Semantic Spotter Project Submission

Prepared by:

Maimana Kowatly

## 1. Background

This project demonstrates how to **build a RAG (Retrieval-Augmented Generation) system** in the **insurance domain** using LangChain.

## 2. Problem Statement

The goal of the project is to **develop a robust generative search system** that can accurately answer questions from a collection of **policy documents**.

## 3. Document

The policy documents used for this project are stored in a specified directory.

## 4. Approach

LangChain is a **framework** that simplifies the development of **LLM-based applications** by providing a suite of **tools, components, and interfaces**. It allows seamless integration with various data sources, making it a **versatile solution** for creating advanced AI-driven applications.

LangChain supports multiple **LLM providers**, including OpenAI, Cohere, and Hugging Face, and is available in **Python and JavaScript/TypeScript**. It follows a **modular and compositional design** that enables developers to build complex applications efficiently.

**LangChain Framework Components**

* **Model I/O:** Interfacing with language models, prompts, and output parsers.
* **Retrieval:** Loading and transforming documents, embedding text, and storing vectors.
* **Chains:** Creating sequences of LLM calls for structured workflows.
* **Memory:** Persisting application state between chain executions.
* **Agents:** Allowing chains to decide which tools to use based on directives.
* **Callbacks:** Logging and streaming intermediate steps.

## 5. System Layers

**Reading & Processing PDF Files**

We use **LangChain’s PyPDFDirectoryLoader** to read and process PDF files from a given directory.

**Document Chunking**

We utilize **LangChain’s RecursiveCharacterTextSplitter**, which splits text based on a hierarchical order: \n\n, \n, , and ``. This ensures that paragraphs, sentences, and words remain as intact as possible.

**Generating Embeddings**

We employ **OpenAIEmbeddings** from LangChain, which creates vector representations of text. These embeddings are useful for **similarity search, text comparison, and sentiment analysis**.

**Storing Embeddings in ChromaDB**

We store embeddings in **ChromaDB**, backed by **LangChain’s CacheBackedEmbeddings** for efficient retrieval.

**Retrievers**

A **retriever** is an interface that returns documents based on an **unstructured query**. It is more general than a **vector store**, as it doesn’t need to store documents but only retrieve them. We use **VectorStoreRetriever**, the most widely supported retriever in LangChain.

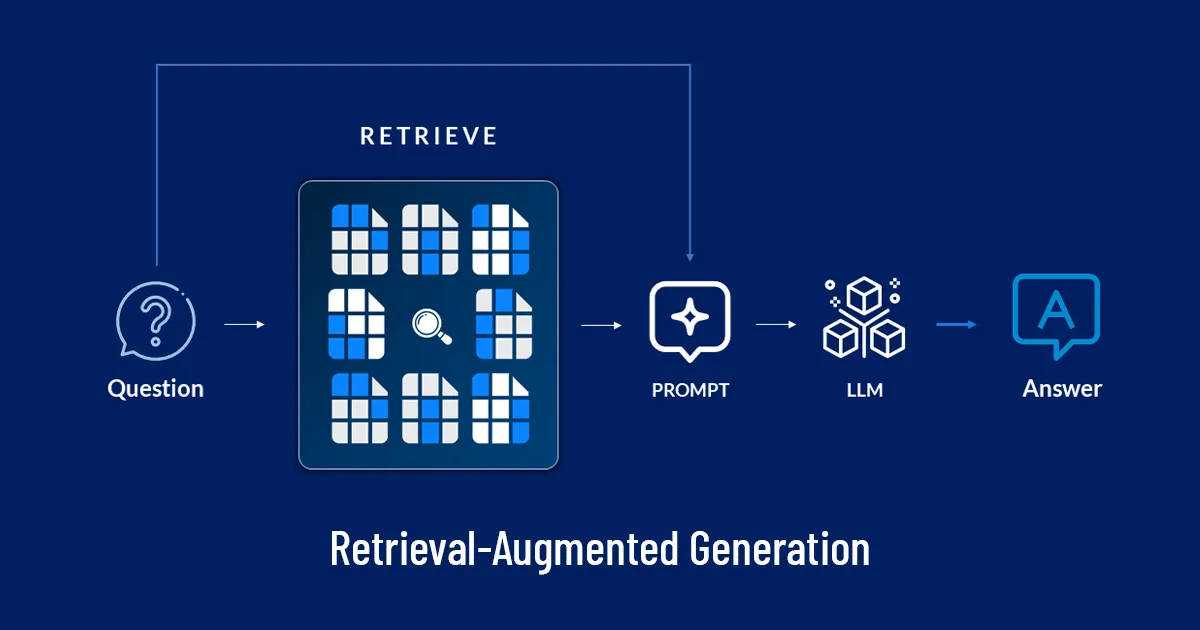
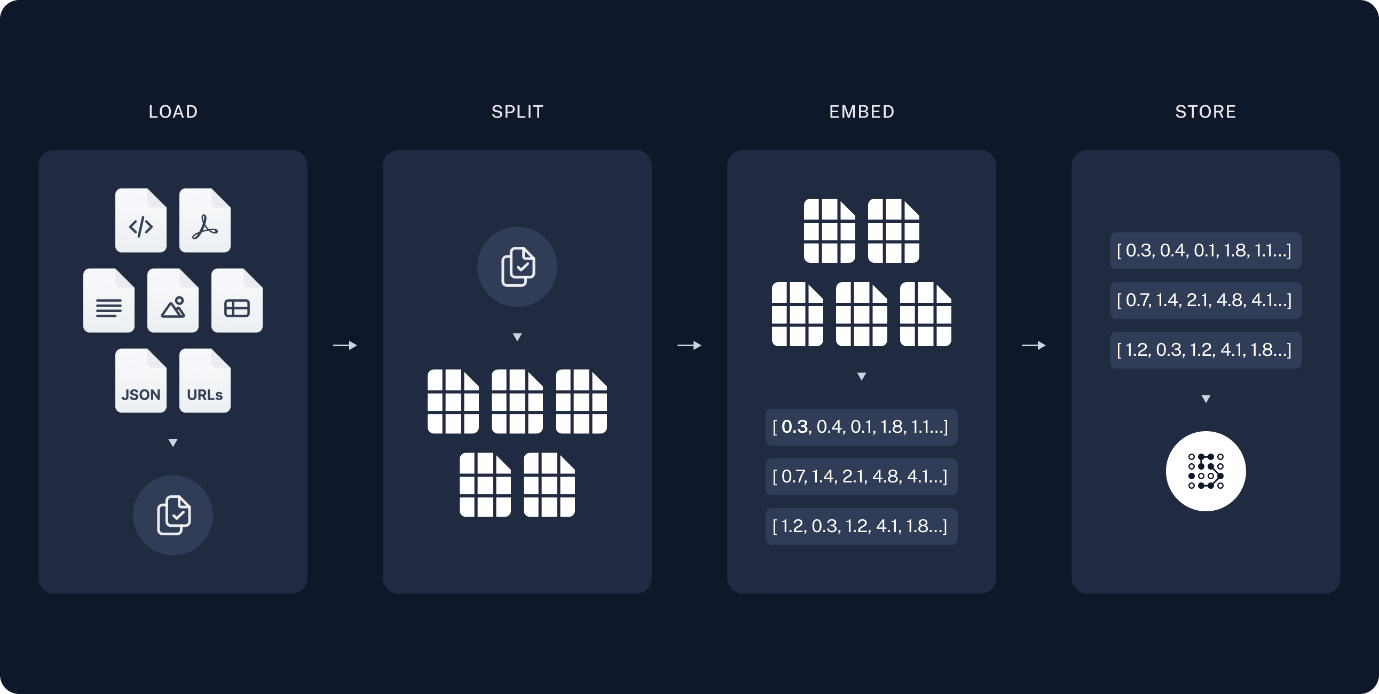
**Re-Ranking with a Cross Encoder**

To improve the relevance of retrieved results, we employ **HuggingFaceCrossEncoder** with the model BAAI/bge-reranker-base. This enhances ranking accuracy by scoring query-response pairs.

**Chains**

LangChain’s **Chains** allow the integration of multiple components. We use a **RAG chain** with the **rlm/rag-prompt** template from LangChain Hub to structure and refine the final output.

This project leverages LangChain’s capabilities to build a **powerful, intelligent, and context-aware search system** for insurance policy documents.

6.System Architecture:  


## 7. Implementation:

**The policies are based on UAE for the following insurance companies in United Arab Emirates (UAE)**

**-Thiqa.**

**-Daman.**

**-GIG (AXA previously).**

**Example test query:**

query = "what is the life insurance coverage for Kidney failure ?"

rag\_chain.invoke(query)

**answer:**

The life insurance coverage for kidney failure, specifically end-stage renal failure, includes regular renal dialysis or kidney transplantation deemed medically necessary by a nephrologist. Acute reversible kidney failure that only requires temporary renal dialysis is not covered. The coverage is contingent on the condition being chronic, irreversible, and necessitating ongoing treatment.

## 8. Prerequisites

* Python 3.7+
* langchain 0.3.13
* chromadb
* faiss-cpu
* docarray
* langchain-community
* tiktoken
* pypdf

## 9.Github repo:

<https://github.com/Maimana-Kowatly/life-Insurance-policies-AI-assistant.git>

* ensure that you add your OpenAI API key named "API\_KEY\_JS.txt"
* Open the notebook in jupyter and run all cells.