Semantic Spotter Project

1. Overview

This project demonstrates the creation of a **Retrieve-and-Generate (RAG)** system within the **insurance domain** using the LangChain framework.

2. Objective

The primary goal of this project is to develop a highly efficient and accurate generative search system that can respond to queries based on a set of insurance policy documents.

3. Documentation

You can find the policy documents https://github.com/kafee651/Semantic-Spotter-Project/tree/main/Policy%2BDocuments

4. Approach

LangChain is a powerful framework designed to simplify the development of language model (LLM) applications. It offers various tools, components, and interfaces that make it easier to build LLM-centric solutions. LangChain connects with language models like **OpenAI**, **Cohere**, and **Hugging Face**, providing flexibility to developers.

Key features of LangChain include:

- **LLM Interface**: Easy interaction with multiple language model providers.
- **Data Integration**: Seamless connection with various data sources.
- **Modular Design**: Offers a highly composable and flexible framework to develop complex applications.
- **Open-Source Framework**: LangChain supports both Python and JavaScript/TypeScript for building applications.

LangChain's core building blocks consist of:

- **Model I/O**: Interfacing with LLMs and Chat models, including prompt creation and output parsing.
- **Retrieval**: Managing documents and data sources (e.g., document loaders, transformers, and text embedding models).
- **Chains**: Constructing sequential LLM calls to perform tasks.
- **Memory**: Persisting the state across different runs of a chain.

- **Agents**: Dynamically selecting tools for chains based on high-level directives.
- **Callbacks**: Logging and streaming intermediate steps in chains.

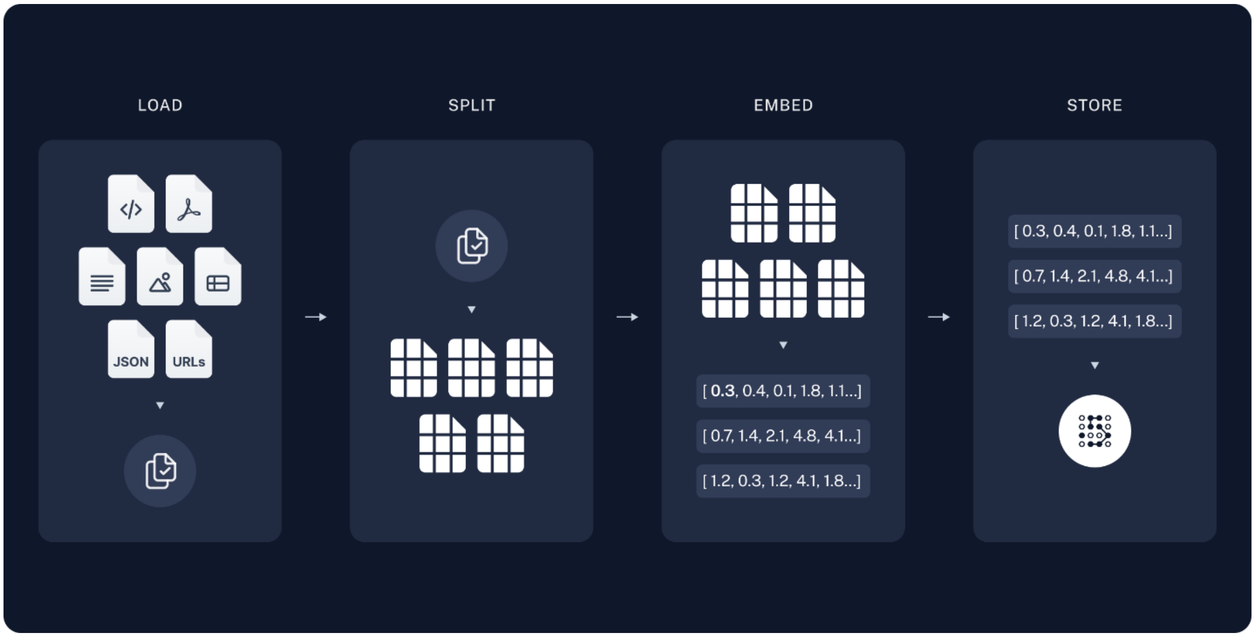
5. System Architecture

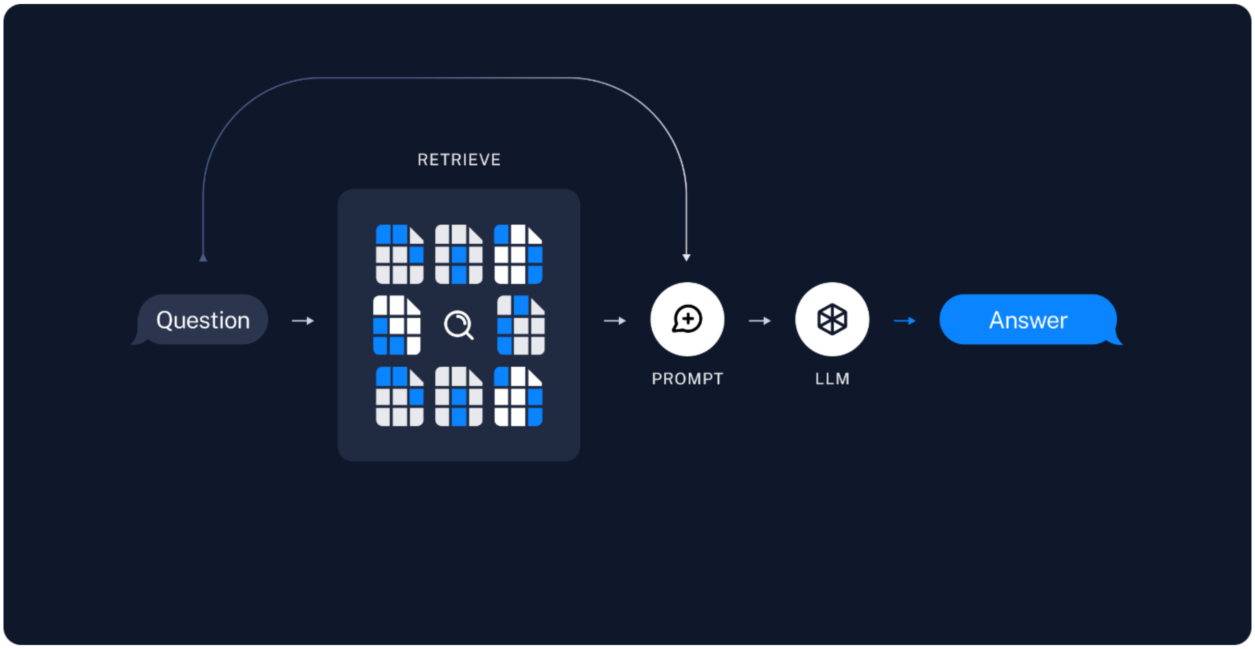
The system is designed to interact with PDF files and perform document processing, data chunking, and embedding. Here's an overview of the components used:

Key Components:

- **PDF File Processing**: LangChain's PyPDFDirectoryLoader reads and processes PDF files from the specified directory.
- **Document Chunking**: We use LangChain's RecursiveCharacterTextSplitter to split documents into chunks. This is effective for general text and preserves semantic structure, keeping paragraphs and sentences together.
- **Text Embeddings**: For creating vector representations of documents and queries, we use OpenAIEmbeddings. These embeddings facilitate similarity searches and text analysis.
- **ChromaDB for Embedding Storage**: Embeddings are stored in **ChromaDB** and are backed by LangChain's CacheBackedEmbeddings.
- **Retrievers**: We use **retrievers** to query unstructured data. The most widely supported type is the VectorStoreRetriever.
- **Re-Ranking with Cross Encoders**: We use a HuggingFaceCrossEncoder (BAAI/bge-reranker-base) to improve the relevance of retrieved documents by re-ranking them based on the query.
- **Chains**: LangChain enables chaining various components together to create a cohesive application. For example, we use a prompt chain from the LangChain hub (rlm/rag-promp) for the RAG system.

6. System Architecture Diagrams





7. Prerequisites

Before running the system, ensure the following:

- Python 3.7+ installed.
- LangChain version 0.3.13 or higher.
- Obtain an OpenAI API key and add it to the variable

8. Setup & Execution

To get started:

1. Clone the repository
2. $ git clone https://github.com/kafee651/Semantic-Spotter-Project.git
3. Open the notebook in jupyter and run all cells.