Building Effective Search Systems HelpMateAI

Insurance Policy Q&A (RAG) System

A Retrieval-Augmented Generation (RAG) system for answering natural-language questions over insurance policy documents.

Built on top of Hugging Face model and ChromaDB for vector storage.

1. Background

Insurance companies produce large and complex documents filled with policy details and legal language. These are hard for customers and agents to navigate manually. Traditional keyword-based search tools are often inadequate as they cannot understand the context or semantics of queries.

This project uses RAG (Retrieval-Augmented Generation), combining document retrieval and generative AI to provide accurate, conversational responses based on the contents of insurance documents.

2. Problem Statement

The main challenge is enabling users to ask questions about insurance policies and get accurate, context-aware responses.

Traditional search systems:

- Do not understand the intent behind queries.
- Cannot handle paraphrased or context-dependent questions.
- Require manual scanning of documents.

Objective: Build a generative question-answering system that uses vector similarity and a language model to retrieve and generate accurate responses from policy documents.

3. Approach

1. **The Embedding Layer**
    a. **Document Ingestion:** Load insurance PDFs using tools like pdfplumber.
    b. **Text Chunking:** Break the text into manageable sections or chunks for better indexing and retrieval.
    c. **Embedding Generation**: Convert text chunks into vector embeddings using models like sentence-transformers.

2. **The Search Layer:**
    a. **Queries:** Design at least 3 queries against which you will test your system.
    b. **Vector Storage:** Store embeddings in a vector database (e.g., ChromaDB) for fast similarity search.
    c. **Retrieval:** Implement the re-ranking block using a range of cross-encoding models on HuggingFace.

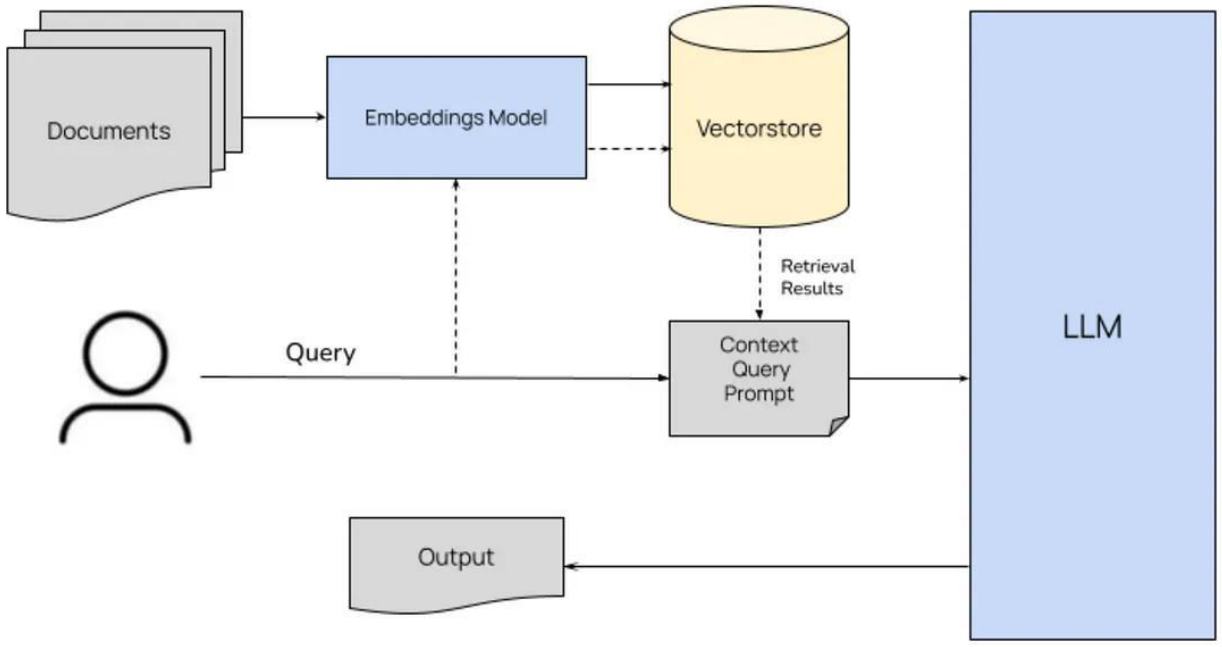
3. **The Generation Layer:** Provide the retrieved chunks as context to a Hugging Face LLM to generate the answer.

4. System Layers

1. **Data Layer:**
    a. **Read & Process PDF:** For loading pdf files (pdfplumber) and cleaning the raw text.
    b. **Document Chunk:** Use basic chunking technique, and chunking the text with fixed size
2. **Embedding Layer:** Generates embeddings with SentenceTransformer with all-MiniLM-L6-v2 model.
3. **Index Layer:** Stores and manages these embeddings for similarity-based search.
4. **Retrieval Layer:** Identifies top-k relevant chunks for a given query.
5. **LLM Layer:** A language model generates a contextual response using retrieved chunks.

5. System Architecture

Insurance PDFs → Document Loader & Text Preprocessing → Text Chunking → Embedding Model → ChromaDB (Vector Store) → Semantic Search → (Optional Reranking) → Hugging Face LLM → Final Answer



6. Prerequisites

- Python 3.x or above
- Required libraries:
    - pdfplumber
    - sentence-transformers
    - chromadb
    - langchain
    - transformers
- Hugging Face API token (if using hosted models)
- ChromaDB setup (local or cloud-based)
- GPU (optional but recommended for embedding and LLM inference)

7. Query Screenshot

```
[ ]: query = 'what is the life insurance coverage for disability'
     df = search(query)
     df = apply_cross_encoder(query, df)
     df = get_topn(3, df)
     prompt = build_prompt(query, df)
     response = generate_answer(prompt, tokenizer, model)
     print(response)
```

```
Dependent's Life Insurance terminates because the Member's Coverage During Disability as described in PART IV, Sectio
n A, ceases because Total Disability ends and the Member does not return to Active Work within 31 days ; or (6) the D
ependent's Life Insurance terminates because the Member's Accelerated Benefits Premium Waiver Period as described in
PART IV, Section A, ceases and the Member does not qualify for Coverage During Disability.
```

```
[ ]: 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)
     prompt = build_prompt(query, df)
     response = generate_answer(prompt, tokenizer, model)
     print(response)
```