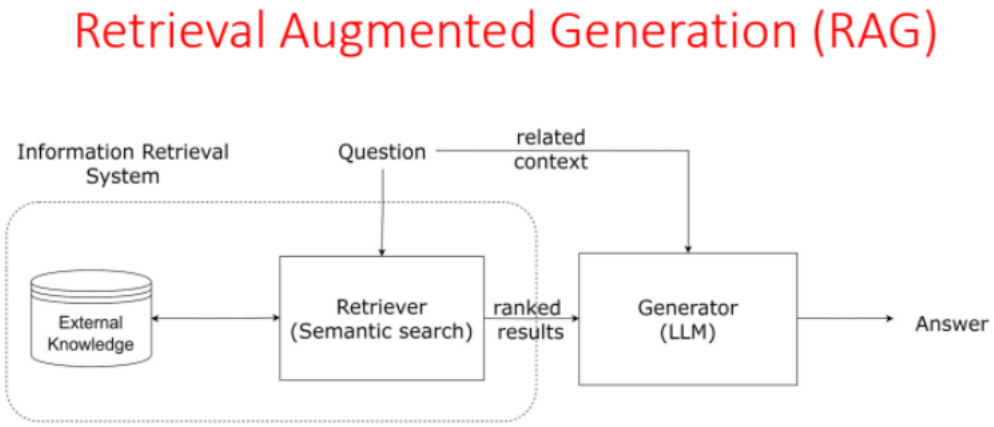
PREPARED BY MYTHILI KANDULA

Project Objective:

The goal of the project is to build a robust generative search system capable of effectively and accurately answering questions from a group policy document "Principal-Sample-Life-Insurance-Policy.pdf"

Design:

I have used Retrieval Augmented Generation (RAG) system



Initially followed the below 3 steps:

- We **retrieved** the information from an external source,
- **Augmented** our query with the relevant information prompt,
- and finally we **generated** the relevant response

Approach:

In order to build an effective search system, implemented three layers.

1. *The Embedding Layer*: The PDF document was effectively processed, cleaned, and chunked for the embeddings. Used text-embedding-ada-002 model
2. *The Search Layer*: Designed 3 queries based on the document. Embeded the queries and searched ChromaDB vector database against each of these queries. Implemented cache mechanism. Finally implemented re-ranking block.

3. *The Generation Layer*: Built a robust prompt template that can give the search results.

Implementation:

Detailed implementation steps as follows:

Data Processing Steps:

- Used good parsers that can load the data from your documents effectively.
- Extract text from a PDF file
- Check the length of all the texts as there might be some empty pages or pages with very few words that we can drop
- Retain only the rows with a text length of at least 10.
- Store the metadata for each page in a separate column.

This concludes the chunking aspect also, as we can see that mostly the pages contain few hundred words, maximum going upto 1000. 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, you 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.

Generate and Store Embeddings using OpenAI and ChromaDB:

Steps include:

- Import the OpenAI Embedding Function into chroma
- Define the path where chroma collections will be stored
- Call PersistentClient()
- Set up the embedding function using the OpenAI embedding model text-embedding-ada-002
- Create an empty collection name “RAG_on_RhodeIslandGroupPolicy“
- Add the documents and metadata to the grouppolicydata collection along with generic integer IDs.
- Create cache_collection

Semantic Search with Cache:

We have created 2 collections

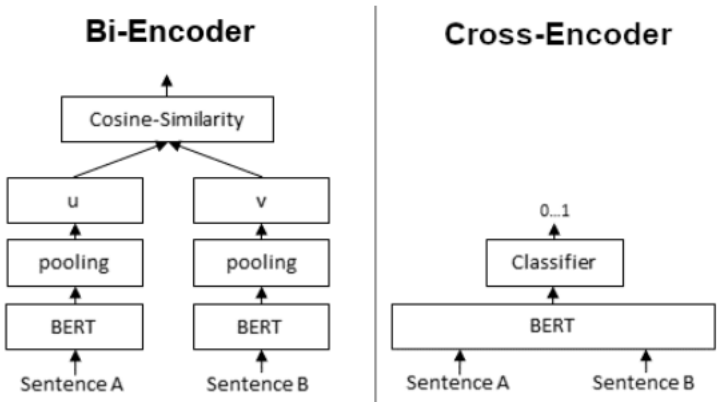
- grouppolicydata_collection (embeddings are already present)
- cache_collection (an empty collection at the beginning)

Our strategy is to first check the cache_collection and see if can answer the query or not.

In case it doesn't the query then passes to the grouppolicydata_collection, and the query results get appended to cache_collection (embeddings, metadata, etc)

Re-Ranking with a Cross Encoder:

Re-ranking the results obtained from your semantic search can sometime significantly improve the relevance of the retrieved results. This is done by passing the query paired with each of the retrieved responses into a cross-encoder to score the relevance of the response w.r.t. the query.



- Get the top 3 results from semantic search.
- Get the top 3 results after reranking.

Build the RAG system:

Built a robust prompt template that can give the search results. Future enhancements can include with the citations such as Page No, metadata, etc.,

Used OpenAI model “gpt-3.5-turbo” for chat completions and retrieved the response.