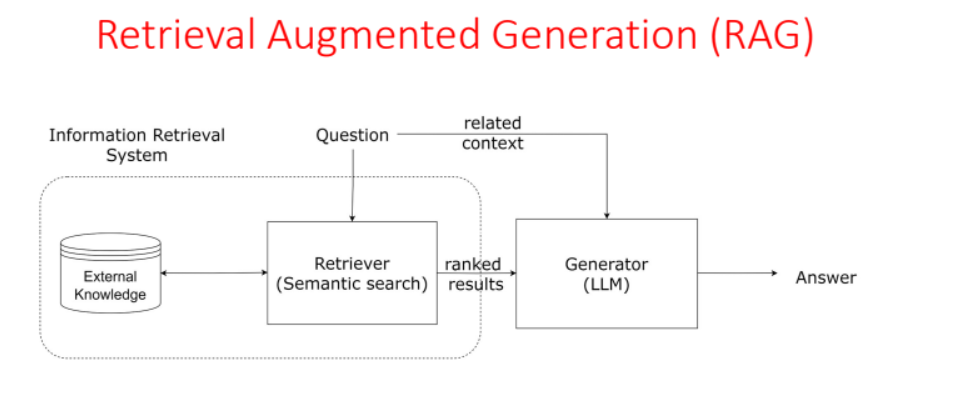
**Project Report**

**Help Mate AI Project report: Retrieval Augmented Generation with LlamaIndex**

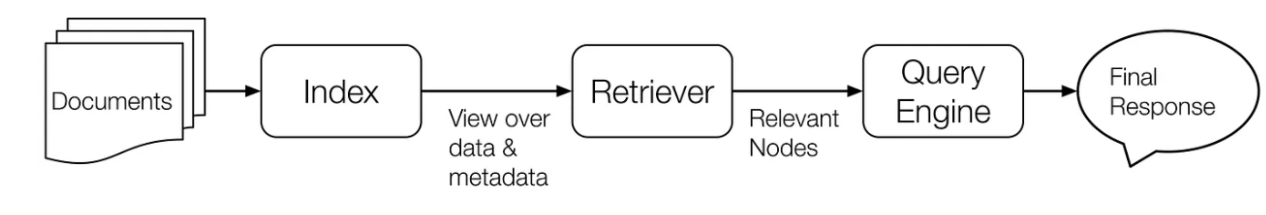
**Problem statement:** We have to build a project in the insurance domain using ‘**Retrieval Augmented Generation with LlamaIndex**’. The goal of the project will be to build a robust generative search system capable of effectively and accurately answering questions from various insurance policy documents.

**Why LlamaIndex used:** 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. LlamaIndex facilitates the retrieval of relevant information from large datasets and feeds it as context to LLMs, enabling them to generate accurate and context-aware responses. 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.

**Design of the model:**



The basic RAG pipeline in LlamaIndex is illustrated below:



The below are the Components of RAG:

• Data Loading

• Building query engine

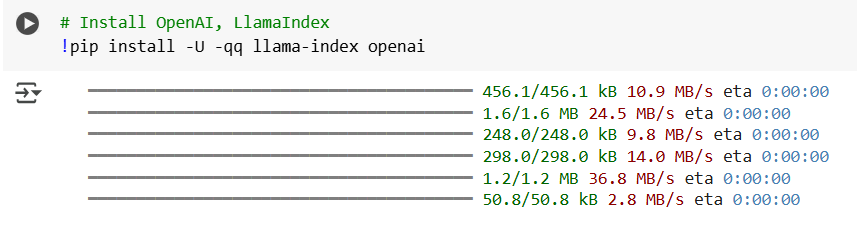
• Creating a Response Pipeline

• Build a Testing Pipeline

• Optimization and fine tuning

**Implementation:**

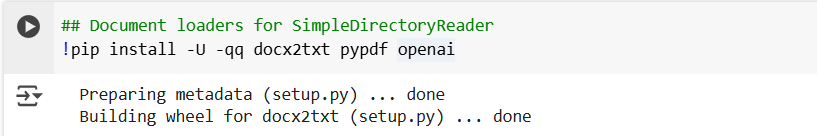
1. Importing Open AI and LlamaIndex library:

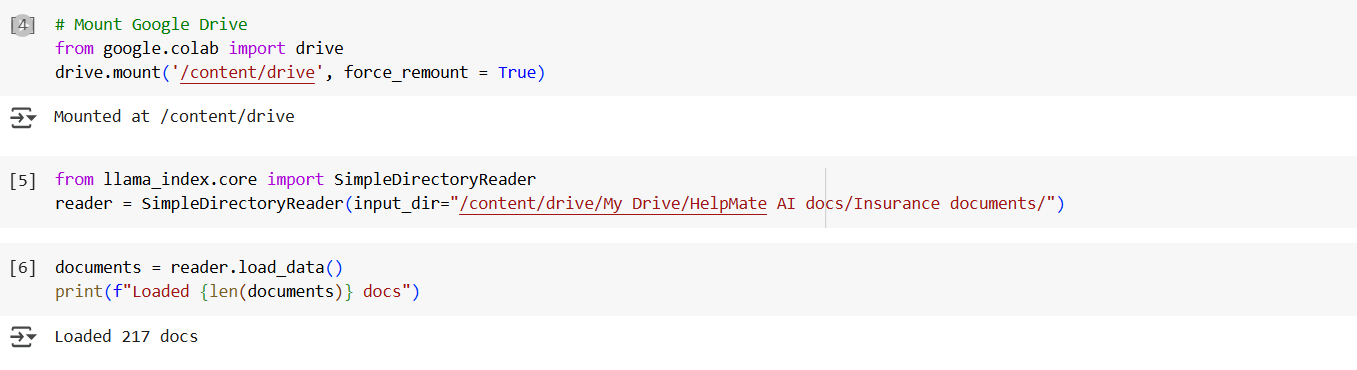




1. Document loaders for SimpleDirectoryReader

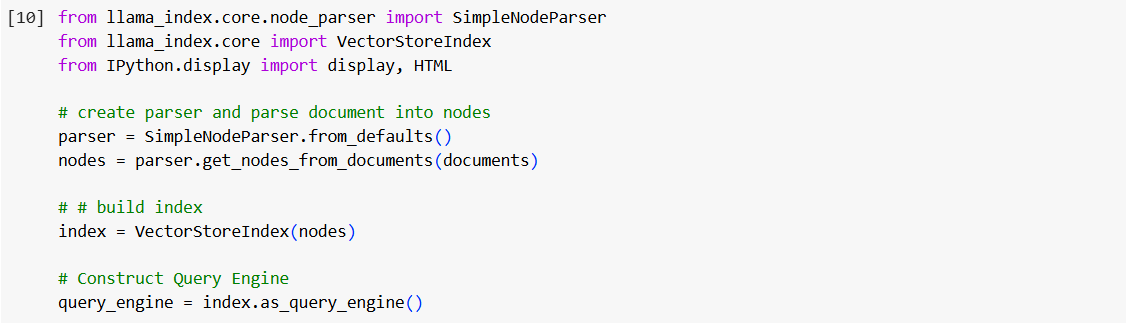
Used the Simple Directory Reader and read the data from the PDFs given as source data and load the data into a variable.

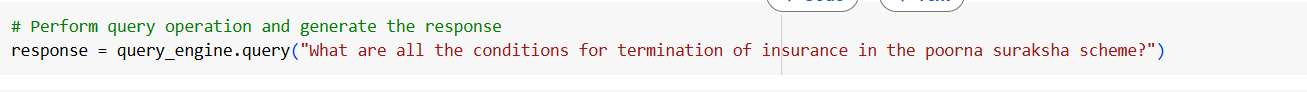




1. Building Query Language

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.





1. Creating a response pipeline:

This involves a query\_response and initialize\_conv() function. 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

We have created atleast 5 questions and store them in the questions list to be queried by the RAG system using the testing\_pipeline function.

**Challenges faced:**

**Extracting Data from PDFs**: Leveraged the SimpleDirectoryReader to efficiently extract content from all PDFs within the specified folder.

**Indexing Large Documents**: Adopted efficient indexing strategies, and considered distributed or incremental indexing for handling extremely large datasets.

**Managing Domain-Specific Language**: Fine-tuned the language model on a corpus of insurance-related texts to enhance its comprehension and generation capabilities.

**Ensuring Accurate Document Retrieval**: Improved the retrieval mechanism using advanced techniques, such as SentenceSplitter from llama.index.core.node\_parser.

**Merging Retrieved Documents and Queries**: Designed a structured method to concatenate queries and retrieved documents while ensuring the input stays within the model's token limit.