**PROJECT REPORT: HELPMATE AI PROJECT**

**Retrieval Augmented Generation with LlamaIndex**

1. **Problem Statement:**

We are building a project in the insurance domain. The goal of the project will be to build a robust generative search system capable of effectively and accurately answering questions from various insurance policy documents.

LlamaIndex, formerly known as GPT Index, is a data framework designed to facilitate the integration, structuring and access to private or domain-specific data for applications that utilise large language models (LLMs), such as GPT-3 or similar systems. It is a Python library for building data-aware applications and question-answering systems.

1. **Why Llama Index:**

LlamaIndex supports ingesting data from various sources like text files, web pages, PDFs, and even structured data like CSV files. Once data is ingested, it builds an index that allows for efficient retrieval and querying of the data. It provides a query engine that enables users to ask natural language questions and receive relevant answers from the indexed data. It is an ideal framework as it offers customizable retrieval strategies, such as vector similarity search, keyword search, and more, to find the most relevant data for a given query.

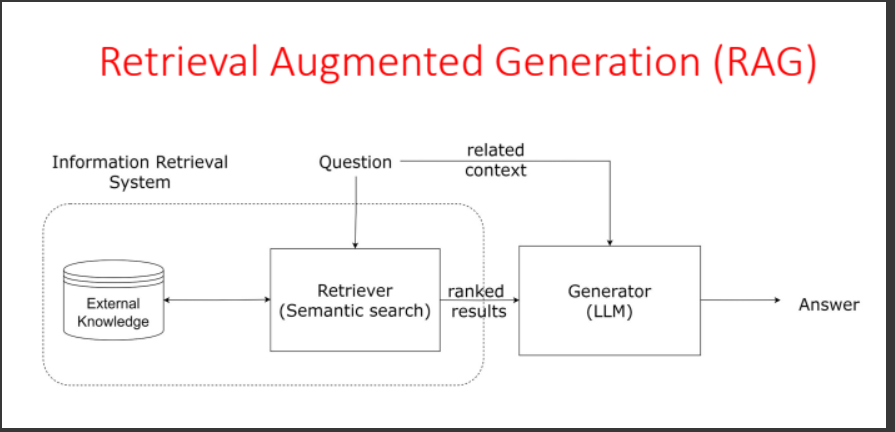
1. **Objectives:**

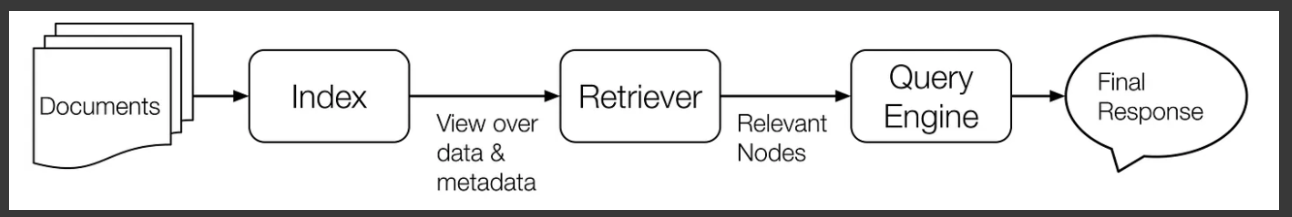
The primary objectives of the project are as follows:

* Build a robust generative search system capable of effectively and accurately answering questions from various insurance policy documents
* Use LangChain or LlamaIndex to build the generative search application.

1. **Design**

RAG System Architecture





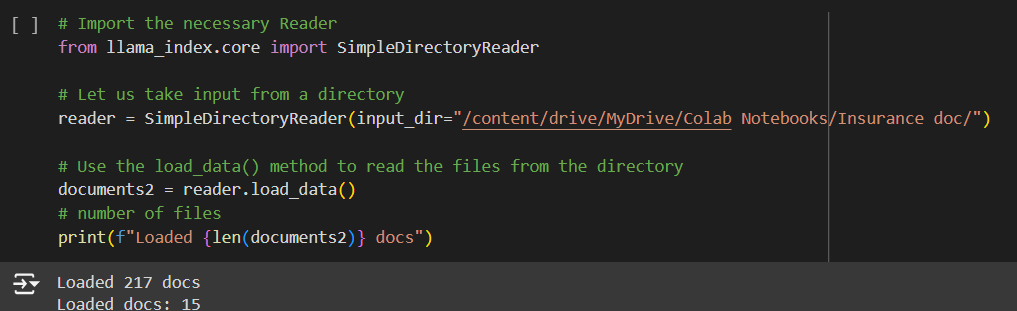
Components:

* Data Loading
* Building query engine
* Creating a Response Pipeline
* Build a Testing Pipeline
* Optimization and fine tuning

1. **Implementation**

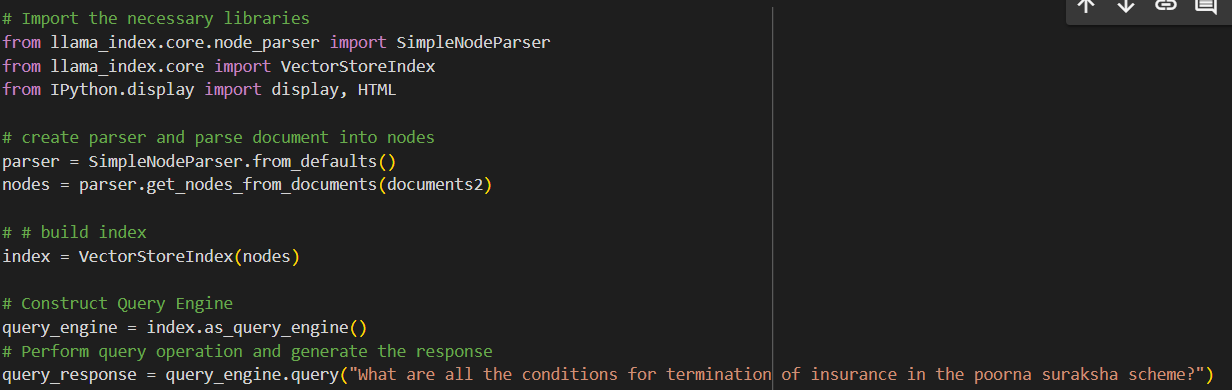
* Data Loading:

Used the Simple Directory Reader and read the data from the PDFs given as source data and load the data into a variable.



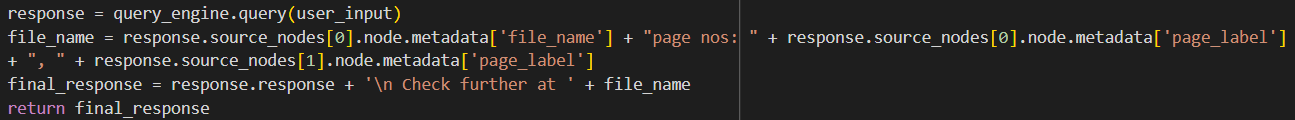
* Building query engine:

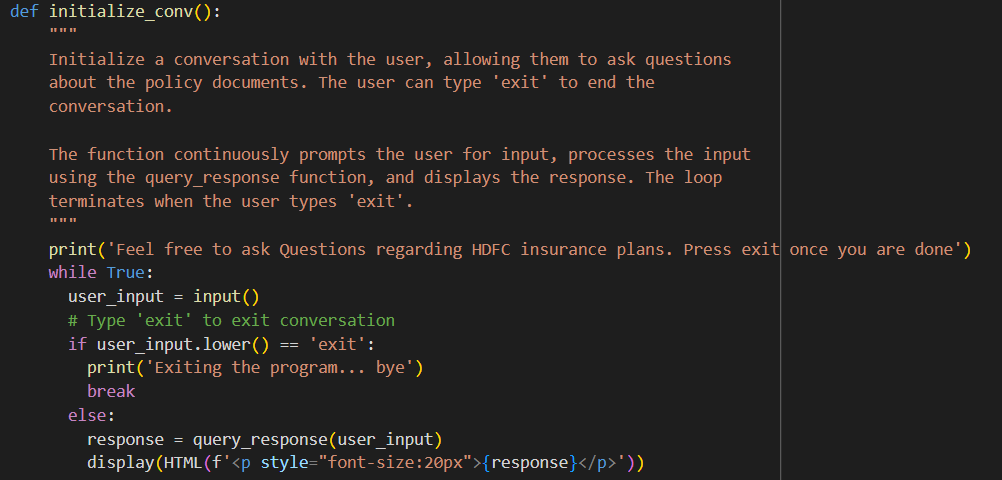
A Query Engine wraps a Retriever and a ResponseSynthesizer into a pipeline. It uses the query string to fetch nodes and then sends them to the Large Language Model (LLM) to generate a response.



* Creating a response pipeline:

This involves a query response function and initialize\_conv function. A Query Response pipeline encapsulates all the necssary steps to build a RAG pipeline and the initialize\_conv() function creates an interactive chatbot.





* Build a Testing Pipeline:

Created 3 questions and store them in the questions list to be queried by the RAG system using the testing\_pipeline function.

* Optimization and fine tuning:

Based on the testing pipeline’s feedback, we can improve the performance by building a custom prompt template, Using customised notes and templates, Sub question query engine, Build RAG using Agentic Systems

1. **Challenges:**
2. Data Extraction from PDFs - Used Simple DirectoryReader to extract all the PDFs from the folder directly
3. Indexing Large Documents - Use efficient indexing strategies and consider distributed or incremental indexing if the dataset is extremely large.
4. Handling Domain-Specific Language - Fine-tune the language model on a corpus of insurance-related texts to improve its understanding and generation capabilities1
5. Ensuring Relevant Document Retrieval - Optimize the retrieval mechanism using advanced techniques like SentenceSplitter from llama.index.core.node\_parser
6. Combining Retrieved Documents and Query - Develop a structured approach to concatenate the query and retrieved documents, ensuring the input remains within the model’s maximum token limit.
7. **Lessons Learnt:**
8. Importance of Robust Data Extraction
9. Effective Data Preprocessing is Crucial
10. Retrieval Mechanism Optimization
11. Managing Input Length Constraints
12. Evaluation is Multifaceted
13. Handling Sensitive Information