Building Effective Search Systems with Helpmate AI

1. Background

This project focuses on developing an advanced and efficient search system, Helpmate AI, designed to process and extract information from a comprehensive life insurance policy document using Retrieval-Augmented Generation (RAG) techniques.

2. Problem Statement

The primary objective of this project is to build a robust generative search system capable of providing accurate and contextually relevant answers to user queries based on a single, long life insurance policy document.

3. Document

The project uses a single, long-form life insurance policy document as the primary data source.

The policy document can be found here.

4. Approach

The architecture of the search system is built around three core layers, each playing a vital role in the system's efficiency and accuracy. Various strategies and experiments are implemented to optimize performance at each layer.

4.1 Embedding Layer

- **Document Processing and Chunking:** The PDF document is processed, cleaned, and divided into meaningful chunks to prepare it for embedding. The choice of chunking strategy significantly impacts the quality of the search results.
- **Embedding Model Selection:** Embeddings are generated using models like OpenAI's embedding models or SentenceTransformers from the HuggingFace library. Experiments are conducted to evaluate and select the most effective model.

4.2 Search Layer

- **Query Design:** At least three test queries are designed based on the document's content to evaluate the system's performance.
- **Semantic Search:** Queries are embedded and matched against the document's ChromaDB vector database. A caching mechanism is implemented to improve search efficiency.
- **Re-Ranking:** Retrieved results are re-ranked using cross-encoding models from the HuggingFace library to enhance relevance and accuracy.

4.3 Generation Layer

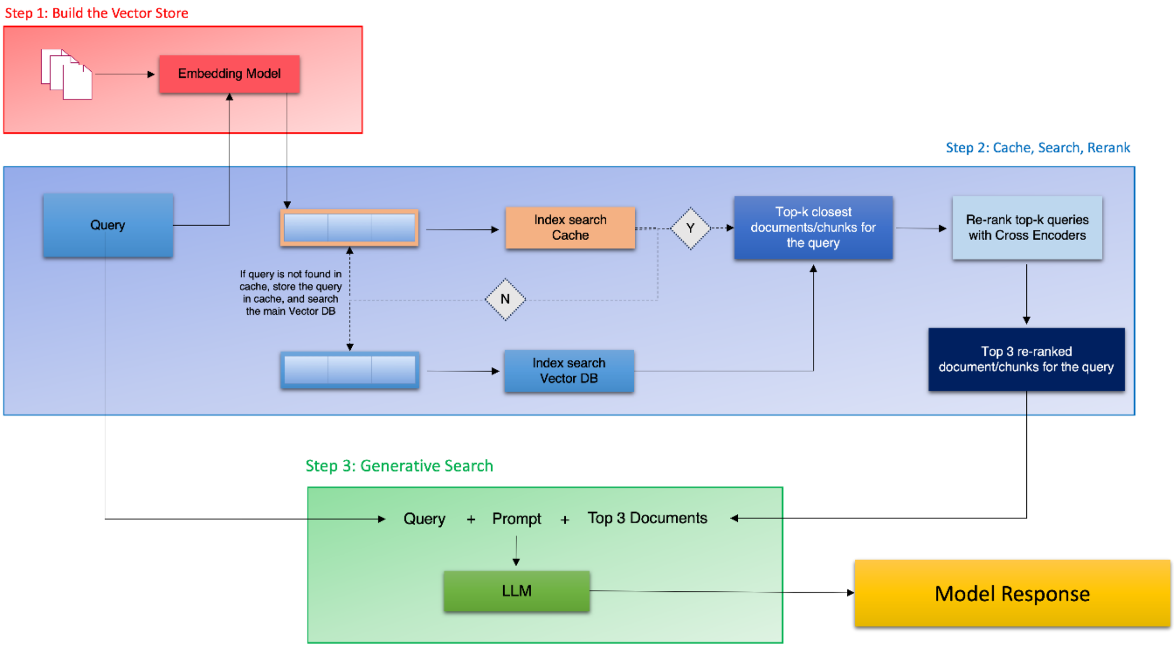
- **Prompt Engineering:** Carefully designed prompts ensure that the information passed to the language model is clear and exhaustive.
- **Few-Shot Examples:** Providing illustrative examples enhances the language model's output quality and accuracy.

5. System Layers

- **PDF Reading and Processing:** Using the pdfplumber library, the system efficiently reads and extracts text, tables, and images from the PDF document.
- **Document Chunking:** The document is divided into fixed-size text chunks to facilitate effective embedding and retrieval.
- **Embedding Generation:** SentenceTransformer's all-MiniLM-L6-v2 model is used to create embeddings.
- **Embedding Storage:** Embeddings are stored in the ChromaDB vector database.
- **Semantic Search with Caching:** A caching layer is introduced to optimize the retrieval of stored embeddings.
- **Re-Ranking:** Cross-encoders refine search results by scoring the relevance of retrieved responses.
- **Retrieval-Augmented Generation:** Final top search results, user queries, and a well-constructed prompt are fed into GPT-3.5 to generate direct, well-cited answers.

6. System Architecture

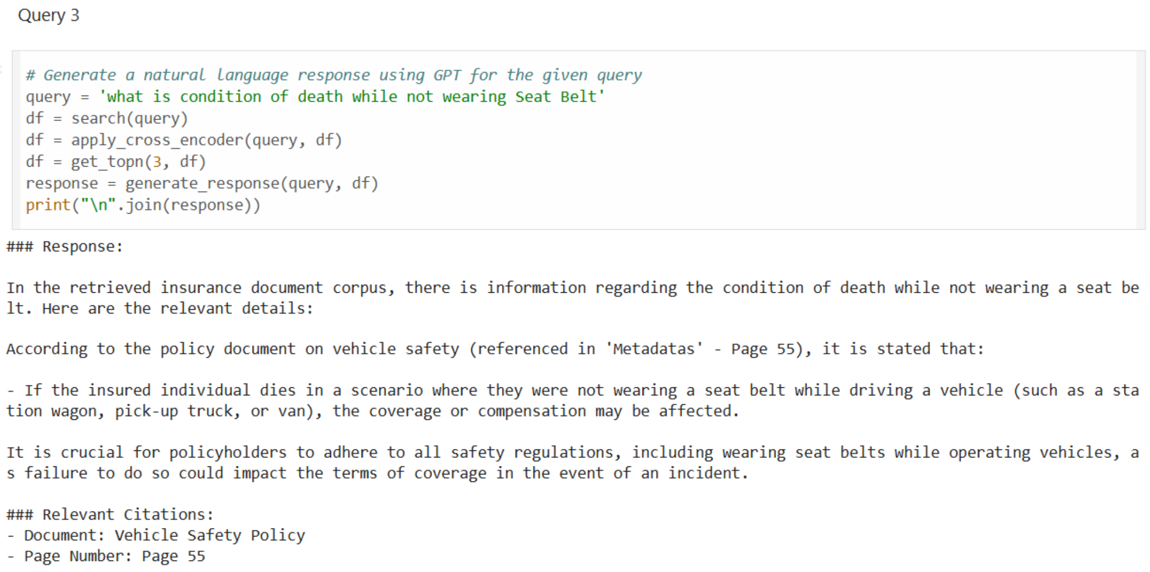
The architecture comprises three integrated layers—embedding, search, and generation—working together to deliver high-accuracy responses with efficient data retrieval and contextual understanding.


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
Step 3: Generative Search
Query + Prompt + Top 3 Documents
LLM
Model Response

7. Prerequisites

- Python 3.7+
- OpenAI API key stored in a text file named OpenAI_API_Key for API access

8. Query Response Output Example



Query 3

```
# Generate a natural language response using GPT for the given query
query = 'what is condition of death while not wearing Seat Belt'
df = search(query)
df = apply_cross_encoder(query, df)
df = get_topn(3, df)
response = generate_response(query, df)
print("\n".join(response))
```

```
### Response:

In the retrieved insurance document corpus, there is information regarding the condition of death while not wearing a seat be
lt. Here are the relevant details:

According to the policy document on vehicle safety (referenced in 'Metadatas' - Page 55), it is stated that:

- If the insured individual dies in a scenario where they were not wearing a seat belt while driving a vehicle (such as a sta
tion wagon, pick-up truck, or van), the coverage or compensation may be affected.

It is crucial for policyholders to adhere to all safety regulations, including wearing seat belts while operating vehicles, a
s failure to do so could impact the terms of coverage in the event of an incident.

### Relevant Citations:
- Document: Vehicle Safety Policy
- Page Number: Page 55
```

9. Challenges and Lessons Learned

- **Challenge:** Selecting the optimal chunking strategy to balance data granularity and embedding efficiency.
    - ✓ **Solution:** Conducting multiple experiments to identify the best-performing chunking method.
- **Challenge:** Ensuring high search relevance and minimizing response latency.
    - ✓ **Solution:** Implementing caching and re-ranking mechanisms to optimize search performance.
- **Lesson:** The quality of embeddings and prompt design directly impacts system output accuracy.

10. Future Improvements

- Explore more advanced embedding models and cross-encoders.
- Implement dynamic chunking based on document structure.
- Enhance the system’s adaptability to different document types.

11. Conclusion

The HelpMate AI search system demonstrates an effective approach to building a generative search platform, leveraging RAG techniques and advanced language models. By optimizing document processing, search efficiency, and response generation, this system delivers accurate, context-rich answers, setting a foundation for future enhancements and applications.