Project :: HelpMateAI

Insurance Document Retrieval Project Report

Table of Contents

1. Project Overview

HelpMateAI is an innovative tool designed to simplify the process of retrieving information from large insurance documents. Traditionally, users faced difficulties locating specific details within documents that span 100+ pages. HelpMateAI leverages advanced AI models to enable precise and quick search capabilities, delivering results with citations for further reading. The solution focuses on saving time, enhancing accuracy, and improving the overall user experience when accessing critical information.

2. Problem Statement

Discuss the challenges faced by users:

- Long and complex documents (100+ pages).
- Manual effort required to find specific information.
- Time wasted searching multiple documents.
- Highlight the need for an AI-powered solution.

3. Objectives

The primary objectives of the project are as follows:

- Develop a semantic search system 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.

4. Design and Architecture

HelpMateAI project is designed in three stages:

- Step 1: Build the vector store using an embedding model.
- Step 2: Query caching, search, and re-ranking.
- Step 3: Generative search using LLMs.

5.Implementation Details

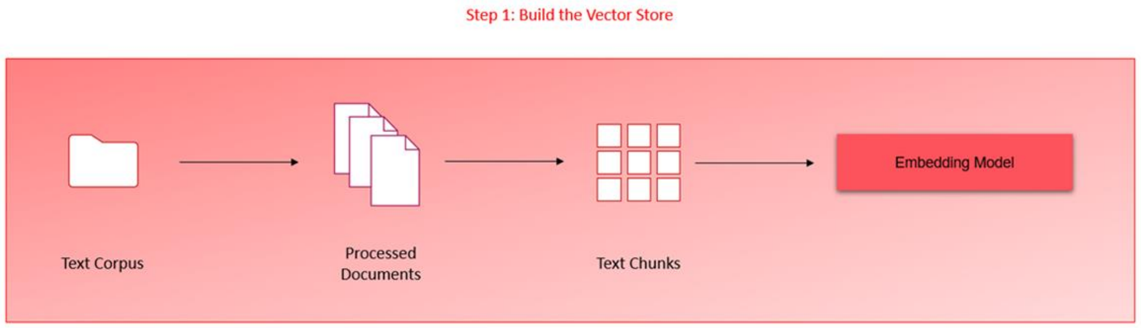
Use Google Colab for development and leverage libraries such as pdfplumber, tiktoken, openai, chromaDB, and sentence-transformers for document processing, embedding, and caching. Implement functions to extract text and tables from PDFs, create a dataframe, generate vector embeddings, and perform semantic searches using the RAG pipeline. Develop a cache system using ChromaDB to store and retrieve previous queries and their results.

The three layers for RAG pipeline are:

1. Embedding Layer
2. Search Layer
3. Generation Layer

Embedding Layer:

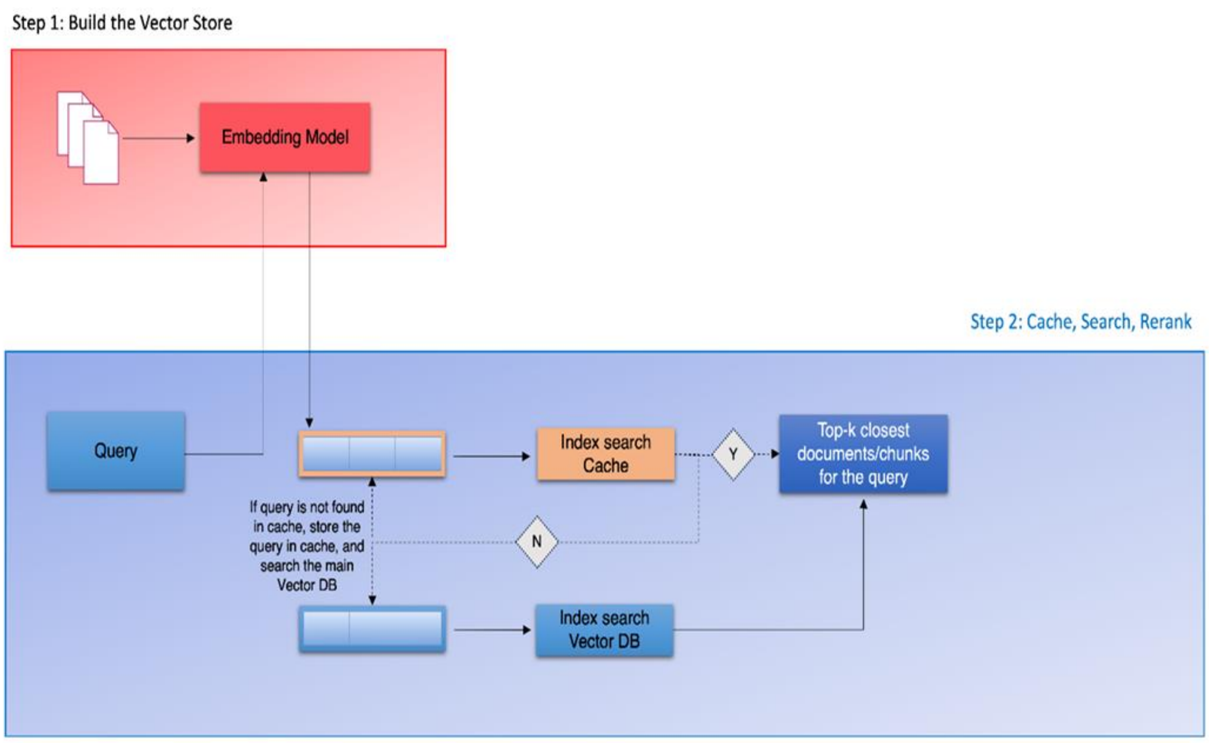
Processing and Chunking: Explore and compare various strategies for effective PDF document processing, cleaning, and chunking. Evaluate the impact of different chunking strategies on the quality of the retrieved results. Embedding Model Choice: Choose between OpenAI's embedding model and SentenceTransformers from HuggingFace. Assess the impact of the selected model on the quality of vector representations.



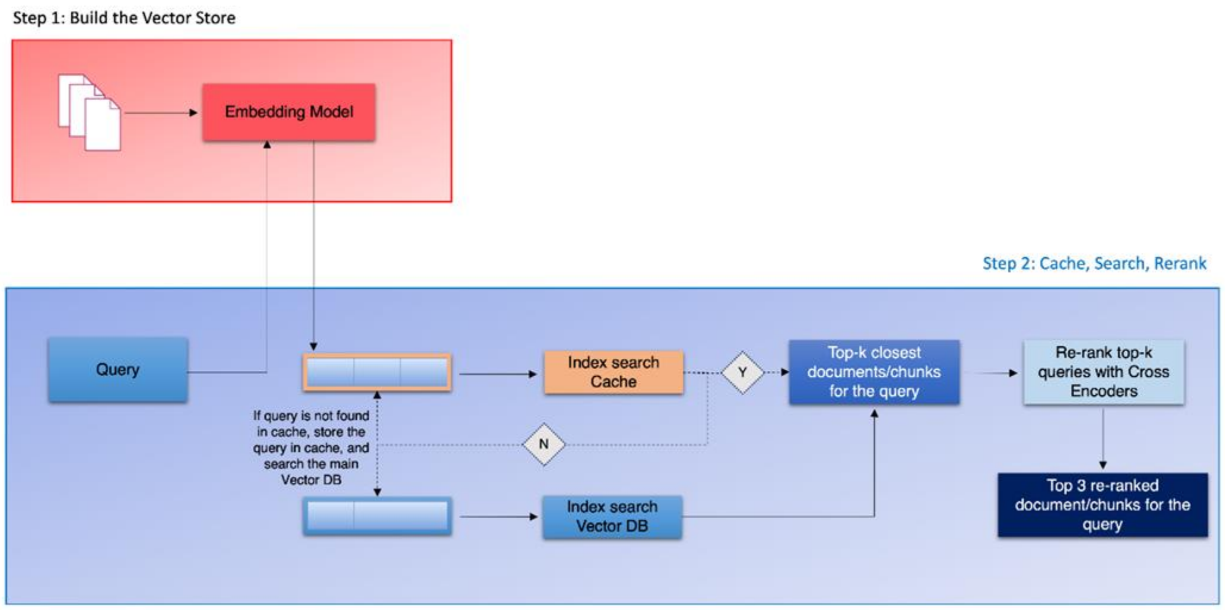
Search Layer:

Query Design: Design a minimum of three queries that reflect information seekers' potential questions in the policy document. Ensure queries cover diverse aspects of the document to thoroughly test the system.

Vector Database Search: Embed queries and perform searches against the ChromaDB vector database. Implement a cache mechanism to store and retrieve previous queries and their results. Re-ranking Block: Integrate a re-ranking block utilizing cross-encoding models from HuggingFace to enhance the relevance and accuracy of search results.


Step 1: Build the Vector Store
Embedding Model
Step 2: Cache, Search, Rerank
Query
Index search Cache
Y
Top-k closest documents/chunks for the query
If query is not found in cache, store the query in cache, and search the main Vector DB
N
Index search Vector DB

When using cross_encoder:


Step 1: Build the Vector Store
Embedding Model
Step 2: Cache, Search, Rerank
Query
Index search Cache
Y
Top-k closest documents/chunks for the query
Re-rank top-k queries with Cross Encoders
If query is not found in cache, store the query in cache, and search the main Vector DB
N
Index search Vector DB
Top 3 re-ranked document/chunks for the query

Generation Layer:

Prompt Design: Focus on designing a comprehensive and instructive prompt for the Language Model (LM) in the generation layer. Ensure the prompt effectively conveys relevant information to the LM for coherent answer generation. Few-shot Examples: Enhance LM performance by providing few-shot examples in the prompt to guide the model in generating more contextually accurate responses.

6. Challenges:

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

7. Lessons Learned:

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

8. 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.