HELP MATE AI: Insurance Document Question Answering Assistant

**Project Overview**

**HELP MATE AI** is a PDF-based question-answering system designed to semantically search and extract contextual answers from insurance policy documents using large language models (LLMs), semantic embeddings, vector databases, and reranking techniques.

This system is optimized for life insurance documents and enables:

* **PDF parsing & preprocessing**
* **Document indexing using OpenAI embeddings**
* **ChromaDB persistent storage**
* **Query caching**
* **Semantic search**
* **Reranking using CrossEncoder**
* **LLM-based contextual response generation**

**📁 Table of Contents**

1. Environment Setup
2. Architecture
3. PDF Parsing & Preprocessing
4. Vector Store Initialization (ChromaDB)
5. Semantic Search Pipeline
6. Reranking with CrossEncoder
7. LLM-based Response Generation
8. Caching System
9. Modular Query Flow
10. Example Queries & Expected Output
11. Conclusion

**Environment Setup**

**Libraries Installed**

pip install -U -q pdfplumber tiktoken openai chromadb sentence-transformers

**Libraries Used**

import pdfplumber, json, os, openai, tiktoken

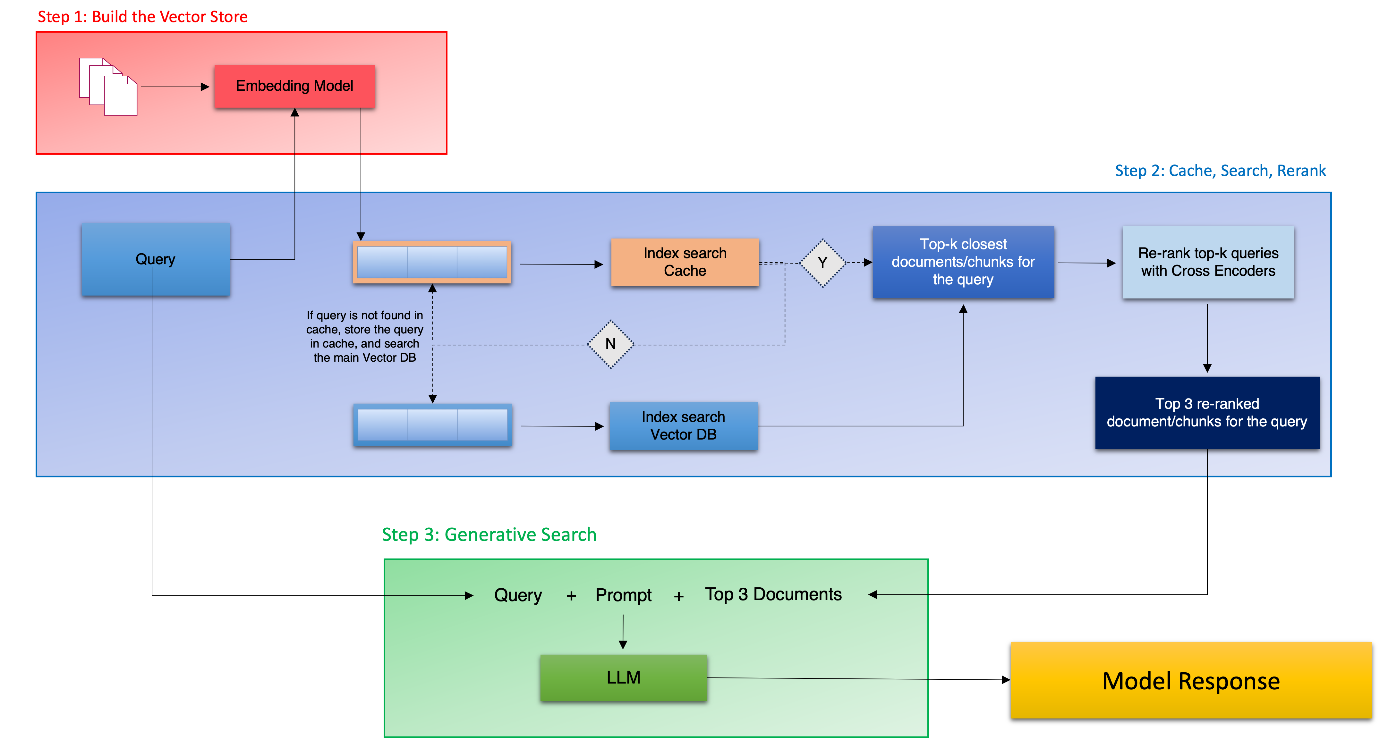
from pathlib import Path

import pandas as pd

import chromadb

from sentence\_transformers import CrossEncoder

**Architecture**



**PDF Parsing & Preprocessing**

**Goal:**

Extract structured and unstructured text (including tables) from insurance policy PDF files.

**Key Steps:**

1. **Mount Google Drive** to access PDF.
2. **Extract content** from each page:
   * Text headers
   * Non-table words
   * Tables (retaining structure)
3. **Filter empty pages**
4. **Prepare structured DataFrame** with metadata for each page.

**Core Function: extract\_text\_from\_pdf()**

**Output:**

* DataFrame: [Page No., Heading, Page\_Text, Metadata]

**Metadata:**

{

'Section': <first line of text>,

'Page\_No.': <page number>

}

**Vector Store Initialization (ChromaDB)**

**Purpose:**

Store the document embeddings in a persistent vector database for fast retrieval.

**Components:**

* **ChromaDB PersistentClient**
* **OpenAI Embedding (text-embedding-ada-002)**
* **Two collections**:
  + InsurancePolicyDoc: Stores all extracted page embeddings.
  + Insurance\_Cache: Caches results of previous queries.

**Setup:**

embedding\_function = OpenAIEmbeddingFunction(api\_key=openai\_api\_key, model\_name="text-embedding-ada-002")

client = chromadb.PersistentClient()

insurance\_collection = client.get\_or\_create\_collection(name='InsurancePolicyDoc', embedding\_function=embedding\_function)

**Semantic Search Pipeline**

**Input:**

* query (user input)

**Logic:**

1. Check if the query exists in the **cache**.
2. If not found or threshold exceeded:
   * Retrieve top 10 relevant pages from InsurancePolicyDoc.
   * Store the results in Insurance\_Cache.
3. Else, return results from the cache.

**Threshold:**

threshold = 0.2

**Output:**

* Dictionary of documents, embeddings, metadata, and distances

**Reranking with CrossEncoder**

**Purpose:**

To improve result precision beyond semantic similarity, a **cross-encoder** scores query-result pairs for relevance.

**Model:**

CrossEncoder('cross-encoder/ms-marco-MiniLM-L-6-v2')

**Output:**

* Reranked results stored in new column: Reranked\_scores
* Top 3 responses selected for generation

**LLM-based Response Generation**

**Model:**

gpt-4o-mini (via OpenAI Chat API)

**Function: generate\_response(query, top\_3\_results\_df)**

**Responsibilities:**

* Ingest query + top 3 document snippets
* Format responses with:
  + Rephrased answers
  + Reformatted tables if relevant
  + Source citations (from metadata)

**Output:**

Human-readable answer with source references.

**🧠 Caching System**

**Collection:**

* Insurance\_Cache

**Logic:**

* Before performing a vector search, check if query exists in cache.
* Save newly retrieved results back into the cache.

**Stored Metadata:**

Flattened key-value structure using JSON.

**🧩 Modular Query Flow**

Modularized into 4 key functions:

| **Function** | **Purpose** |
| --- | --- |
| get\_context() | Check cache → retrieve from main collection if needed |
| rerank() | Relevance scoring using cross-encoder |
| top\_3\_context() | Extract top 3 results |
| get\_reply() | Full end-to-end response generator |

**Final Input Flow:**

query = input()

print(get\_reply(query))

**Example Queries & Expected Output**

**Query:**

What is the coverage for accidental death?

**Sample Output:**

The policy provides coverage for accidental death, where the benefit amount is typically 200% of the base coverage. The death must occur as a direct result of an accident and within a specified period, usually 90–180 days depending on the policy.

**Citations:**

* Section: “Coverage Benefits”, Page 3
* Section: “Policy Benefits Table”, Page 4

**✅ Advantages**

* ✅ Efficient PDF parsing with table support
* ✅ Persistent vector DB using Chroma
* ✅ Fast & accurate search using embeddings + reranking
* ✅ Query caching to reduce repeated API cost
* ✅ Human-like LLM-generated answers with citation context

**Dependencies**

| **Library** | **Purpose** |
| --- | --- |
| pdfplumber | PDF parsing |
| tiktoken | Token counting |
| openai | Embeddings + GPT chat |
| chromadb | Persistent vector storage |
| sentence-transformers | Cross-encoder reranking |
| pandas | Data structuring |
| google.colab.userdata | Secure API key storage |
|  |  |

**File & Folder Structure**

/MyDrive/  
└── Colab Notebooks/  
 ├── insuranceDb/  
 ├── Principal-Sample-Life-Insurance-Policy.pdf  
 └── helpmate\_ai\_notebook.ipynb

**System Requirements**

- Python 3.8+  
- Google Colab or Jupyter  
- OpenAI API key  
- Internet connection

**Final Notes**

Demonstrates an effective retrieval + generation pipeline for understanding insurance documents.