Mateusz Mulka RAG-LLM Report

The RAG LLM (ReAct Agent Large Language Model) represents a breakthrough in natural language processing technology, designed to streamline information retrieval and enhance user interaction. By leveraging advanced AI capabilities, RAG LLM acts as an intelligent agent, understanding user queries and providing relevant responses from extensive data sources. This report explores RAG LLM's functionalities and applications, highlighting its role in revolutionizing information access across various domains.

System design:

This system is organized with a usage of a few modules. Inside of tools directory you can find embeddings.py that take a crucial part in preparing the dataset for further usage of the model. You can also find prompts.py that contains instruction_str, prompt_template, context – parameters that are passed to PandasQueryEngine and ReActAgent in order to optimize querying. Due to the simplicity of the business logic in this project, no design patterns were used.

Technologies selected:

For the tech stack I have chosen Llama index, tiktoken, OpenAI API and LangChain. Those are one of the most popular tools corresponding to projects based on RAG LLM infrastructure. From Llama index I have imported Pandas Query Engine that is an engine for creating queries based on a pandas dataframe. Doing so allowed me to feed the csv file into the LLM easly. On top of that OpenAI imported from llama_index.llms.openai allowed me to connect to the gpt-4 model and set the temperature to a desired value (this variable affects the volatility and hallucinations of the model) . ReActAgent was a useful tool for combining Open AI's LLM with desired query engine based on certain query template implemented in prompts.py . On the other hand tiktoken, OpenAI API and Langchain took crucial part in creating vector embeddings. Tiktoken was used for encoding, OpenAI API delivered embeddings based on my dataset and by using Langchain's RecursiveCharacterTextSplitter I was able to make sure that not a single request sent to the API will exceed it's limit. I was able to do that by splitting some rows into parts (chunking Text column) based on their token size.

Challenges encountered:

Choosing best tools : I have made a few attempts for applying different tools before choosing this tech stack. Prior to llama indexing and OpenAI LLM model I have tried an open source LLM model and I was experimenting with LangChain's possibilities.

Vector Database : I have wanted to connect this project with a vector database. Unfortunately, for reasons beyond my control, I had limited access to the computer for several days and didn't manage to do it because of lack of time. Due to a lack of experience regarding vector databases I was also a little bit lost in all the different embedding methods already implemented in some vector databases' DBMS. Connecting this project with a vector database can be a crucial point in further development.

Creating embeddings: I have encountered a problem with passing some parts of this dataset to the embedding API. I did manage to overcome this problem by chunking certain rows of the dataset however I believe this can be improved even more.

Monitoring progress: Doing that was a little bit harder while creating RAG LLM in comparison to all the different metrics used in my prior ML/DL projects.

Future development:

This project has a huge potential for a future development. One of the areas is data storage, by connecting to a vector database the reusability and consistency of this project can be ensured.

Optimizing Vector/Context retrieval can be a huge milestone in developing this project. Preprocessing df into more efficient format like TF-IDF can be a good starting point. After connecting to a vector database it would be a great idea to create an additional query retriever based on vector store to ensure better answers generated by the model. Changing the way that the context is loaded can also help optimizing the model. Instead of reloading the entire context for each query, the context can be update incrementally based on the conversation history. This can be done by storing the conversation history and using it to inform future queries.

Experimenting with different model types and model parameters can be beneficial while trying to optimize it. Different models can have different performances regarding a certain task. One of the parameters to be interfered could be temperature, right now it is set to value 0.1 which decreases the chances of hallucinations while still giving the model a little flexibility generating answers.

Applying more advance indexing methods followed up with optimizing similarity search can improve model's accuracy while generating a response.

This project has already implemented exceptions handling mechanism but before publishing to production it would be necessary to expand it a little bit more. Probably creating exceptions dictionary that can be sent to a frontend would be also beneficial.

Further more the user's experience can be also increased. Connecting to the LLM I have set verbose parameter to True, by doing that an end used can see the insights of generating the response, however UX can be improved further more. One of way of doing that could be implementing UI and sending model's response to a certain endpoint. One of the popular tools for doing that in data science projects is Gradio.

As I have mentioned before in this report, I did not see a point in implementing any design patterns in such a small project, but as the project's structure and complexity would be increased it would be a good idea to think about implementing a few patterns.