**Documentation for RAG-based**

**Chatbot Assistant**

Objective of the Test

Develop and deploy a chatbot assistant that uses the RAG architecture to answer questions about the content of the Promtior website, based on the LangChain library.

Proyect Overview

Implementation Logic

* Data preparation: The content from the Promtior website was gathered and stored in a structured format. Langchain libraries were used such as DirectoryLoader, WebBaseLoader and RecursiveCharacterTextSplitter. It was then embedded into vector representations using a sentence transformer from HuggingFace and stored in a FAISS vector store for efficient retrieval.
* Model Selection: The LlaMA2 model was chosen since it was the one suggested if we didn’t have a subscription to the OpenAI API. The model was integrated using the Ollama library to facilitate interaction with LangChain.
* Chain Configuration: A retrieval-based QA chain was configured using LangChain’s RetrievalQA. The chain uses the FAISS vector store as the retriever and LlaMA2 for generating answers.
* Creating user interface: Streamlit library was used since it contains features specialized for chatbot applications.

Main Challenges

Integrating the LlaMA2 model with the LangChain library was the most exhausting problem I had. It seemed to be everything perfect, every necessary line of code was there, but I couldn’t detect the error. I read the LangChain and Ollama documentation, debugged the integration process step-by-step, watched videos, but couldn’t fixed it. The server was working, I could run LlaMA2 from the command processor. I tried a lot of changes, thinking that the problem was the way I was accessing the server in the coding syntaxis part. However, the problem was that I was trying to access from Google Collab to the local server Ollama. I then used Jupyter Notebook and the problem was solved.

Apart from this, I run the Ollama server in my 8RAM laptop working extremely slowly. However, I maintained using LlaMA2 since it was the one suggested for the test, and according to the documentation it should work although it was the limit (*“You should have at least 8 GB of RAM available to run the 7B models”).*

Moreover, another important problem I had was ensuring it interacted correctly with the retriever. The documentation showed how to do it, but with a paid license of OpenAI. Therefore I needed to find how to do it with Ollama but there were many documents and each one used some resource I didn´t have. It first showed me the following error:

***“ValueError****: Ollama call failed with status code 500. Details: {"error":"llama runner process has terminated: exit status 0xc0000005 "}”*

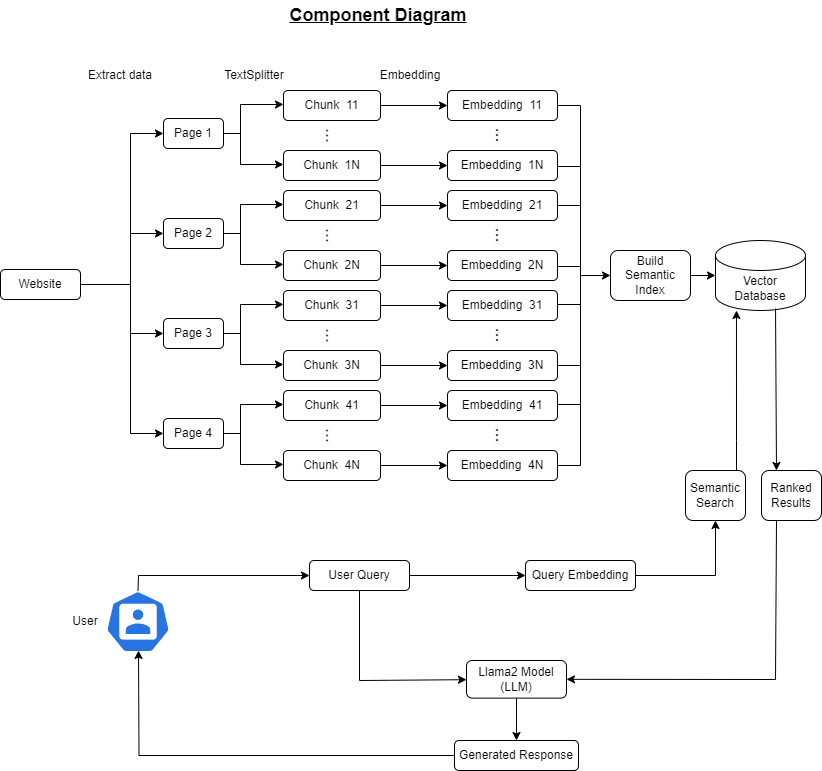
I didn´t know why, but I tried many things and believe that when adding the OpenAI Key it changed the error and printed the following one:

As I couldn´t fix this problem I chose to follow the steps from another website.

I didn’t have time to deploy it using Langserve, a deployment tool provided by LangChain, on Railway management.

Component Diagram

The following diagram shows the components involved in the solution and their interactions from the time the question is received by the chatbot until the response is given.



Made with Draw.io

The solution is divided into two parts. One is the information gathered from the website and the other the user interaction part.

Firstly, all the pages from the Promtior website are collected. Then we extract all the information from them and split the data into small chunks. Then the embeddings for each chunk are created. They are vectors which contain floating point numbers. All these embeddings are going to be stored in the vector database after building the sematic indexes. Therefore, we have all the information available on the website in this knowledge base.

Apart from this, we have user interaction. When he asks a question, embeddings are created for that question. Then a semantic search is done in the knowledge base. All the website information was stored here in the form of embeddings and organized with semantic indexes. Therefore, it is possible to do a semantic search comparing the query embeddings with the embeddings of the Promtior website data. After finding several answers they are passed to the Large Language Model (Llama2) together with the user question. Finally, the LLM will generate a natural response using the retrieved data as context. The generated answer is sent back to the user.

User Interaction:

* The user interacts with the chatbot through a web interface or API.

Query:

* The user’s question is received by the chatbot.

Retriever Component:

* The question is passed to the FAISS retriever, which searches the vector store for relevant documents.

Document Retrieval:

* The retriever fetches the most relevant documents based on the user’s question.

LLM Component:

* The retrieved documents and the user’s question are passed to the LLaMA2 model.

Answer Generation:

* The LLaMA2 model generates an answer using the retrieved documents as context.

Response Delivery:

* The generated answer is sent back to the user through the web interface or API.

Bibliography

<https://towardsdatascience.com/can-a-llama-2-powered-chatbot-be-trained-on-a-cpu-ce9ec6ebe3c2>

<https://medium.com/credera-engineering/build-a-simple-rag-chatbot-with-langchain-b96b233e1b2a>

<https://python.langchain.com/v0.1/docs/get_started/quickstart/>

<https://www.youtube.com/watch?v=UKJv1qPBElk>

<https://www.youtube.com/watch?v=M4G2zk4X93Y>