Mr.HelpMate AI

Background

The "HelpMateAI" project demonstrates the implementation of a Retrieval-Augmented Generation (RAG) approach to efficiently extract and generate responses from a long insurance policy document. The system leverages embeddings, semantic search, re-ranking, and generative AI techniques to provide accurate and relevant answers to user queries.

Problem Statement

The objective of this project is to develop a robust search system capable of effectively retrieving and generating answers from a lengthy life insurance policy document. The system should:

- Process and chunk the document efficiently.
- Use high-quality embeddings for improved retrieval.
- Implement a semantic search mechanism with caching.
- Enhance search results using a re-ranking approach.
- Generate well-structured responses using a generative model with proper context and citations.

Data Sources

The primary data source for this project is a single long-form life insurance policy document. This document is preprocessed and stored in a vector database for efficient retrieval.

System Design and Approach

1. Embedding Layer

Reading & Processing the PDF

- **Tool Used**: `pdfplumber` is employed to extract text from the insurance policy document.
- **Capabilities**: `pdfplumber` efficiently extracts structured text, tables, and other relevant information for improved preprocessing.

Document Chunking

- The document is large and spans multiple pages, requiring segmentation before generating embeddings.
- Chunking Strategy:
    - Multiple chunking strategies were explored, including fixed text, sentence-based, paragraph-based, and semantic chunking.
    - Among these, semantic chunking yielded the highest re-ranked scores, demonstrating improved retrieval effectiveness.

Generating Embeddings

- **Embedding Model**: The `text-embedding-ada-002` model from OpenAI is used to create dense vector embeddings.
- These embeddings enable high-quality semantic search.

Storing Embeddings in ChromaDB

- The embeddings are stored in `ChromaDB`, an efficient vector database, for quick retrieval and search.

2. Search Layer

Querying and Semantic Search

- Three queries are designed to test the effectiveness of the retrieval system.
- User queries are embedded and compared against stored embeddings in ChromaDB.

Cache Mechanism

- A caching layer is introduced to store frequent search queries and corresponding results, reducing retrieval latency.

Re-Ranking with a Cross Encoder

- A re-ranking step is added to improve the relevance of search results.
- The retrieved results are paired with the query and processed through a cross-encoder model from HuggingFace to refine relevance scores.

3. Generation Layer

Retrieval-Augmented Generation (RAG)

- Top-ranked search results are fed into GPT-3.5 alongside the user query.
- A well-structured prompt is engineered to improve response accuracy and coherence.
- The generated responses include citations to maintain transparency and reliability.

Challenges Faced & Solutions

Challenge	Solution Implemented
Handling a large document efficiently	Implemented optimized chunking techniques and tested various chunking strategies to improve retrieval accuracy.
Efficient storage and retrieval	Used ChromaDB for fast and scalable vector search, ensuring low-latency retrieval.
Improving search accuracy	Implemented a re-ranking model using a cross-encoder from HuggingFace to refine retrieved results.

Enhancing response generation	Designed an optimized prompt with structured context and explored few-shot examples for improving GPT-3.5 outputs.
Managing irrelevant or noisy retrievals	Fine-tuned the chunking strategy and re-ranking mechanism to filter out low-relevance results.
Optimizing caching for repeated queries	Introduced an efficient caching layer to store frequently accessed embeddings and reduce computational overhead.

Conclusion

This project successfully demonstrates a comprehensive pipeline for building an effective retrieval-based search system. By leveraging advanced embedding models, optimized chunking strategies, caching mechanisms, and retrieval-augmented generation techniques, the system ensures accurate and contextually relevant responses from a long insurance policy document.