AI Engineer Take-Home Assignment: Natural Language to SQL with Llama 3 70B

Content :

1. Approach

Objective
The primary objective was to build a platform that takes natural language queries from a user, translates them into SQL queries using the Llama3 70 LLM, and returns the results from the database.

Architecture
The system is divided into three main components:

LLM Block: This component, implemented using TGI* (Text Generation Interface) does the following :
- Deploy the model as an endpoint on AWS
- Takes NLP input and, with an elaborate prompt, asks the model to generate the corresponding SQL code
- Selects only the SQL code from the output
- Sends the code as a POST request to the next component (Flask API)

Prompt engineering here had an important role to play regarding the pipeline's good functioning. This skill is indeed needed to converse effectively with the models and are instructions given to them to enforce rules, automate processes and ensure specific quality of generated output.
The more context and precision are given to the LLM, the better the output.

In the end, the prompt was created based on several references (from 1 to 6). I thought that giving the database architecture to the model would have a positive impact on its answers. In previous experiments, adding "\n" to properly separate the context window's different components also gave better results.

```
prompt = f"""
    You are a helpful assistant specializing in data analysis in a {type} database. \n\n
    Answer the question by providing SQL code compatible with the {type} environment. \n\n

    Question: \n\n
    {query}
    \n\n
    ### Database Schema \n
    This query will run on a database whose schema is represented as follows: \n\n
    {schema}
    \n\n
    ### SQL \n
    Given the database schema, here is the SQL query that answers the question:\n\n

    """
```

Example :

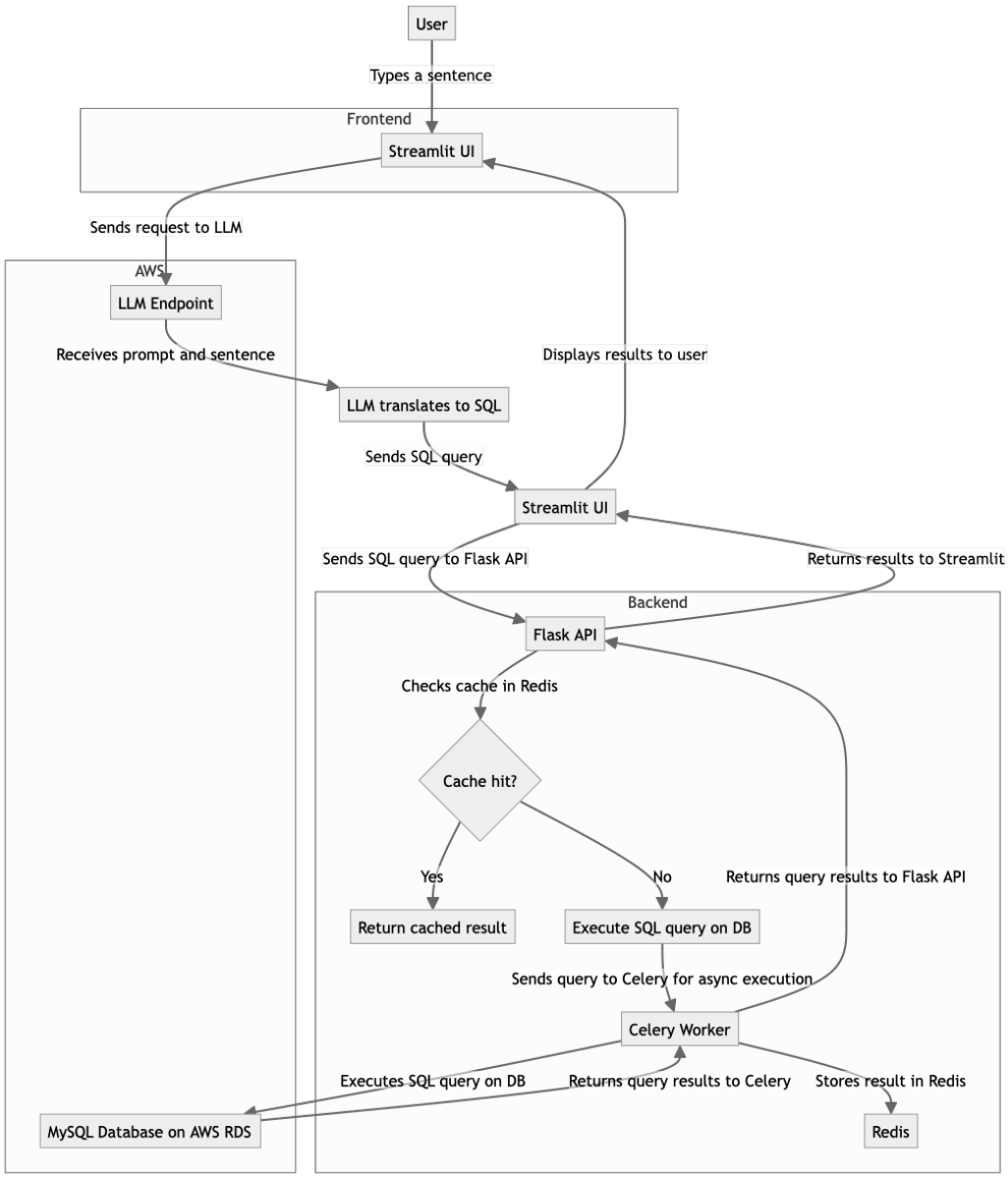
```
prompt = f"""
    You are a helpful assistant specializing in data analysis in a MySQL database. \n\n
    Answer the question by providing SQL code compatible with the MySQL environment. \n\n
    Question: \n\n
    Give me the number of laptops that Aaron placed.
    \n\n
    ### Database Schema \n
    This query will run on a database whose schema is represented as follows: \n\n
    {{
        "platform": {{
            "orders": {{
                "orderID": "INTEGER",
                "userName": "VARCHAR(50)",
                "orderType": "VARCHAR(255)",
                "purchaseDate": "DATE"
            }},
            "products": {{
                "productID": "INTEGER",
                "productType": "VARCHAR(50)",
                "operatingSystem": "VARCHAR(50)"
            }}
        }}
    }}
    \n\n
    ### SQL \n
    Given the database schema, here is the SQL query that answers the question:\n\n

    """
```

* TGI a toolkit from HuggingFace for deploying and serving LLMs. TGI enables high-performance text generation for the most popular open-source LLMs, including Llama, Falcon, StarCoder, etc.

Flask API Block: This component serves an an intermediate layer to receive the SQL code from the LLM and interacts with the database to execute the query.

Streamlit Block: Streamlit provides a simple and fast-to-implement user interface with the system, allowing users to select their Schema in the database, input queries and view the results.

Overall architecture



Key design decisions

The system was design into blocks to have a clear separation of concernes. The LLM, API, and UI components were implemented as separate services to facilitate easy maintenance and scalability. The front-end communicate with the LLM and Database via APIs.

2. Demo

A short video showing a live demo of the software is attached to this report/ email.

3. Evaluation Method and Results

In order to evaluate the translated query to SQL, two main methods were implemented :

- ```validate_sql_syntax``` is designed to validate the syntax of a given SQL code snippet using SQLAlchemy Engine. The SQL code is prefixed with EXPLAIN, which is a SQL command used to obtain a description of the query execution plan without actually executing the query. This is a common technique to validate SQL syntax without affecting the database.
- ```check_schema_compliance``` is designed to verify if a given SQL code snippet complies with the schema of the database it is intended to be executed against.

Further improvements can be made to guarantee the query's correctness, as well as better performance (response time for example)

4. Discussion

Challenges

LLM-related

First of all, the Llama3 70B model as it is is a big LLM, very hard to deploy at the Edge.
Two approaches can be explored from there.
A Cloud solution to host it (on a AWS endpoint for example) seemed to be the fastest, most practical solution especially for a production environment.
However, techniques like quantization, model compression/distillation (transfering knowledge from a large, expensive model to a smaller one) can be applied in order to reduce the size of the model while preserving its capabilities.
Alternative models can be chosen to run locally, like the latests Llama3.1 8B.
Also, the Llama3 70B model being outdated, the new 3.1 70B can be used here.

Another challenge was to find the right prompt to call the model. The prompt had to be designed in such way that the models would understand perfectly the query: this involved telling the model about the chosen database's architecture (Schema, Tables, etc), giving it an elaborate context window. Techniques to find the appropriate prompts have been investigated intensively the past few years. Also, I got inspired here by multiple sources, especially the 'Evol-Instruct' technique used to generate synthetic data.

Scalability-related
At the moment, everything works locally and is not properly configured to scale but a first step could be to wrap everything in Docker in order to make it available on other machines. As the number of users grow, several endpoint can be created to balance the traffic.

Security-related
At the moment, no security-related component was added to the system, but a dynamic login component, both for user identification and database connection should be added to secure the platform at its minimum.

Ideas to go further
The project being limited by time constraints, further improvement can be made to this first draft.
- Adding Dockerfile/docker-compose components to better handle the connections between database, UI, and LLM, as well as allowing reproducibility, load management and scalability.
- To further enhance to lastly quoted benefits, Kubernetes can provide advanced scaling and load balancing capabilities. It can manage complex, large-scale applications, handling more users.
- Using Redis to cache the queries that have been already asked, reducing the latency and preventing from calling the database unnecessarily.
- User better error-tracking/handling techniques :
- Using an edited version of CodeT (Microsoft) to generate unit tests associated with the SQL code produced by the LLM to check if the code would run or not.
- Use Celery to manage asynchronous tasks in the database (like long-running SQL queries)
- CI/CD for continuous automation and testing of the platform

References :

1. Prompt Engineering for a Better SQL Code Generation With LLMs (https://medium.com/datamindedbe/prompt-engineering-for-a-better-sql-code-generation-with-llms-263562c0c35d)
2. Guide to prompt engineering: Translating natural language to SQL with Llama 2 https://blogs.oracle.com/ai-and-datascience/post/prompt-engineering-natural-language-sql-llama2
3. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT (https://arxiv.org/pdf/2302.11382)
4. WizardLM: Empowering Large Language Models to Follow Complex Instructions (https://arxiv.org/abs/2304.12244)
5. WizardCoder: Empowering Code Large Language Models with Evol-Instruct (https://arxiv.org/abs/2306.08568)
6. Evol Dataset, theblackcat102 (https://github.com/theblackcat102/evol-dataset/tree/main/evolinstruct/instructions)