Symantic Spotter with LlamaIndex

Project Objective

Build a project in the insurance domain, similar to the project you saw in the 'Retrieval Augmented Generation' session.

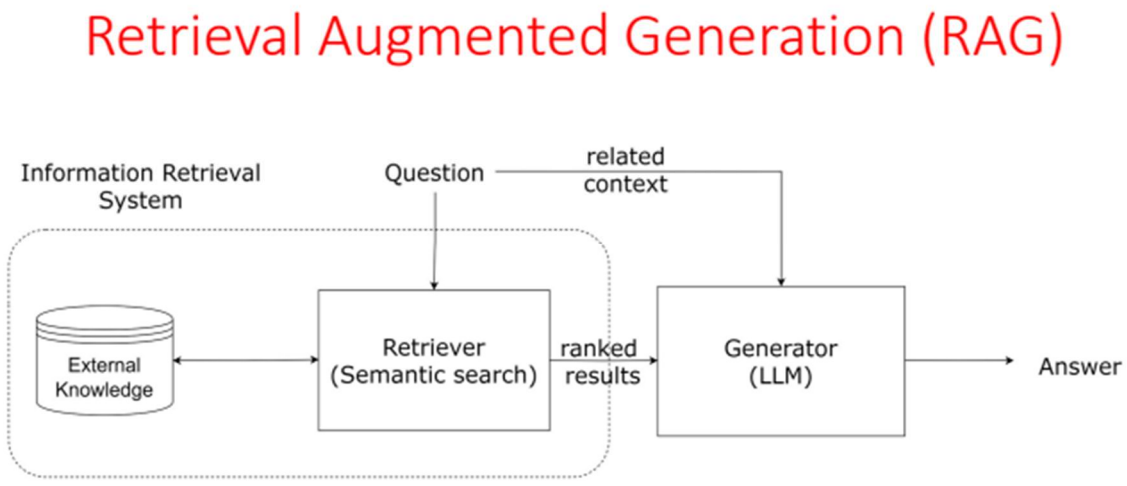
The goal of the project will be to build a robust generative search system capable of effectively and accurately answering questions from various policy documents. We are given set of life insurance documents. Our purpose is to build a RAG application for efficient searching in the documents with Llama Index.

Solution Strategy

Build a solution which should solve the following requirements using LlamaIndex:

- Users would responses from insurance policy knowledge base.
- If user want to perform a query system must be able to response to query accurately.

Recap for RAG



Goal

Solving the above two requirements well in and would ensure that the accuracy of the overall model is good.

Data Used

HDFC various Insurance policy documents stored in single folder.

Tools used

LlamaIndex, ChatGPT has been used due to its powerful query engine, fast data processing using data loaders and directory readers as well as easier and faster implementation using fewer lines of code.

Model Used

OpenAI from Llama_Index with model="gpt-3.5-turbo"

Why LlamaIndex ?

LlamaIndex is an innovative data framework specially designed to support LLM-based RAG framework application development. It offers an advanced framework that empowers developers to integrate diverse data sources with large language models

LlamaIndex includes a variety of file formats, such as PDFs and PowerPoints, as well as applications like Notion and Slack and even databases like Postgres and MongoDB.

The framework brings an array of connectors that assist in data ingestion, facilitating a seamless interaction with LLMs. Moreover, LlamaIndex boasts an efficient data retrieval and query interface.

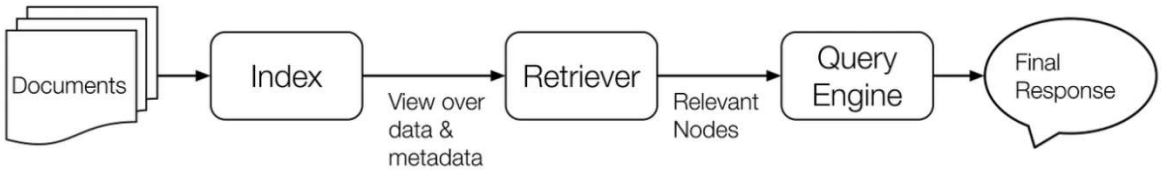
LlamaIndex enables developers to input any LLM prompt and, in return, receive an output that is both context-rich and knowledge-augmentation.

Key Feature of LlamaIndex:

● Data connectors allow ingestion from various data sources and formats.

● It can synthesize data from multiple documents or heterogeneous data sources.

● It provides numerous integrations with vector stores, ChatGPT plugins, tracing tools, LangChain, and more.



Documents and Nodes

Documents in LlamaIndex may be different from your traditional perception of documents. Document and Node objects are core abstractions within LlamaIndex.

A **Document** is a generic container around any data source - for instance, a PDF, an API output, or retrieved data from a database. They can be constructed manually or created automatically via data loaders. By default, a Document stores text along with some other attributes. Some of these are

metadata - a dictionary of annotations that can be appended to the text (basically, additional info about the document)
relationships - a dictionary containing relationships to other Documents/Nodes.

A **Node** represents a "chunk" of a source Document, whether that is a text chunk, an image, or other. Like Documents, they contain metadata and relationship information with other nodes.

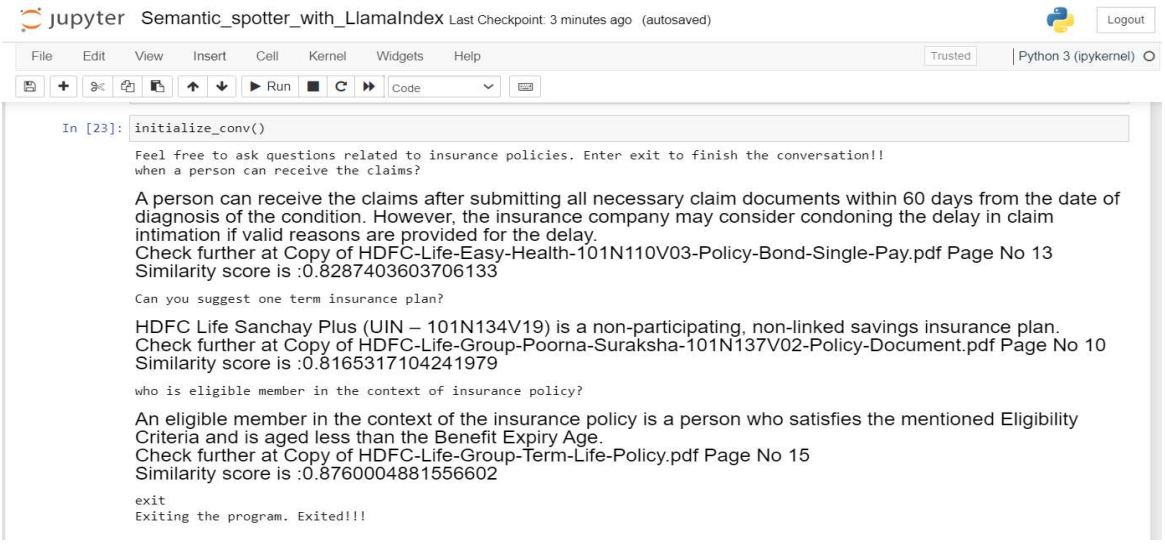
Nodes are a first-class citizen in LlamaIndex. You can choose to define Nodes and all its attributes directly. You may also choose to "parse" source Documents into Nodes through our NodeParser classes. By default, every Node derived from a Document will inherit the same metadata from that Document (e.g. a "file_name" filed in the Document is propagated to every Node).

Both Documents and Nodes have unique identifies called ID. These can be set automatically or manually. ID is generally used to identify, update, and define relationships between documents (or nodes).

Generative Search Response from Insurance documents:

Below are attached custom query generative search results.

Conversational Message – Single Query:



Testing pipeline – Multiple Queries:

Code

Jupyter Notebook developed for Semantic Spotter.

Challenges Faced

Tried to get better score with different techniques.

Faced challenges to implement caching mechanism.

Future Work

Based on the current POC, it can be understood where to improve further for better results.

Overall score can be more efficient and better by cleaning up data

Some more tools can be explored to improve the POC further and for more better accuracy and fast retrieval.