Assignment - Building Applications with LLMs

Question Answering System

Introduction

Recently, GenAI systems have taken the world by storm, with applications in almost all domains. Retrieval-augmented generation (RAG) enabled the use of pre-trained models by providing the right context to the LLM.

Providing the right context from a knowledge source (e.g., a set of documents) allows:

- Incorporating domain knowledge into the LLM
- Reducing the LLM hallucination

Problem Statement

1. Business Problem

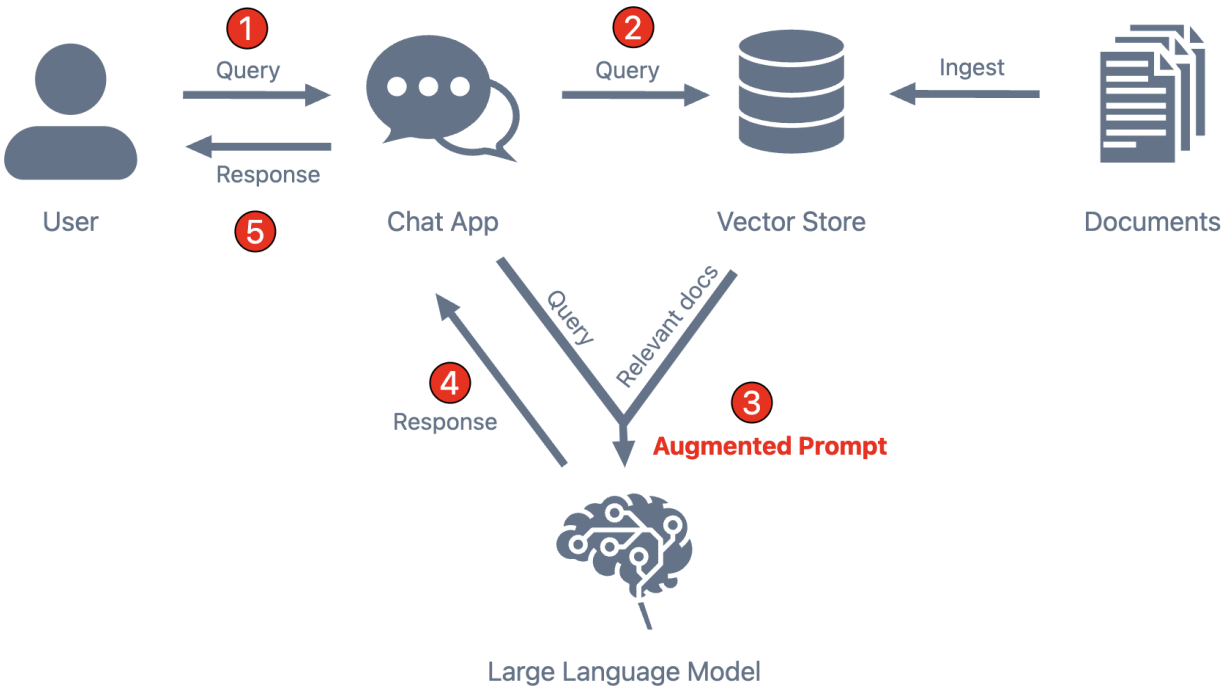
You work at a company, XYZ, that builds the best chatbots.
A typical client for XYZ has a website and wants to add a virtual assistant that can help me find answers to my questions.

A client has heard about the success of XYZ. So, that client decided to get help from XYZ. The client will give XYZ a URL, and XYZ will build that virtual assistant (or chatbot) that answers questions only from the content on the website.

2. ML Problem

As the first step, the team's tech lead took that business problem and mapped it to an ML problem. This will allow the tech lead to break down the project into smaller tasks.
The tech lead decided to formulate the problem as a RAG system. Which consists of the following components (as depicted in the below figure)

- Data sources (documents): In this use case, we will have a single webpage
  - Hint: So you need to use the correct loader from LangChain
- Vector store: to store the document representation
- LLM to answer user questions



The project is significant to XYZ, and the Tech Lead, with the help of product and program managers, came up with the following list of tasks for the team:

1. We need to chunk (split) the input document to fit the maximum length of the model. You can check the following resources.

    a. https://python.langchain.com/v0.1/docs/modules/data_connection/document_transformers/character_text_splitter/
    b. https://python.langchain.com/v0.1/docs/modules/data_connection/document_transformers/recursive_text_splitter/

2. For each chunk, generate embedding (feature vector). You can use
    a. OpenAIEmbeddings or
    b. Sentence transformer from Hugging face https://huggingface.co/sentence-transformers

3. Store the embeddings in a vector store. You can use
    a. https://python.langchain.com/v0.2/docs/integrations/vectorstores/faiss/

4. Prompt the LLM to use the retrieval as a source of knowledge in order to answer the user question. You can use
    a. https://api.python.langchain.com/en/latest/prompts/langchain_core.prompts.prompt.PromptTemplate.html

The tech lead suggested using Langchain as it makes the chaining of these steps much easier.

Extra Requirements

There is an extra requirement that XYZ did not promise to deliver, but it said it would try. The requirement is to make the system conversational. This means that the user can ask multiple related questions.

For example, the user can say something like

- User: I want to purchase an iPhone
- System: There are iPhone 13, 14 and 15
- User: What is the price of the oldest one

For the last query, the system should understand that the user is interested in the price of the oldest iPhone from 13, 14 and 15 without the user writing.

> **"What is the price of the iPhone 13?"**

The Tech Lead mentioned that https://python.langchain.com/v0.2/docs/tutorials/qa_chat_history/#tying-it-together can be helpful

Deliverables

Once you've completed your solution in a Google Colab notebook, please submit it using the link provided here.

You can use https://www.gradio.app/ to build a simple UI for your chatbot