HelpMateAI- Building an Effective Search System

1. Project Statement

The objective of this project is to build an **effective generative search system** for answering questions from long and complex insurance policy documents.

Using Retrieval-Augmented Generation (RAG) techniques, the system:

- Processes and chunks large PDF documents into manageable units.
- Embeds these chunks into vector representations for semantic understanding.
- Retrieves and re-ranks the most relevant sections for any user query.
- Generates precise, context-aware answers with proper citations from the source policy.

This project demonstrates how to design a robust, multi-layer RAG pipeline that combines:

1. Embedding Layer – converting text into embeddings for semantic search.
2. Retrieval Layer– fetching and re-ranking the most relevant context.
3. Generation Layer – using LLMs to produce grounded, user-friendly responses.

The outcome is a search system that is **accurate, explainable, and scalable**, capable of supporting real-world use cases like **insurance policy exploration and question answering**.

2. Summary of Work Completed

a. PDF Ingestion & Preprocessing

- - Extracted text and tables from the insurance policy using **pdfplumber**.
- - Built a structured **DataFrame** with:
- - `Page_No.` → page identifier
- - `Page_Text` → extracted content
- - Applied **document chunking** (~500 words per chunk) for embeddings.
- - Added metadata (page and chunk IDs) to support retrieval and citation.

b. Embedding Layer (Indexing)

- - Generated dense vector embeddings using **SentenceTransformer (all-MiniLM-L6-v2)**.
- - Stored embeddings + documents + metadata in a **persistent ChromaDB collection** (`insurance-collection`).
- - Established a scalable indexing mechanism for efficient semantic search.

c. Retrieval Layer

- - Implemented **semantic search** over ChromaDB using similarity scores.
- - Added a **semantic cache** (`insurance-collection-cache`) to reduce repeated searches:
- - If a close match is found → returns from cache.
- - If not → queries main index, then writes result back to cache.
- - Incorporated **re-ranking** using a **CrossEncoder** model (`cross-encoder/ms-marco-MiniLM-L-6-v2`):
- - Scores query–chunk pairs.
- - Selects the **top-3 most relevant chunks** for answer generation.
-

d. Generation Layer (RAG)

- - Designed a **structured prompt** that:
- - Injects the top-3 retrieved chunks (`Documents`) + their `Metadatas`.
- - Instructs the model to answer **strictly from sources**.
- - Extracts values from **tables** when relevant.
- - Requires **citations** (policy name and page) in every response.
- - Used **OpenAI GPT (gpt-3.5-turbo)** via a `generate_response(query, top_3)` function.
- - Ensured responses are **accurate, source-grounded, and citation-rich**.

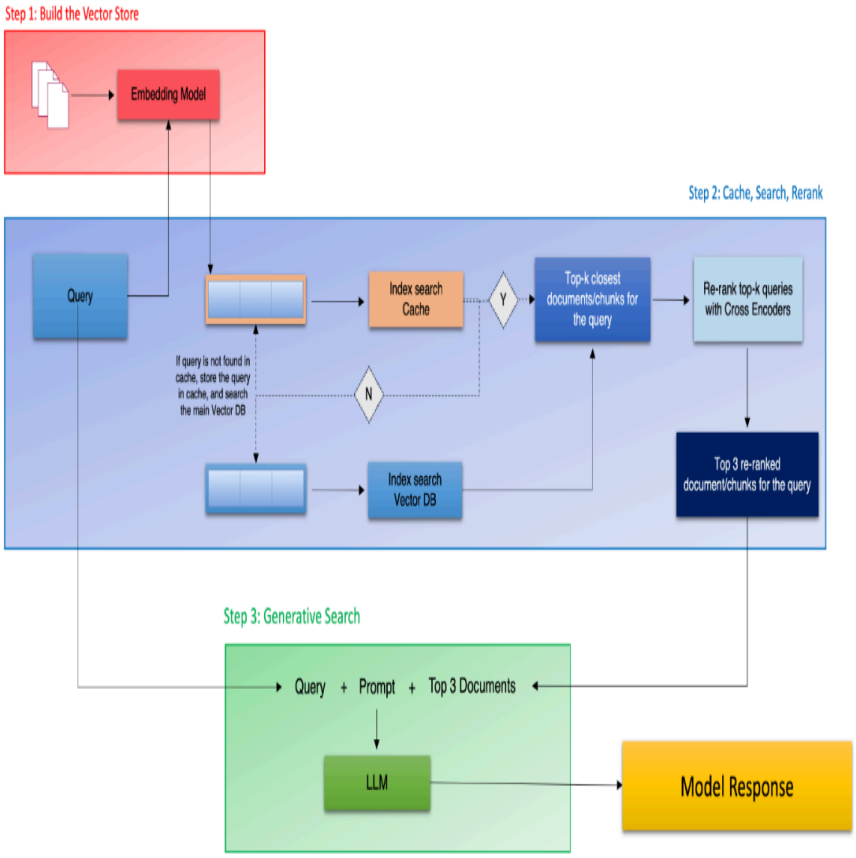
e. Demonstration & Queries

- 1. User query →
- 2. Semantic retrieval →
- 3. Cache check & rerank →
- 4. Top-3 context selection →
- 5. Final answer generation with citations.
- - Example queries confirm the system's ability to retrieve, rerank, and generate coherent answers.

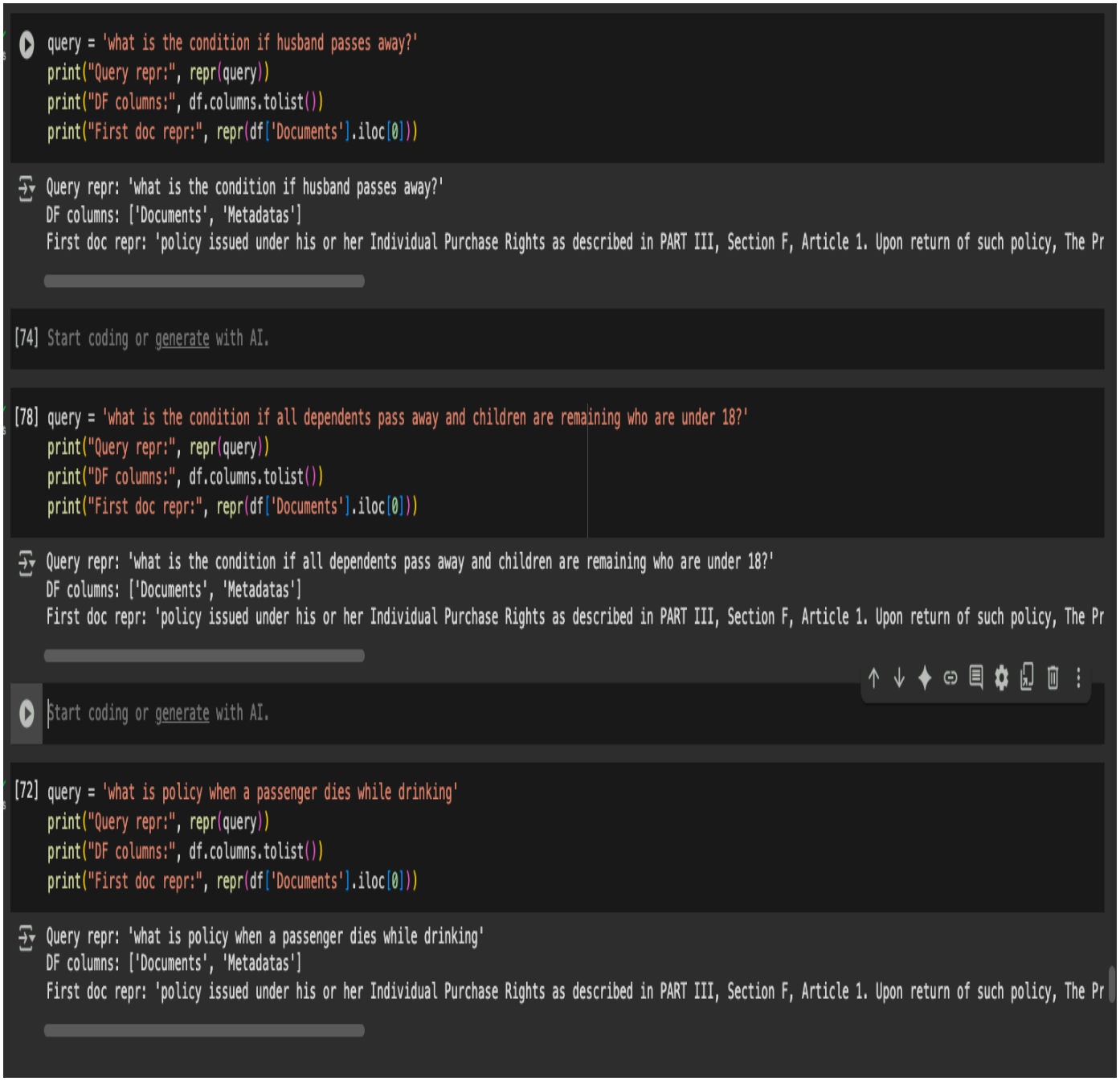
f. Additional Highlights

- - Persistent storage: avoids re-embedding documents across sessions.
- - 3-layer RAG design: clear modular pipeline (Embedding → Retrieval → Generation).
- - Extensibility: supports multiple policy PDFs in the same vector DB.
- - Reusability: modular helper functions for ingestion, embedding, retrieval, and generation.

g. System Architecture:



3. Sample Queries and Answers:



```
query = 'what is the condition if husband passes away?'
print("Query repr:", repr(query))
print("DF columns:", df.columns.tolist())
print("First doc repr:", repr(df['Documents'].iloc[0]))
```

```
Query repr: 'what is the condition if husband passes away?'
DF columns: ['Documents', 'Metadatas']
First doc repr: 'policy issued under his or her Individual Purchase Rights as described in PART III, Section F, Article 1. Upon return of such policy, The Pr
```

```
[74] Start coding or generate with AI.
```

```
[78] query = 'what is the condition if all dependents pass away and children are remaining who are under 18?'
print("Query repr:", repr(query))
print("DF columns:", df.columns.tolist())
print("First doc repr:", repr(df['Documents'].iloc[0]))
```

```
Query repr: 'what is the condition if all dependents pass away and children are remaining who are under 18?'
DF columns: ['Documents', 'Metadatas']
First doc repr: 'policy issued under his or her Individual Purchase Rights as described in PART III, Section F, Article 1. Upon return of such policy, The Pr
```

```
Start coding or generate with AI.
```

```
[72] query = 'what is policy when a passenger dies while drinking'
print("Query repr:", repr(query))
print("DF columns:", df.columns.tolist())
print("First doc repr:", repr(df['Documents'].iloc[0]))
```

```
Query repr: 'what is policy when a passenger dies while drinking'
DF columns: ['Documents', 'Metadatas']
First doc repr: 'policy issued under his or her Individual Purchase Rights as described in PART III, Section F, Article 1. Upon return of such policy, The Pr
```