Documentation of HelpMateAI

Contributor: Ayushi Pitchika, ACP Gen AI C10, UpGrad

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.

The primary objectives of the project are as follows:

- Develop a semantic search system pipeline using the RAG (Embedding Layer, Search and Rank Layer, Generation Layer) pipeline for efficient document retrieval.
- Extract relevant information from PDF documents, store them in a structured format, and generate vector representations using SentenceTransformerEmbedding's all-MiniLM-L6-v2 model.
- Implement a cache layer to enhance system performance by storing and retrieving previous queries and their results.

System Design

Dataset: A single long life insurance policy document in .pdf format named "**Group Member Life Insurance Policy**".

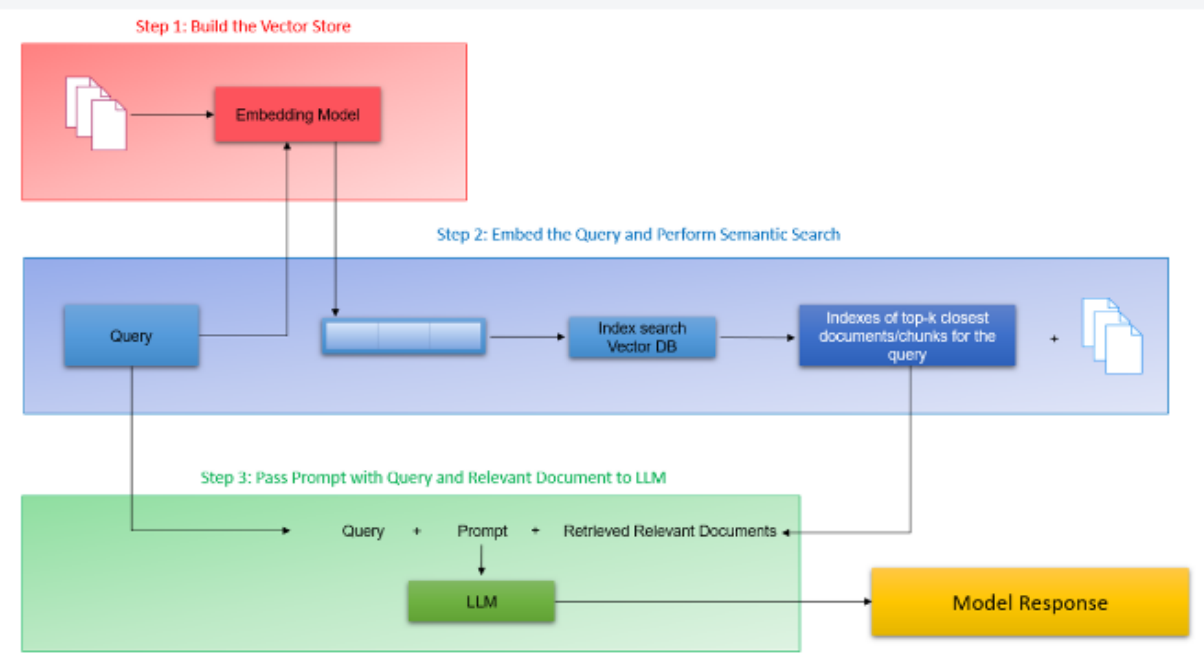
The RAG pipeline will consist of the following three layers:

1. **Embedding Layer**- The first step is to build the vector store. This step involves ingesting the documents, processing them to create individual chunks and passing these to an embedding model to create individual vector representations of the text.
   Focus:
   - chunking strategy
   - choice of embedding model

2. **Search and Rank Layer**- The second layer in the pipeline is the search and rank layer, which will perform a semantic similarity search on the knowledge bank based on the query and retrieve the top results. The output of this layer is the top K closest documents or chunks for the query and their indices.
   Focus:
   - design 3 test queries
   - embed and search vector DB against each query. Implement cache mechanism
   - implement reranking block with choice of cross-encoding model

3. **Generation Layer**- The last layer is the generation layer, which receives the results of the previous layer, which contains the top retrieved search results, the original user query and a well-constructed prompt to the LLM. These inputs allow the LLM to generate a more coherent answer that is relevant to the user query with information/relevant chunks stored in the knowledge base.
   Focus:
   - final prompt to be exhaustive, accurate and includes few-shot examples

Implementation



Retrieval Augmented Generation - System Design

Used Google Colab for development and leveraged libraries such as pdfplumber, tiktoken, openai, chromaDB, and sentence-transformers for document processing, embedding, and caching.

- Implemented functions to extract text and tables from PDFs, create a dataframe, generate vector embeddings, and perform semantic searches using the RAG pipeline.
- Developed a cache system using ChromaDB to store and retrieve previous queries and their results.
    - Set a threshold of 0.2 for semantic similarity.
    - Store queries and results in a cache_collection in ChromaDB for easy embedding and searching.
    - Use ChromaDB's utility functions to add documents, ids, and metadata to the cache_collection.

Challenges

1. Performance Scaling: Address concerns about system performance with an increased number of documents or users by implementing vector databases and scaling up compute units.
2. Cache Storage: Optimize the cache collection to efficiently store and retrieve queries and results.

Lessons Leaned

1. Efficient Document Processing: Processing PDFs efficiently is crucial; libraries like pdfplumber and suitable data structures for storage play a vital role.
2. Semantic Search Optimization: Fine-tune semantic search parameters and thresholds for optimal results.
3. Cache Management: Implement an effective cache management strategy to balance storage and retrieval efficiency.

Conclusion

The project successfully implements a semantic search system with the RAG pipeline and cache layer. The objectives are met, and the challenges are overcome with lessons learned for future improvements. The system provides a scalable and efficient solution for document retrieval and information extraction.