Building Effective Search Systems Help-MateAI

1. Background

This project demonstrate "Building Effective Search Systems HelpMateAI" with a long insurance policy document using RAG techniques.

2. Problem Statement

The goal of the project is to build a robust generative search system capable of effectively and accurately answering questions from a policy document.

We will be using a single long life insurance policy document for this project.

3. Document

1. The policy document can be found here

4. Approach

The project should implement all the three layers effectively. It will be key to try out various strategies and experiments in various layers in order to build an effective search system. Let's explore what we need to do in each of the layers.

1. **The Embedding Layer:** The PDF document needs to be effectively processed, cleaned, and chunked for the embeddings. Here, the choice of the chunking strategy will have a large impact on the final quality of the retrieved results. So, we need to make sure that we try out various stratgies and compare their performances.

Another important aspect in the embedding layer is the choice of the embedding model. we can choose to embed the chunks using the OpenAI embedding model or any model from the SentenceTransformers library on HuggingFace.

2. **The Search Layer:** Here, we need to design at least 3 queries against which you will test your system. You need to understand and skim through the document, and accordingly come up with some queries, the answers to which can be found in the policy document.

   Next, we need to embed the queries and search your ChromaDB vector database against each of these queries. Implementing a cache mechanism is also mandatory.

   Finally, we need to implement the re-ranking block, and for this we can choose from a range of cross-encoding models on HuggingFace.

3. **The Generation Layer:** In the generation layer, the final prompt that we design is the major component. We need to make sure that the prompt is exhaustive in its instructions, and the relevant information is correctly passed to the prompt. We may also choose to provide some few-shot examples in an attempt to improve the LLM output.

5. System Layers

- **Reading & Processing PDF File:** We will be using pdfplumber to read and process the PDF files. pdfplumber allows for better parsing of the PDF file as it can read various elements of the PDF apart from the plain text, such as, tables, images, etc. It also offers wide functionalities and visual debugging features to help with advanced preprocessing as well.

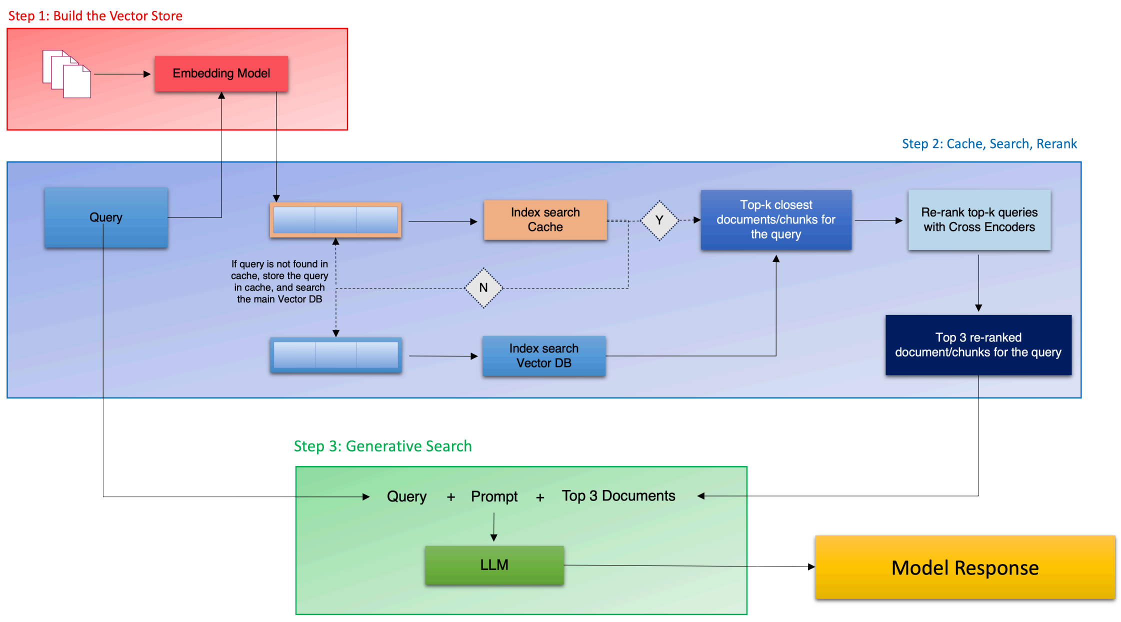
- **Document Chunking:** The document contains several pages and contains huge text, before generating the embeddings, we need to generate the chunks. Let’s start with a basic chunking technique, and chunking the text with fixed size.

- **Generating Embeddings:** Generates embedding with SentenceTransformer with all-MiniLM-L6-v2 model.

- **Store Embeddings In ChromaDB:** In this section we will store embedding in ChromaDB.

- **Semantic Search with Cache:** In this section we will introduce cache collection layer for embeddings.
- **Re-Ranking with a Cross Encoder:** Re-ranking the results obtained from the semantic search will sometime significantly improve the relevance of the retrieved results. This is often done by passing the query paired with each of the retrieved responses into a cross-encoder to score the relevance of the response w.r.t. the query.
- **Retrieval Augmented Generation:** Now we have the final top search results, we can pass it to an GPT 3.5 along with the user query and a well-engineered prompt, to generate a direct answer to the query along with citations.

6. System Architecture


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+
- Please ensure that you add your OpenAI API key to the empty text file named “OpenAI_API_Key” in order to access the OpenAI API.

8. Query Screenshots

```python
query = 'what is the life insurance coverage for disability'
df = search(query)
df = apply_cross_encoder(query, df)
df = get_topn(3, df)
response = generate_response(query, df)
print("\n".join(response))
```

```
The life insurance coverage for disability typically depends on the specific policy terms and conditions outlined
in the insurance document. To provide accurate information on coverage details, it is essential to review the poli
cy document. Please refer to the relevant sections within the policy document for specific details on life insuran
ce coverage for disability.

Citation:
- Policy Name: Member Life Insurance or Coverage During Disability
- Page Number: Page 42

Please review the policy document for detailed information on life insurance coverage for disability.
```

```python
query = 'what is the Proof of ADL Disability or Total Disability'
df = search(query)
df = apply_cross_encoder(query, df)
df = get_topn(3, df)
response = generate_response(query, df)
print("\n".join(response))
```

```
The Proof of ADL Disability or Total Disability typically refers to the documentation required to prove the insure
d individual's inability to perform Activities of Daily Living (ADLs) or work due to a disability.

Based on the available information in the insurance documents, the policy document titled "Member Life Insurance o
r Coverage During Disability" on Page 42 discusses the termination of Dependent's Life Insurance. This document ma
y not specifically outline the Proof of ADL Disability or Total Disability.

Therefore, the query about the Proof of ADL Disability or Total Disability is not directly addressed in the provid
ed insurance documents.

Citations:
Policy Name: Member Life Insurance or Coverage During Disability
Page Number: Page 42
```

```python
query = 'what is condition of deatht 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))
```

```
The specific query about the condition of death while not wearing a seatbelt is not directly addressed in the prov
ided insurance documents. However, you can refer to the following related sections in the documents for more infor
mation on conditions surrounding death, benefits, and coverage:

1. Payment of benefits subject to the Benefit Precedence provision
2. Member Life Insurance or Coverage During Disability
3. Termination of Dependent's Life Insurance

For detailed information regarding death conditions without wearing a seatbelt, it is recommended to review the en
tire document or search for specific keywords related to seatbelt non-usage in the respective policy documents.

Citations:
1. Policy Name: Payment of benefits
   Source Page: Page 49

2. Policy Name: Member Life Insurance or Coverage During Disability
   Source Page: Page 42

3. Policy Name: Dependent's Life Insurance termination
   Source Page: Page 44
```