Semantic Spotter Project Report

1. Background:

This project demonstrates the implementation of a RAG (Retrieval-Augmented Generation) system within the insurance domain using LangChain.

2. Problem Statement:

The primary objective of this project is to construct a robust generative search system capable of effectively and accurately answering inquiries posed from a comprehensive collection of policy documents.

3 Approach:

LangChain is a comprehensive framework that simplifies the development of language model (LLM) applications. It offers a suite of tools, components, and interfaces that facilitate the construction of LLM-centric applications. LangChain empowers developers to build applications capable of generating creative and contextually relevant content.

LangChain provides an LLM class specifically designed for interfacing with various language model providers, including OpenAI, Cohere, and Hugging Face.

LangChain's versatility and flexibility enable seamless integration with diverse data sources, making it a comprehensive solution for developing advanced language model-powered applications.

LangChain's open-source framework is accessible for application development in Python or JavaScript/TypeScript. Its core design principle emphasizes composition and modularity, allowing developers to swiftly construct complex LLM-based applications by combining modules and components. LangChain's open-source nature facilitates the development of powerful and personalized applications leveraging LLMs relevant to user interests and needs. It facilitates external system integration to access pertinent information for solving intricate problems. LangChain provides abstractions for essential functionalities required for building an LLM application and integrates seamlessly with data reading and writing operations, thereby expediting the application development process. LangChain's framework enables the construction of applications agnostic to the underlying language model, with its continuously expanding support for various LLMs offering a unique value proposition for application development and continuous iteration.

LangChain framework comprises the following components:

Components: LangChain provides modular abstractions for the essential components required to interact with language models. Additionally, it offers comprehensive implementations of these abstractions. These components are designed to be user-friendly, regardless of whether they are utilized within the LangChain framework or not.

Use-Case Specific Chains: Chains are designed to assemble specific components in a particular manner to effectively accomplish a particular use case. These chains serve as a higher-level interface, enabling users to easily initiate a specific use case. Additionally, chains are customizable.

The LangChain framework is built upon the following fundamental components:

Model I/O: Interface with language models (LLMs and Chat Models), prompts, and output parsers.

Retrieval: Interface with application-specific data, including document loaders, document transformers, text embedding models, vector stores, and retrievers.

Chains: Construct sequences or chains of LLM calls.

Memory: Persist application state between runs of a chain.

Agents: Allow chains to select appropriate tools based on high-level directives.

Callbacks: Log and stream intermediate steps of any chain.

4. System Layers

Reading & Processing PDF Files: We will be using LangChain P yPDFDirectoryLoader to read and process the PDF files from specified directory.

Document Chunking: We will be using LangChain RecursiveCharacterTextSplitter. This text splitter is the recommended one for generic text. It is parameterized by a list of characters. It tries to split on them in order until the chunks are small enough. The default list is ["\n\n", "\n", " ", ""]. This has the effect of trying to keep all paragraphs (and then sentences, and then words) together as long as possible, as those would generically seem to be the strongest semantically related pieces of text..

Generating Embeddings: We will be using OpenAIEmbeddings from LangChain package. The Embeddings class is a class designed for inter facing with text embedding models. LangChain provides support for most of the embedding model providers (OpenAI, Cohere) including sentence transformers library from Hugging Face. Embeddings create a vector representation of a piece of text and supports all the operations such as similarity search, text comparison, sentiment analysis etc. The base Embeddings class in LangChain provides two methods: one for embedding documents and one for embedding a query.

Store Embeddings In ChromaDB: In this section we will store embedding in ChromaDB. This embedding is backed by LangChain CacheBackedEmbeddings

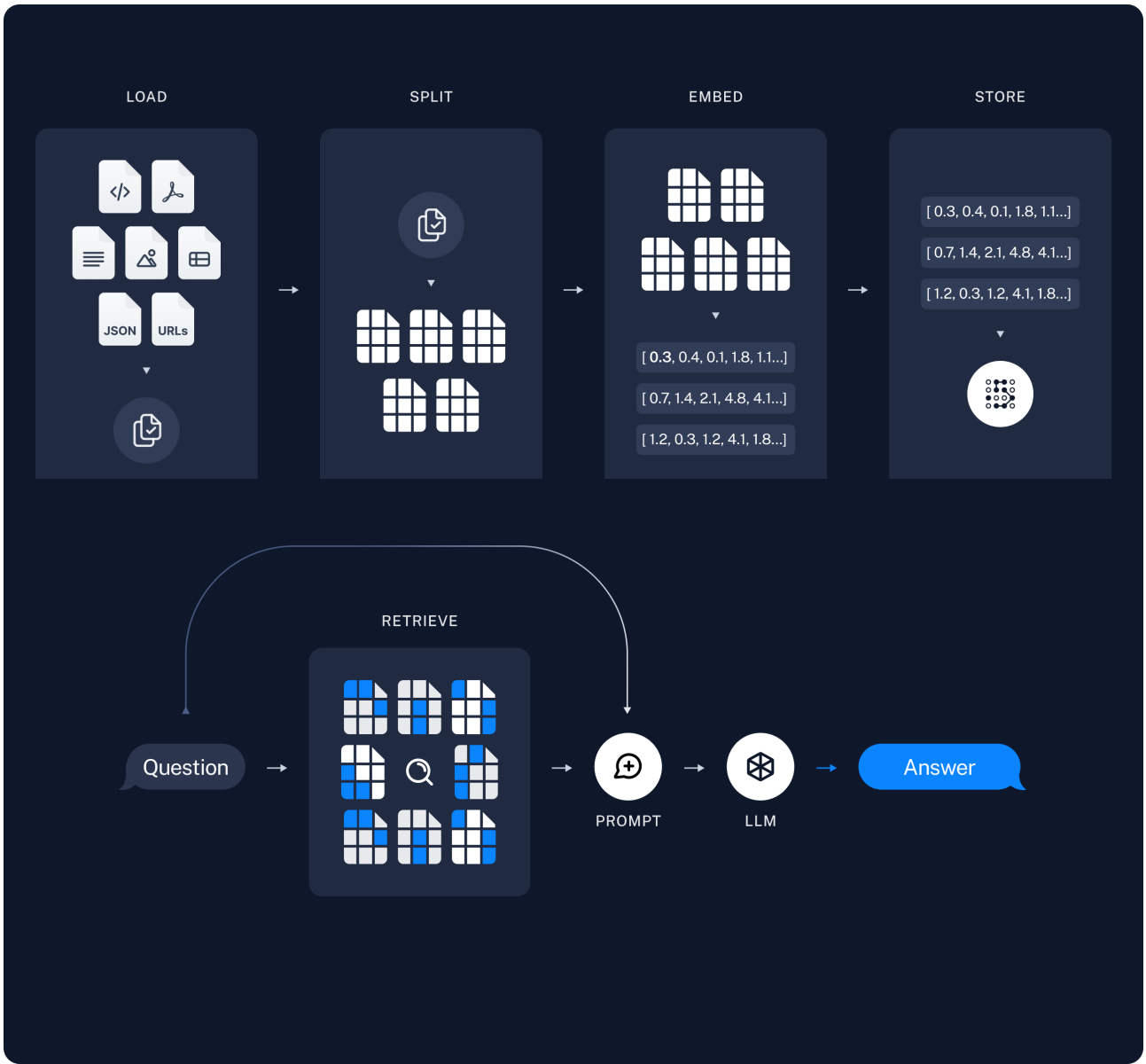
Retrievers: Retrievers provide Easy way to combine documents with language models.A retriever is an inter face that returns documents given an unstructured query. It is more

general than a vector store. A retriever does not need to be able to store documents, only to return (or retrieve) them. Retriever stores data for it to be queried by a language model. It provides an inter face that will return documents based on an unstructured query. Vector stores can be used as the backbone of a retriever, but there are other types of retrievers as well. There are many different types of retriever, the most widely supported is the VectorStoreRetriever.

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. The above retriever is associated with HuggingFaceCrossEncoder with model BAAI/bge-reranker-base

Chains: LangChain provides Chains that can be used to combine multiple components together to create a single, coherent application. For example, we can create a chain that takes user input, formats it with a PromptTemplate, and then passes the formatted response to an LLM. We can build more complex chains by combining multiple chains together, or by combining chains with other components. We are using pulling prompt rlm/rag-promp from langchain hub to use in RAG chain.

5. System Architecture:



6. Prerequisites
P ython 3.7+
langchain 0.3.13
Please ensure that you add your OpenAI API key in order to access the OpenAI API.