TASK 3 (LLM CHAT ASSISTANT WITH DYNAMIC CONTEXT BASED ON QUERY)

0.1 Problem Statement

Design and implement a basic LLM based chatbot of your choice to dynamically get context via external sources (via api) and documents and answer the user query.

1 INTRODUCING OUR HEALTH INSURANCE CHAT BOT - YOUR ULTIMATE HEALTH COMPANION!

1.1 About

This innovative application simplifies the management of our Health Insurance Plan, offering easy access to your policy details through a comprehensive and user-friendly Policy Document. Seamlessly integrating external data from LIC (Life Insurance Corporation of India), our chat bot expands its capabilities to provide users with real-time information about various life insurance products. With a focus on efficiency, the app enables quick resolution of policy-related queries, ensuring clarity and transparency in understanding coverage terms. Additionally, the Health Companion app fetches contact details and addresses from relevant websites, making it effortless for users to connect with their insurance provider.

METHODOLOGY-APPLICATION DESIGN

2 DATA SOURCES

To keep things straightforward, I've opted for two data sources.

- PDF: The initial data source is a PDF obtained through internet
- LIC : This data source corresponds to the website link of LIC.

LlamaIndex serves as a versatile and straightforward data framework designed to seamlessly integrate custom data sources with expansive language models. It equips us with essential tools to enhance our Large Language Model (LLM) applications by effortlessly incorporating diverse datasets.

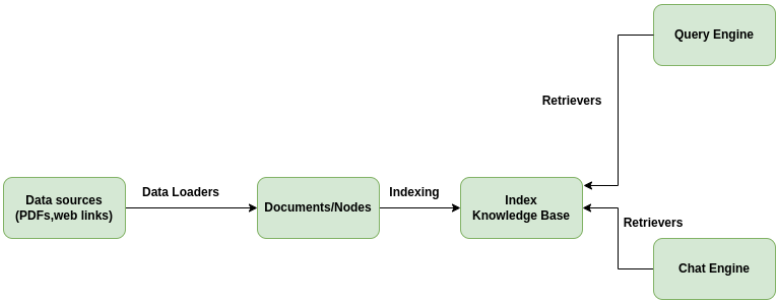

Figure 1. High level Architecture of the Application

3 DATA CONNECTORS

I've employed the SimpleDirectoryReader from Llamaindex to load the PDF data, while the download_loader is utilized for extracting information from the website.

4 MODELS

4.1 LLM

In the initial stages of constructing an LLM-based application, a crucial decision involves selecting the appropriate LLM to employ. LLMs serve as integral components within LlamaIndex, functioning either independently or seamlessly integrated into other core modules like indices, retrievers, and query engines.

My choice for the LLM in this task is the Open Source **meta-llama/Llama-2–7b-chat-hf**, and we assess its performance in terms of memory and computation. The goal is to ensure compatibility for running it on a T4 GPU within the free tier on Colab.

4.2 Embeddings

Within LlamaIndex, embeddings play a crucial role in expressing our documents/data through an advanced numerical representation, providing a sophisticated means of characterizing the data.Text input is processed by embedding models, generating an extensive list of numerical values that effectively encapsulate the semantic nuances inherent in the given text.

In this task , I have used **sentence-transformers/all-MiniLM-L6-v2** embedding.

4.3 Service Context

I use **SentenceSplitter** that split the text while respecting the boundaries of sentences. This function is used to break down large bodies of text into smaller sections, ensuring that sentences aren't split in the middle and making it easier to process or analyze text data in manageable portions.

A prevalent application of an embedding model involves placing it within the **ServiceContext** object and subsequently utilizing it for the creation of an index and conducting queries. The input documents are segmented into nodes, with the embedding model generating a unique embedding for each node. During query execution, the embedding model is once again employed to embed the query text.

4.4 Index Setup

An Index is a data structure composed of Document objects, designed to enable querying by an LLM.Indexers split your Document into Node objects, which are similar to Documents (they contain text and metadata) but have a relationship to their parent Document.

.from_documents() method which accepts an array of Document objects and will correctly parse and chunk them up.When we want to search your embeddings, our query is itself turned into a vector embedding, and then a mathematical operation is carried out **VectorStorendex** to rank all the embeddings by how semantically similar they are to your query.

4.5 Storing

Once we have data loaded and indexed, we want to store it to avoid the time and cost of re-indexing it. By default, our indexed data is stored only in memory.

4.5.1 Vector Stores

The **VectorStorendex** takes our Documents and splits them up into Nodes. It then creates vector embeddings of the text of every node, ready to be queried by an LLM.I am using Chroma as a vector store for this task which is an open soruce Vector db.

4.6 Querying

Now we have loaded our data, built an index, and stored that index for later,its time for querying. Querying is a prompt to an LLM: so it can be a question and get an answer, or a request for summarization, or a much more complex instruction.

4.6.1 Query Engine

The foundation for all querying lies in the **QueryEngine**. Retrieval,Postprocessing and Response synthesis are three stages of Query Engine.Llamaindex provides numerous choices for Response Modes.To keep things simple, I opted for the refine mode.

5 CODE

Please refer the following notebook for this assignment:

Code : **Task_3.ipynb**

6 IMPROVEMENT SUGGESTIONS

- Alternative models, such as Falcon ,Zephyr7balpha, mistralai etc, can be explored for experimentation.
- Various embeddings, such as hkunlp/instructorlarge, can be investigated for exploration.
- Various vector databases like Faiss , Weaviate,Pinecone can be investigated for exploration.

MY PORTFOLIO

Please feel free to explore my projects:

- Github