**HelpMate AI – Project Case Study**

Submitted by: Akanksha Dwivedi, Snehal Kedar; DSC64

**1. Project Goal**

The primary goal of this project is to develop a robust and highly effective generative search system designed to answer questions accurately and efficiently based on various insurance policy documents. With the growing complexity of insurance policies, it is often challenging for users to navigate through lengthy and dense texts to find the specific information they need. This system aims to bridge that gap to generate precise and contextually relevant responses to user queries.

The process begins when a user submits a question related to a specific insurance policy document. The system processes the query and generates an answer by analysing the content of the policy document. The chatbot can respond to complex insurance-related queries, such as coverage details, claims procedures, exclusions, and more, in a highly accurate and context-aware manner. The response is tailored to address the user’s question in a clear and concise manner, ensuring that the information provided is both relevant and understandable. In cases where further clarity or verification is needed, the system can also cite the exact section or clause within the policy document from which the answer was derived. This helps the user cross-reference and validate the response, ensuring transparency and trust in the generated information.

**2. Data Sources**

Policy documents are used for this project having information of Policy plan, benefits, terms and conditions, claim procedure etc.

Files can be downloaded from the Link

**3. Design**

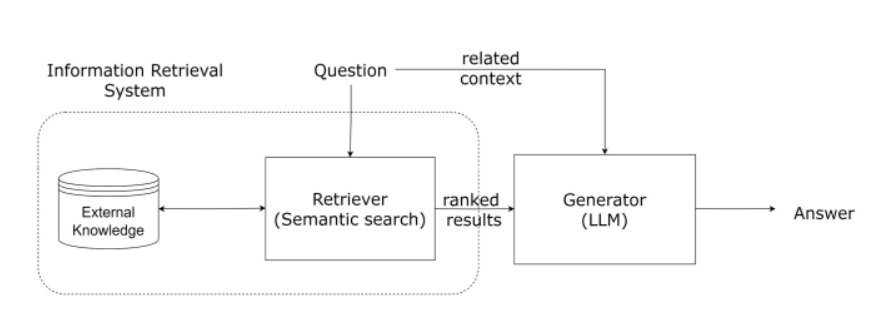
The proposed solution for designing a chatbot to answer insurance-related queries using Large Language Models (LLM) and Retrieval-Augmented Generation (RAG) must focus on ensuring rapid data processing, accurate information retrieval, and reliable answer generation while maintaining a simple and efficient implementation. By leveraging RAG, the system can seamlessly enhance the capabilities of LLMs by incorporating external knowledge sources—such as insurance policy documents in this project, ensuring that the responses are both relevant and trustworthy.

Incorporating LlamaIndex for RAG integration is an excellent choice for this system due to below key advantages:

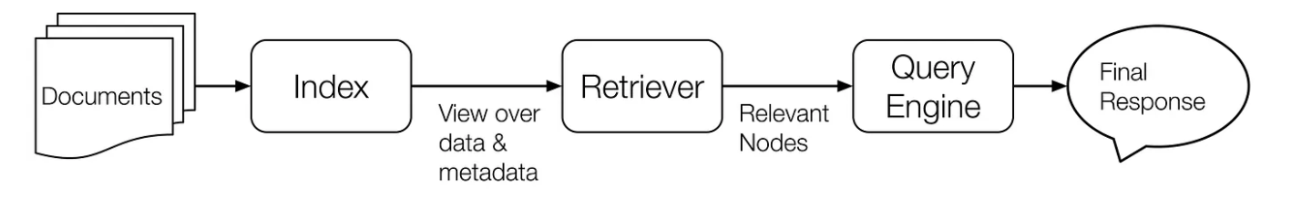
1. **Efficient Document Indexing**: LlamaIndex ensures that documents, including lengthy insurance policies, are indexed effectively, allowing for rapid and efficient searching and retrieval of relevant information.
2. **Integration with Generative Models**: LlamaIndex offers seamless integration with powerful generative models, such as LLMs, enabling them to generate answers that are both informed by up-to-date knowledge and aligned with the user’s specific query.
3. **Scalability**: The system is scalable, ensuring it can handle an increasing volume of queries and documents as the insurance database grows. This ensures that the solution remains effective and responsive even as more data is added over time.
4. **Ease of Use and Rapid Implementation**: LlamaIndex simplifies the process of implementing RAG-based systems. Its user-friendly design enables quicker deployment, reducing the time required for development and testing. This makes it a great choice for businesses looking to quickly integrate a chatbot solution without excessive complexity.
5. **Contextual Search and Advanced Retrieval**: LlamaIndex enhances the system’s ability to conduct contextual searches, ensuring that the retrieved information is not only accurate but also contextually relevant to the user’s question. This advanced retrieval capability ensures that even complex or nuanced queries receive appropriate answers.

**4. System Architecture**

RAG - System Architecture



The basic RAG pipeline in LlamaIndex is illustrated below.



**5. High-Level Workflow**

1. **Document Loading:**

Load the insurance policy documents using data loaders.

1. **User Query:**

The user submits a query related to a specific aspect of an insurance policy (e.g.,"What is Claims Procedure for HDFC-Life-Easy-Health?").

1. **Document Retrieval:**

The system retrieves the most relevant sections of the document using LlamaIndex's powerful query engine.

1. **Augmentation:**

Combine the retrieved sections from the policy document with the original user query to create a more informative context for the response. This allows the generative model to access both the query and the relevant document content.

1. **Answer Generation:**

A generative model (LLM) is used to produce a coherent and contextually accurate answer, drawing from the retrieved information.

1. **Returning Response:**

The final response is returned to the user, If required, the response includes specific references to the document.

**6. Various Layers**

* Document Ingestion and Preprocessing Layer: Loads and processes documents into a usable format.
* Text Chunking Layer: Breaks documents into smaller chunks for better retrieval.
* Embedding Layer: Converts text into vector embeddings for efficient similarity comparison.
* Retrieval Layer: Uses semantic search to retrieve the most relevant document chunks based on the user’s query.
* Generation Layer: Generates a response using the retrieved chunks and the user’s query.
* Post-Processing and Formatting Layer: Refines and formats the answer, including citations and references.
* User Interface Layer: Allows users to submit queries and view responses.

**7. Potential Challenges**

* Insurance policies are often written in complex legal language with intricate clauses and exceptions. This can make it difficult to retrieve and generate precise answers.
* User queries might be ambiguous, which could lead to incorrect or incomplete answers being generated. For example, a user could ask, “What is the coverage for dental procedures?” without specifying the type of dental procedure.
* Complex queries may require retrieving information from multiple sections or clauses, and integrating those details into a coherent answer.

**README**

This README is to provide a guide on how to deploy and use the RAG-based policy document Q&A system with LlamaIndex, from installation to running the system.

Before running the project, make sure you have the following installed:

* Python 3.7+
* pip (Python package manager)
* Install the necessary Python packages

Below are the steps.

1. Import the necessary libraries.
2. Mount your Google Drive and Set the API key.
3. Use the appropriate document loader for loading the documents.
4. Building the query engine. The general process for creating the query\_engine is:

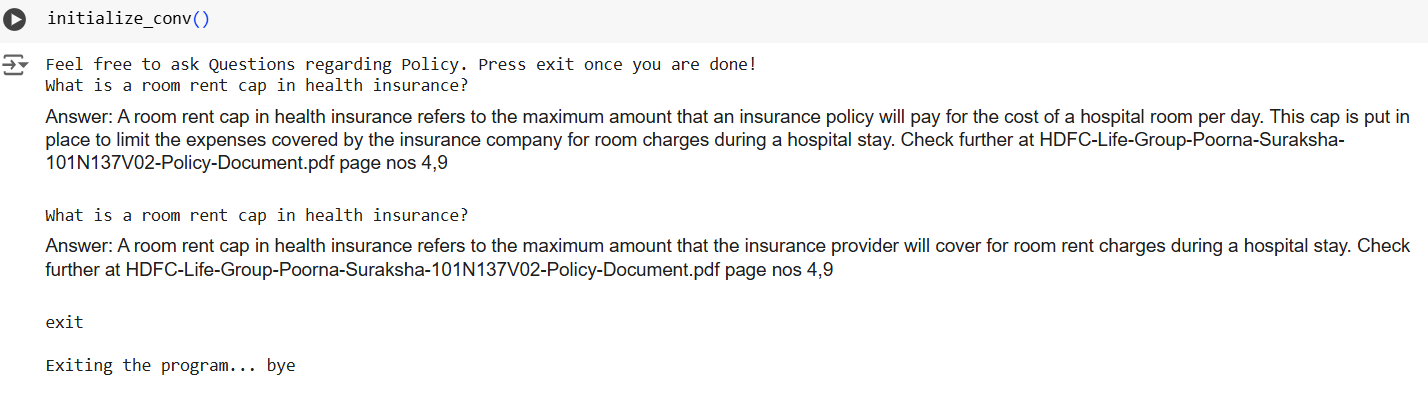
* Load the documents
* Create nodes from the documents
* Create index from documents
* Initialise the Query Engine
* Query the index with the prompt
* Generate the response using the retrieved nodes

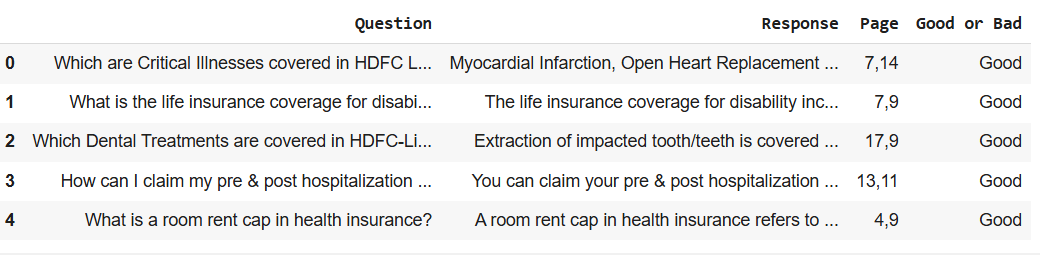
1. Creating a Response Pipeline

A Query Response pipeline encapsulates all the necessary steps to build a RAG pipeline. Modify the functions query\_response and initialize\_conv() below. The query\_response functions return the query response from the query engine along with the supporting documents and the initialize\_conv() function creates an interactive chatbot.

1. Build a Testing Pipeline

Here we feed a series of questions to the Q/A bot and store the responses along with the feedback on whether it's accurate or not from the user. Create questions and store them in the questions list to be queried by the RAG system using the testing\_pipeline function.



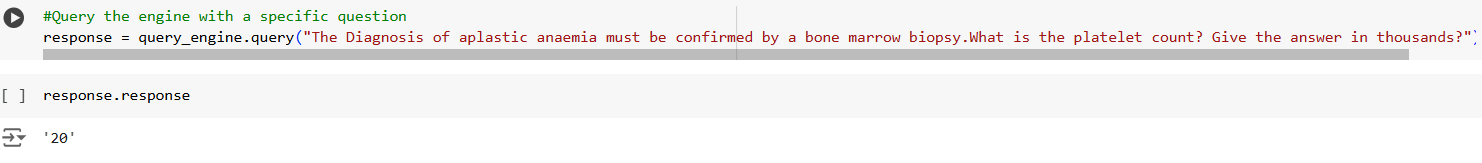


1. Recommendations on how to further improve this

Based on the testing pipeline's feedback develop a strategy on how to improve the POC further This can be through building a better/cleaner dataset, or utilising better data pre-processing techniques If the accuracy is good enough, think of implementing some more use cases and user stories. Maybe a set of users want to do XYZ, and that requirement needs to be implemented in the solution. Think of a proper tool that can enable you to do that. Few suggestions based on the common tools that we have in the pipeline.

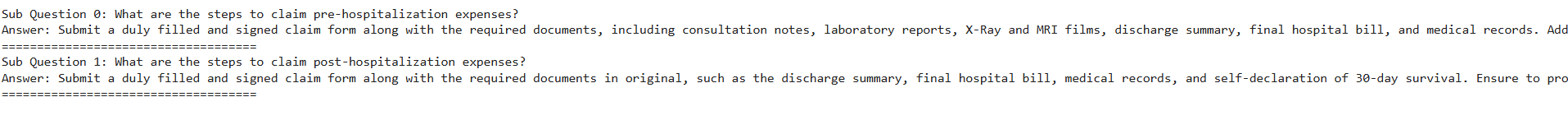
**Suggestion 1** - Using Customized Nodes and LLMs

This can be used if your responses are not accurate or is not being summarized very well.



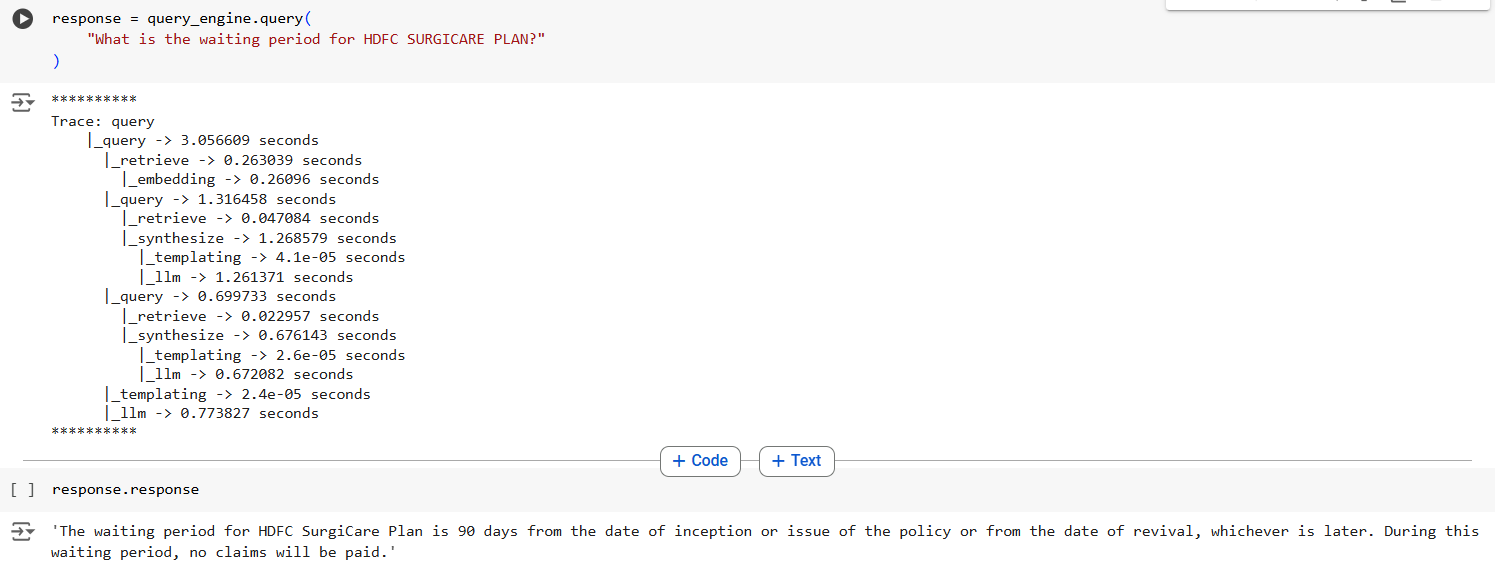
**Suggestion 2** - Sub Question Query Engine

Sub question query engine is used to tackle the problem of answering a complex query using multiple data sources. It first breaks down the complex query into sub questions for each relevant data source, then gather all the intermediate responses and synthesizes a final response.



**Suggestion 3**: Retriever Router Query Engine

We define a router query engine based on a retriever. The retriever will select a set of nodes, and we will in turn select the right QueryEngine. We use ToolRetrieverRouterQueryEngine class for this!



**Conclusion**

**The goal** of the project to build a robust generative search system that effectively and accurately answers questions from various insurance policy documents using Retrieval Augmented Generation (RAG) with LlamaIndex **has been successfully achieved.**”

The recommendations from the testing pipeline's feedback have been implemented. This includes improving the dataset, applying more effective data pre-processing methods, and addressing new user stories and use cases, by using Sub Question Query Engine and Retriever Router Query Engine.

**Sub Question Query Engine**

Sub question query engine is used to tackle the problem of answering a complex query using multiple data sources. It first breaks down the complex query into sub questions for each relevant data source, then gather all the intermediate responses and synthesizes a final response.

**Retriever Router Query Engine**

We define a router query engine based on a retriever. The retriever will select a set of nodes, and we will in turn select the right QueryEngine.