Insurance Agent – RAG on Insurance Data using LlamaIndex

### Objective

Insurance documents are often lengthy and complex. The objective of this project is to develop an efficient Generative Search system for the insurance domain. The system will be robust and capable of accurately answering questions derived from policy documents. This will enable users to easily retrieve relevant information from numerous documents, each with a large number of pages. For this project, we will utilize a collection of sample HDFC insurance policy documents provided in previous modules.

### Design Choice

We will develop a Retrieval Augmented Generation (RAG) based Agent to assist the user in their search. A RAG system can be implemented in several ways, such as using existing libraries with a custom approach or utilizing frameworks like LangChain or LlamaIndex that simplify the process of building RAG systems.

The current implementation utilizes LlamaIndex due to its emphasis on search and retrieval. Additionally, LlamaIndex is designed for efficiency, particularly in managing large data volumes. It provides a simple interface for interacting with LLMs, which facilitates the processes of searching and retrieving data.

### System Design

The following section describes the system design of this application. It outlines the different layers involved in building this system and the LlamaIndex components used in each layer.A diagram of a diagram

AI-generated content may be incorrect.

**Figure 1. RAG System Using LlamaIndex**

1. **Data Loading**

Data connectors and data loaders facilitate connections to various data sources and support loading of diverse file types. Currently, our policy documents are available in PDF format. For the current implementation, we utilize SimpleDirectoryReader to read multiple files from a directory. This method is employed to load files into LlamaIndex as documents for subsequent processing.

1. **Index Creation**

Each document is segmented into smaller units called nodes using the SimpleNodeParser. The indexing process involves generating nodes and linking embeddings with the data within each node. The vector storage contains the embedding vectors of processed document segments. In the current implementation, VectorStoreIndex from LlamaIndex is utilized. This approach employs LlamaIndex's default use of text-embedding-ada-002 from OpenAI. Furthermore, VectorStoreIndex relies on an in-memory SimpleVectorStore.

1. **Building Query Engine**

The Retriever is an essential component within LlamaIndex that is responsible for obtaining the most pertinent data nodes based on a user's query. The Response Synthesizer processes the user's query in conjunction with the retrieved nodes to generate a suitable response. The Query Engine serves as the central mechanism that integrates all other components, encompassing the index, retriever, and response synthesizer. It interprets the user's query, initiates the retrieval process, and produces a response that is subsequently presented to the user. The Query Engine is derived from the VectorStoreIndex established during the index creation phase. By default, LlamaIndex utilizes OpenAI's GPT-3.5-turbo model.

### Steps of Project Execution:

The pre-requisites for executing this project are Python and OpenAI API key.

To get started with the Insurance Agent, execute the python notebook submitted for this project. And follow the input prompts generated by the conversation function. To end the conversation at any point type “exit” in the input prompt.

### Output

The following are the screenshots of results from the system. For each query, it contains the responses from the RAG system and the corresponding citations as well from the documents provided as data store.

