ChatToMySQL

Ke Wu

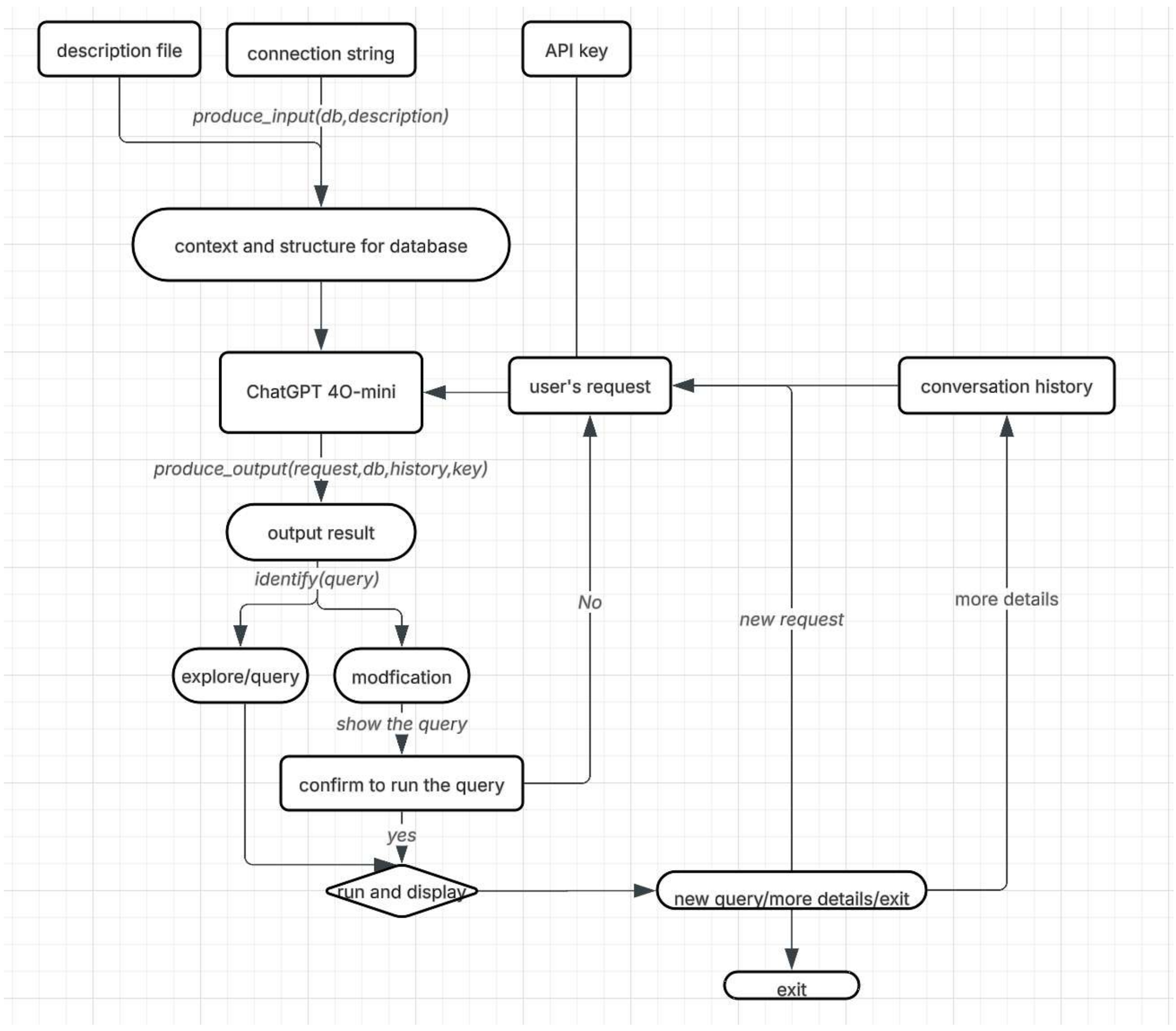
- **Introduction**

The rise of Large Language Models (LLMs) presents the possibility for humans to interact with databases using natural language. This would be convenient for individuals unfamiliar with SQL and could save programmers time when dealing with complex SQL logic. This project aims to build a Natural Language Interface (NLI) on top of an LLM, enabling users to explore, query, and modify data within MySQL database.

- **Planned Implementation**

Use OpenAI as a natural language model to achieve the goal, or compare it with other LLMs, and choose the one with best performance. Coding work will be roughly divided into NLP model, connection on and execution on MySQL database, interface construction three parts.

- **Architecture Design (Flow Diagram and its description)**



created by lucidChart

To initiate the interface, users provide data description files, a database connection string, and an OpenAI API key. Upon connecting to the user-provided database, a produce_input function iterates over all tables to retrieve the database schema. This schema, combined with the data description files, forms part of the system input for calls to the OpenAI API. Subsequently, using the API key and the user's natural language request, a response is generated via the GPT-4o Mini model.

A simple extraction function then filters the SQL from this response, and this SQL is automatically classified as either a query (for data exploration or retrieval) or a modification. If classified as a query, the system executes it and displays the results using PyMySQL. If it's a modification, an additional confirmation step is required before execution, as database modifications warrant caution.

Following each request execution, the program prompts the user with three options: 'New Query,' 'More Details,' or 'Exit.' 'New Query' allows for an entirely new request to the database, while 'Exit' terminates the program. The 'More Details' option enables the user to refine the current query if the results are not entirely satisfactory. For instance, if the user wants to sort the previous query's results, they can input 'sort by last name.' To ensure the LLM API can process such conversational follow-ups, the system includes the ongoing conversation history as part of the input for each API call. This process outlines the basic workflow of the program.

- **Implementation**
    1. Functionalities

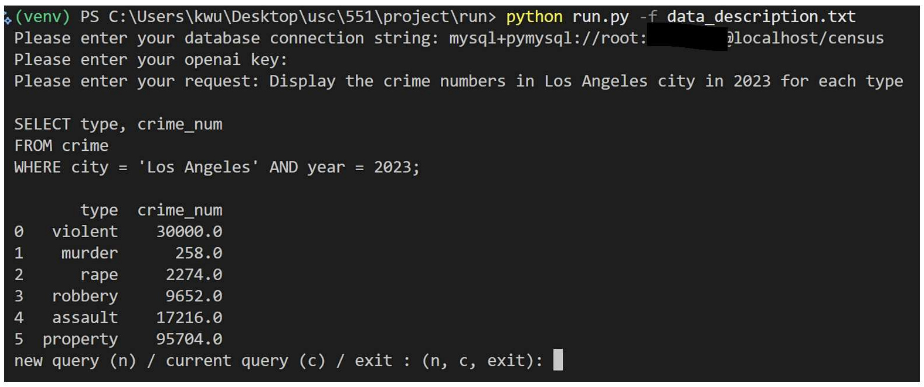
       Fundamentally, ChatToMySQL enables users to explore, query, and modify (INSERT, DELETE and UPDATE) MySQL databases. Specifically, it supports a comprehensive range of SQL operations including SELECT, FILTER (e.g., WHERE clauses), JOIN, aggregation functions (with HAVING), set operations, subqueries, OFFSET, ORDER BY, LIMIT, and GROUP BY. Furthermore, the system allows users to build upon previous interactions by referencing conversational history in their requests.

    2. Tech Stack

       This program is developed in a **Python** environment. It utilizes the **openai** package for making API calls to GPT-4o Mini. Connectivity with MySQL databases and data handling are managed using the **pymysql, pandas, and sqlalchemy** packages. Additionally, the **click** package was employed to build the command-line interface (CLI) for the application.

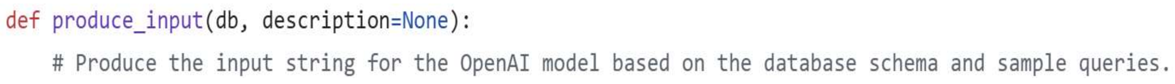
3. Implementation Screenshots

Here's a simple example of ChatToMySQL program:

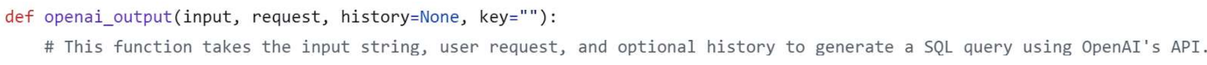


```
(venv) PS C:\Users\kwu\Desktop\usc\551\project\run> python run.py -f data_description.txt
Please enter your database connection string: mysql+pymysql://root:[redacted]@localhost/census
Please enter your openai key:
Please enter your request: Display the crime numbers in Los Angeles city in 2023 for each type

SELECT type, crime_num
FROM crime
WHERE city = 'Los Angeles' AND year = 2023;

       type  crime_num
0   violent    30000.0
1    murder      258.0
2      rape     2274.0
3   robbery     9652.0
4   assault    17216.0
5  property    95704.0
new query (n) / current query (c) / exit : (n, c, exit):
```

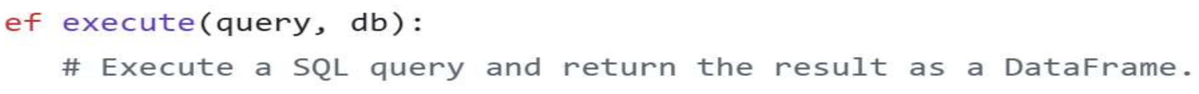
Here are all functions utilized in the program:



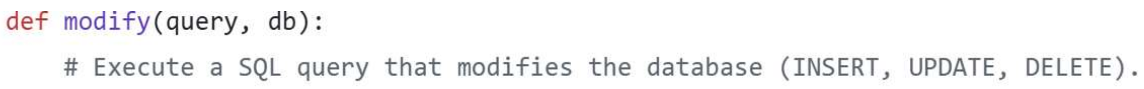
```
def produce_input(db, description=None):
    # Produce the input string for the OpenAI model based on the database schema and sample queries.
```



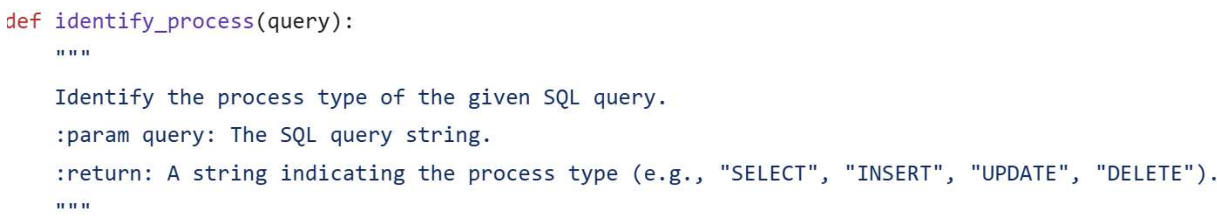
```
def openai_output(input, request, history=None, key=""):
    # This function takes the input string, user request, and optional history to generate a SQL query using OpenAI's API.
```



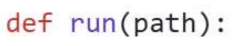
```
ef execute(query, db):
    # Execute a SQL query and return the result as a DataFrame.
```



```
def modify(query, db):
    # Execute a SQL query that modifies the database (INSERT, UPDATE, DELETE).
```



```
def identify_process(query):
    """
    Identify the process type of the given SQL query.
    :param query: The SQL query string.
    :return: A string indicating the process type (e.g., "SELECT", "INSERT", "UPDATE", "DELETE").
    """
```



```
def run(path):
```

- **Learning Outcomes**

  Through this project, I primarily learned to leverage Large Language Model (LLM) APIs for specific domain applications and to establish Python connections with MySQL databases. Developing tests for the program further deepened my familiarity with MySQL. Additionally, I gained experience in creating command-line interfaces (CLIs).

- **Challenges Faced**

  A primary challenge revolved around improving accuracy. Initially, providing only the database schema as input to the LLM yielded unsatisfactory results, although it performed adequately with simple models. However, when the relationships between tables became more complex, the accuracy dropped below 0.3. After further research, I incorporated context by adding data description files to the model's input. This, however, often required multiple attempts to produce description files that the model could clearly understand. For instance, an initial description for a 'year' column in the database stated, 'this table is time-series and ranging from 2020 to 2023.' Consequently, when I attempted to request an insertion of a 2024 record, the model responded that 2024 was out of range for this table. Subsequently, I had to revise the description files to prevent such misunderstandings between the user and the agent.

Additionally, after addressing accuracy, another challenge encountered was managing conversational context. Specifically, when interacting with the OpenAI API, each call is typically stateless and does not inherently retain memory of previous interactions. This means the LLM cannot easily answer user requests that depend on the preceding conversation. For example, if a user wishes to refine a query with a follow-up like, 'sort by city name based on the last query result,' the LLM, lacking this conversational history, would likely not return the expected answer. To address this, the solution involved sending the accumulated conversation history back to the model with each new request, ensuring it could understand and respond within the established context.

- **Conclusion**

In conclusion, ChatToMySQL demonstrates the capability to empower users to interact with MySQL databases using natural language. Its core functionalities include schema and data exploration, retrieving results from natural language queries, and executing natural language commands for data modification. Performance testing indicates a first-attempt accuracy of 85% (17/20) in converting natural language to the correct SQL query. For queries not successfully generated on the first try, the system typically achieves the desired outcome within a maximum of three attempts.

- **Future Scope**

Future work will focus on two parts: reducing the cost and optimizing the user interface.

While incorporating data description files significantly improved accuracy, it also increased token consumption. Similarly, maintaining conversational context by repeatedly sending the entire interaction history back to the model can become token-intensive, especially during extended user sessions. Another area for enhancement is the user interface. The current command-line interface (CLI) presents limitations, particularly for features like easily reviewing past requests. Therefore, transitioning to a web-based interface or a Graphical User Interface (GUI) would offer a more intuitive user experience and facilitate a broader range of functionalities.

Reference:

https://community.openai.com/t/how-do-you-maintain-historical-context-in-repeat-api-calls/34395/2

https://realpython.com/python-click/

https://github.com/vanna-ai/vanna/blob/main/papers/ai-sql-accuracy-2023-08-17.md#ai-sql-accuracy-testing-different-llms--context-strategies-to-maximize-sql-generation-accuracy