Building Effective Search Systems

Prepared by: Maimana Kowatly

Date:29/01/2025

Contents

Background

This project focuses on building a generative search system capable of effectively answering user queries. To achieve this, we will experiment with various components, including chunking strategies, embedding models, re-rankers, and prompt engineering. The system will be structured to ensure accurate and efficient retrieval of relevant information.

Problem Statement

The goal of this project is to develop a robust search system that can accurately respond to questions based on an insurance policy document. We will work with a single long life insurance policy document (document can be found here), ensuring it is processed and structured effectively for optimal search performance.

Approach

The project implements all the three layers effectively. It will be key to try out various strategies and experiments in various layers to build an effective search system.

- **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, this project we'll be trying various strategies and comparing their performances.

- **The Search Layer**: design at least 3 queries against which will test the system. to understand and skim through the document is required, and accordingly come up with some queries, the answers to which can be found in the policy document.

- **The Generation Layer**: In the generation layer, the final prompt that is designed is the major component. Making sure that the prompt is exhaustive in its instructions, and the relevant information is correctly passed to the prompt. Some few-shot examples is provided in an attempt to improve the LLM output.

System Layers

1. **Reading & Processing PDF Files:**

   We begin by using **pdfplumber** to read and process PDF files, which offers advanced functionality for parsing beyond just plain text. **pdfplumber** can extract various elements such as tables, images, and text with a high degree of accuracy, making it ideal for extracting structured and unstructured content from complex PDF documents. It also includes debugging features that assist in visualizing and refining the document extraction process.

2. **Document Chunking:**

   Given the large volume of text in PDF files, the next step is to divide the text into smaller, more manageable chunks. This process, known as document chunking, breaks the text into fixed-size segments, which makes it easier to process and retrieve information later. By chunking the document, we prepare the data for generating embeddings while maintaining context and preserving the document's overall structure.

3. **Generating Embeddings**

   After chunking the text, we use **SentenceTransformer** with the all-MiniLM-L6-v2 model to generate embeddings for each chunk. These embeddings capture the semantic meaning of the text, converting it into a numerical representation that can be efficiently compared and retrieved. This step is critical for enabling meaningful searches and answering user queries based on the content's underlying meaning.

4. **Store Embeddings In ChromaDB**

   The embeddings are then stored in **ChromaDB**, a specialized database designed for the efficient storage and retrieval of embeddings. ChromaDB optimizes search and retrieval performance, making it easier to access relevant information quickly during subsequent search operations.

5. **Semantic Search with Cache**

   To further enhance performance, we introduce a **cache collection layer** for embeddings. This ensures faster retrieval of previously accessed embeddings, speeding up the semantic search process by reducing redundant computation.

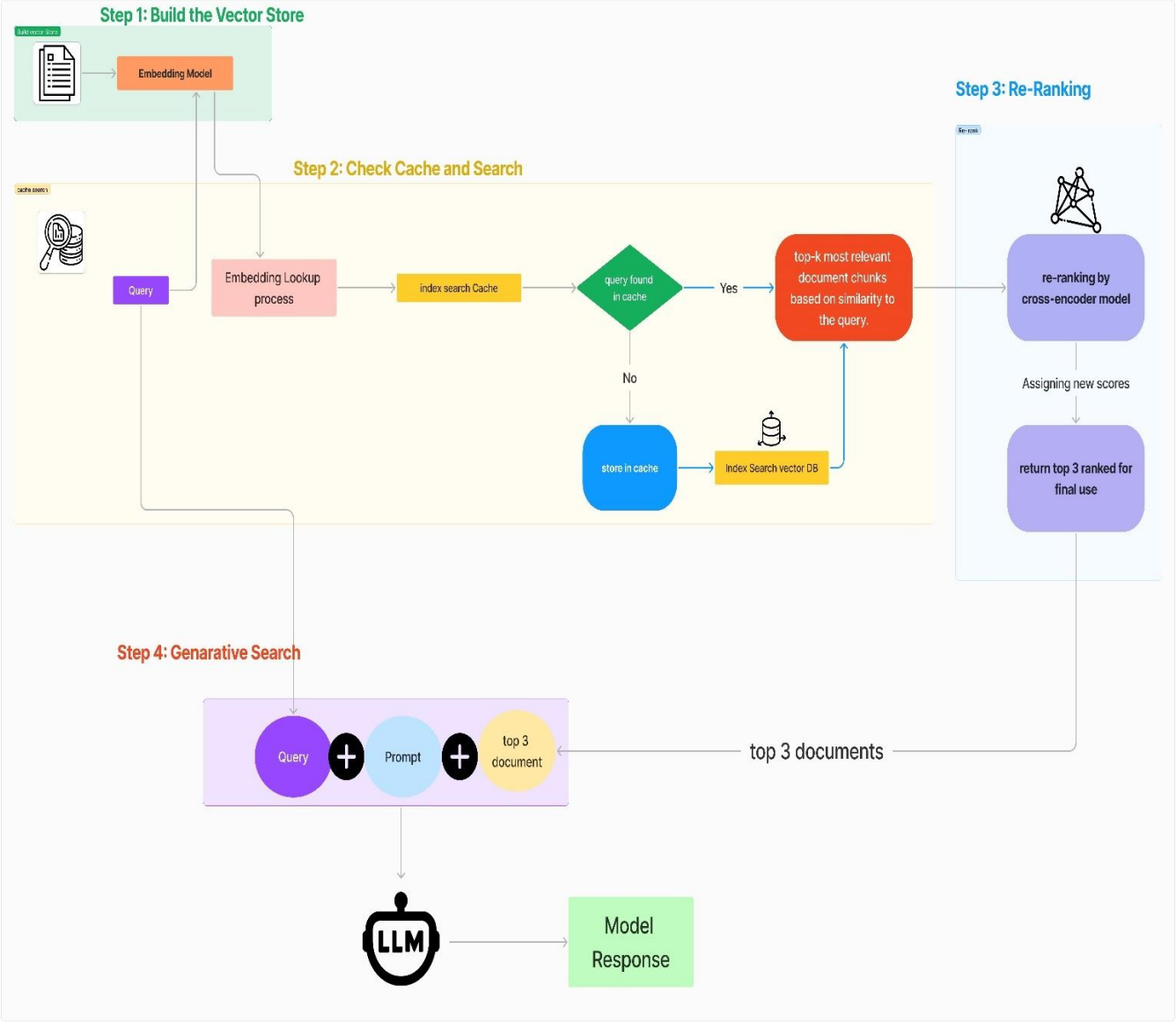
6. **Re-Ranking with a Cross Encoder**

   Once search results are obtained, we apply **re-ranking with a cross encoder** to refine the relevance of the results. The query and response are passed together into a cross-encoder, which scores their relevance and improves the accuracy of the results.

7. **Retrieval Augmented Generation** (RAG)

   Used to generate direct answers to user queries. The top search results are passed to **GPT-3.5** along with a well-crafted prompt. GPT-3.5 processes the information and generates a response with citations, providing the user with both an accurate answer and relevant references.

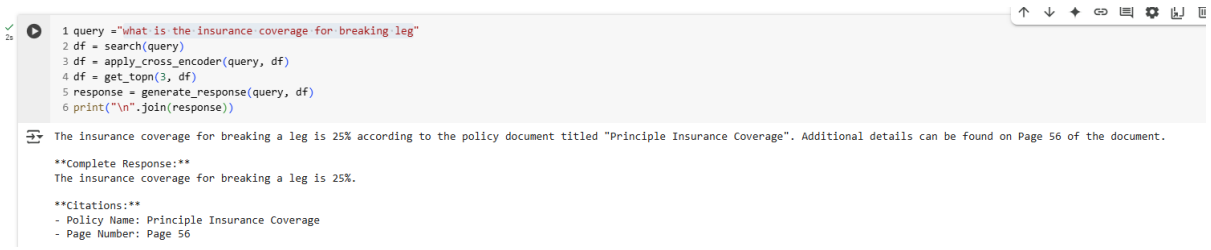
System Architecture



Prerequisites

OpenAI API key

Query Screenshots

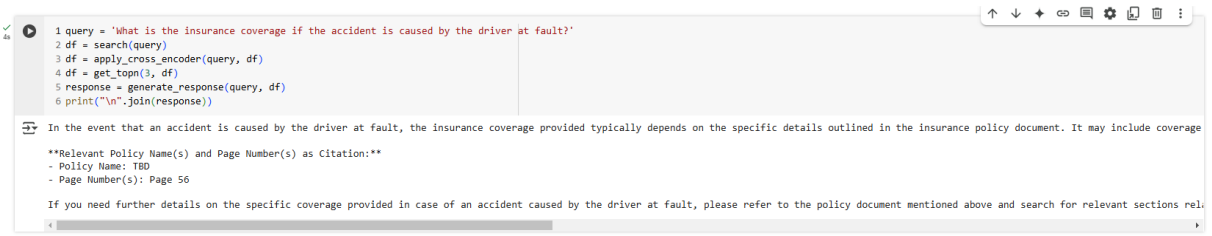


```
query ="what is the insurance coverage for breaking leg"
df = search(query)
df = apply_cross_encoder(query, df)
df = get_topn(3, df)
response = generate_response(query, df)
print("\n".join(response))
```

```
The insurance coverage for breaking a leg is 25% according to the policy document titled "Principle Insurance Coverage". Additional details can be found on Page 56 of the document.

**Complete Response:**
The insurance coverage for breaking a leg is 25%.

**Citations:**
- Policy Name: Principle Insurance Coverage
- Page Number: Page 56
```



```
query = 'What is the insurance coverage if the accident is caused by the driver at fault?'
df = search(query)
df = apply_cross_encoder(query, df)
df = get_topn(3, df)
response = generate_response(query, df)
print("\n".join(response))
```

```
In the event that an accident is caused by the driver at fault, the insurance coverage provided typically depends on the specific details outlined in the insurance policy document. It may include coverage

**Relevant Policy Name(s) and Page Number(s) as Citation:**
- Policy Name: TBD
- Page Number(s): Page 56

If you need further details on the specific coverage provided in case of an accident caused by the driver at fault, please refer to the policy document mentioned above and search for relevant sections rel
```

```
# Extract the top 3 results from the reranked list
# Select only the "Documents" and "Metadatas" columns for these top 3 entries
#Retrieval Augmented Generation
#query input: what is the insurance coverage for breaking leg
top_3_RAG = top_3_rerank[["Documents", "Metadatas"]][:3]
top_3_RAG
```

	Documents	Metadatas
0	One Leg 25% One Hand or One Foot 25% The Princ...	{'Chunk_No.': 3, 'Page_No.': 'Page 56'}
1	Article 2 - Benefit Qualification To qualify f...	{'Chunk_No.': 3, 'Page_No.': 'Page 53'}
2	If a Member sustains an injury, and as a resul...	{'Chunk_No.': 3, 'Page_No.': 'Page 56'}

Next steps: Generate code with top_3_RAG | View recommended plots | New interactive sheet

Run cell (Ctrl+Enter)
cell executed since last change

executed by Maimana Kowatly

ugmented Generation