LLM-DB Midterm Progress Report

1. Team Details

Team name: LLM-DB
Team members:

- Member Sean Li
  - Background: Data science student, familiar with Python, SQL and database design, has participated in multiple web development and GUI development projects, understands the basic concepts of AI and natural language processing, and can use basic artificial intelligence algorithms to solve problems.
  - Skills: MySQL database management, SQL optimization, Python backend development, basic NLP model call.

- Member ShiXin Fang
  - Background: Data science student, good at NoSQL-based database design, experience in building services using Axure and Google Cloud, familiar with LLM's technical architecture, familiar with RESTful-based API design.
  - Skills: MongoDB aggregation pipeline design, Python programming, system testing, API performance tuning.

Project Division:

Member	Responsibilities
Sean Li	RDBMS module responsible person: ● MySQL database design and query generation logic ● SQL syntax verification and performance optimization ● Integrate Hugging Face model to parse natural language
ShiXin Fang	NoSQL module leader: ● MongoDB collection design and aggregation pipeline generation ● Error handling and API result formatting ● OpenAI API calls and cost management
Common tasks	System joint debugging: ● API interface design and joint debugging test ● Cross-database function verification and document writing

2. Implementation Question

- Teck Stack Used:
  Database: Mysql, Mongodb, pymysql
  Web UI: FastAPI, JQuery, Ajax, Javascript

LLM: OpenAI / DeepSeek

- Query Syntax Implementation Plan:

The core of this program is to use prompt to fine-tune LLM, set certain parameters, input natural language into LLM, and obtain LLM output. The SQL query and execution statement output by LLM are passed to the backend through the UI interface, and the corresponding database is connected to query.

- Database Selection:

First, LLM will be clearly informed of the databases available for selection, and the functions, types, and existing fields of each database. After receiving the user's input, it will determine whether the user's input semantics are related to the existing database content. If so, the corresponding database will be selected for execution. If not, an exception will be notified, and the user will be told which existing databases LLM can handle.

3. Planned Implementation

- System architecture:
  - ➢ Front-end: Simple command line interface (CLI) + API testing tools (such as postman). In order to ensure that the functions can be implemented normally, Web UI/GUI is not designed for the time being.
  - ➢ Back-end: Python + FastAPI, providing RESTful API to receive user input and distribute it to different database modules for processing.

- Natural language parser:
  - ➢ Use the fine-tuning model of open source LLM to convert user input into a feasible preliminary query structure, while optimizing the format of prompts to ensure the reliability of LLM output.

- Execution engine:
  - ➢ RDBMS module: After receiving the input from the natural language parser, execute the corresponding SQL statement, dynamically process table connections, automatically identify foreign keys, and perform conditional logic nesting.
  - ➢ NoSQL module: After receiving the input from the natural language parser, parse it into a MongoDB aggregation pipeline.

- Error handling:
  - ➢ Return the results in JSON format uniformly, including status code, data, and error information (such as "cannot find table students").

- Sample database design:
  - ➢ MySQL:

Students: id, name, age, student_id

Faculty: id, name, position

- ➢ MongoDB:

Course: {course_id, teacher_id, quantity}

Teachers: {name, major, course}

4. Project Status

A. Database design

I. We first created a student_db database based on MySQL, which stores some simulated student information. The Nosql part is temporarily shelved. The design of the entire table is as follows:

II. Students:id,student_id, name,age (student table, records student name, age and student id, college id)

III. Faculty:id, faculty_id, name, subject (college table, records the name of the college and the subjects of the college)

IV. Grade: id, student_id, subject, score (score table, records student id, subject, and the student's score in the subject)

V. We use the CLI interface to operate the data through the built maridadb service. The overview of the data is as follows:

VI. **Students table**

id	student_id	name	age	faculty_id
1	S0001	George Logan	19	PH
2	S0002	Jennifer Reeves	24	PH
3	S0003	Richard Garcia	22	CH
......				

VII. **Faculty Table**

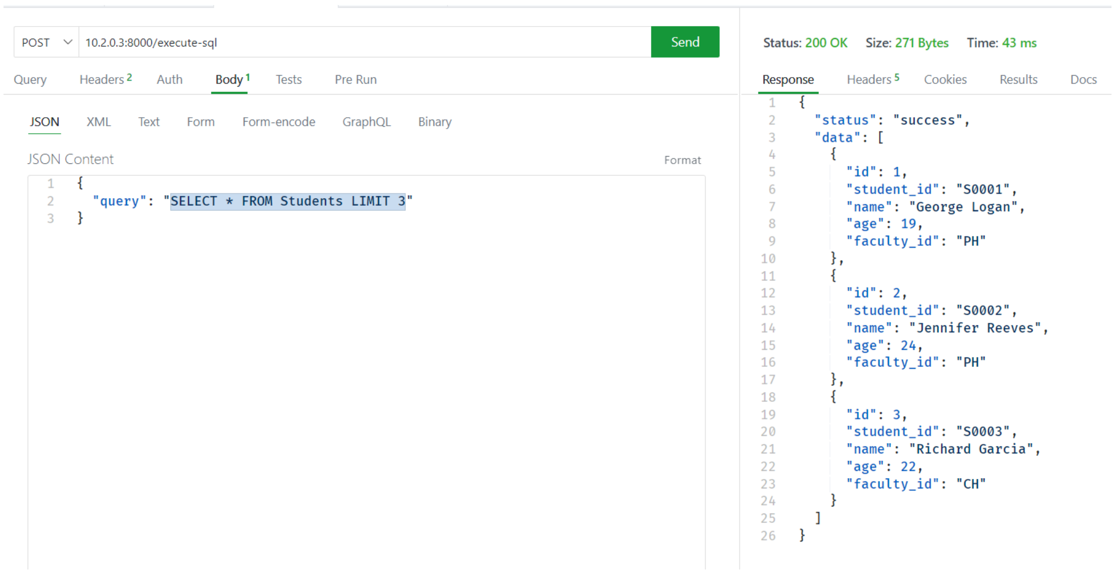
id	faculty_id	name	subject
1	CS	Computer	Data Structure
2	CS	Computer	Algorithm
3	CS	Computer	Database
......			

VIII. Grade Table

id	student_id	subject	score
1	S0001	Data Structure	80
2	S0001	Electromagnetism	64
3	S0001	Mechanics	77
......			

B. Use Fastapi to implement remote API calls

FastAPI is a modern, fast (high-performance) web framework for building APIs, designed for building RESTful APIs in Python. The web middleware code built with Fastapi is very concise, and its core function is to accept the user's POST request, which includes natural language and SQL statements. Natural language is the function we want to implement, and the input of SQL statements is mainly for testing. Currently, we have not yet implemented the processing operation after natural language acceptance, because we need to ensure that the SQL statement is in a feasible state. After using Fastapi to build the basic framework of the web middleware, use the api testing tool to test the acceptance status of the web server. Here, we use the Thunder Client client. Thunder Client is a VSCode plug-in that can perform RESTful functional testing. A typical test case screenshot is as follows:



```
{
  "query": "SELECT * FROM Students LIMIT 3"
}
```

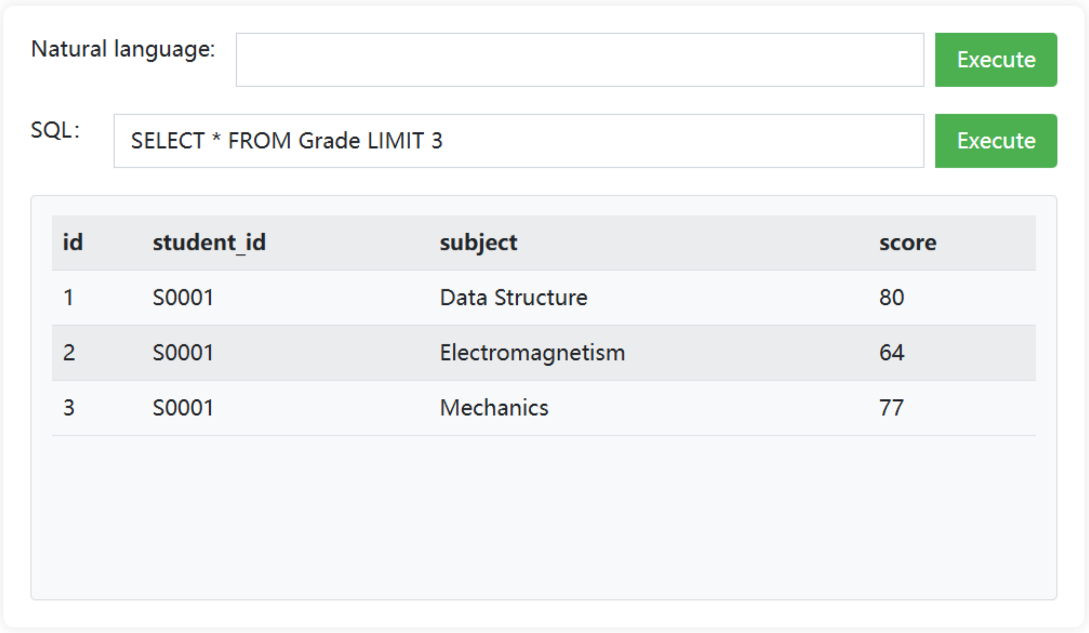
```
{
  "status": "success",
  "data": [
    {
      "id": 1,
      "student_id": "S0001",
      "name": "George Logan",
      "age": 19,
      "faculty_id": "PH"
    },
    {
      "id": 2,
      "student_id": "S0002",
      "name": "Jennifer Reeves",
      "age": 24,
      "faculty_id": "PH"
    },
    {
      "id": 3,
      "student_id": "S0003",
      "name": "Richard Garcia",
      "age": 22,
      "faculty_id": "CH"
    }
  ]
}
```

As you can see, after executing the correct POST request operation, the server returns the data content stored in Mysql. This means that the web server middleware built by Fastapi we use is running successfully.

C. Visual interface design

In order to ensure a good user experience, we conducted a visual design based on the jquery+ajax technology stack to send data to the server, and used css+html for certain beautification. In the visual design part, the overall core part lies in three parts:

- i. The page layout is centered. The overall layout is:
  - text-input-button
  - text2-input2-button
  - display area
- ii. The first text tag, the content is "Natural language:", there is an input bar on the right, and there is a button on the right of the input, and the button content is "Execute".
- iii. The second text tag, the content is "SQL:", there is an input bar on the right, and there is a button on the right of the input, and the button content is "Execute".
- iv. When the user clicks the button of the first label, no operation is needed for the time being, and we will continue to implement its functions later; when the user clicks the button of the second label, the SQL statement in the input column will be sent to the backend server, and after execution, the result will be displayed in the display area. According to the above assumptions, the web interface we built is as follows:



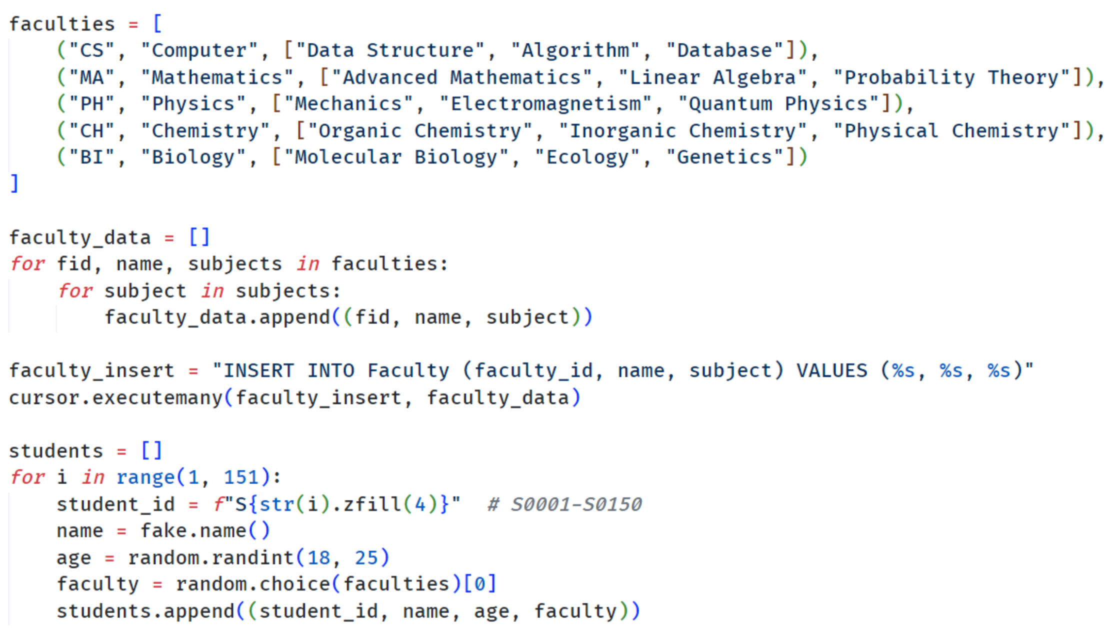
```
SELECT * FROM Grade LIMIT 3
```

id	student_id	subject	score
1	S0001	Data Structure	80
2	S0001	Electromagnetism	64
3	S0001	Mechanics	77

We entered some SQL statements in the SQL input column to observe the test server response and saw that the server was able to return the data text we needed normally, which proved that this part of the project had been successfully completed.

5. Challenges Faced

The current project is progressing smoothly, and many necessary basic work has been completed, but we still encountered many challenges during the implementation process. For example:
i. Database design
In order to ensure that the data is sufficiently suitable for the project requirements, we use the method of simulating data instead of directly downloading certain databases from the Internet. In the process of simulating and designing data, we need to ensure the relevance and validity of the data, which is very important for relational databases, because this project needs to ensure that various SQL operations can be successfully executed after being converted using natural language. Therefore, a reasonable database design can ensure the test coverage of subsequent projects. At the same time, the data should be as much as possible, otherwise the purpose of comprehensive testing cannot be achieved. For this, we use the Faker library of python to create 150 random data to ensure that the data is non-repetitive and representative. A sample code is as follows:



```python
faculties = [
    ("CS", "Computer", ["Data Structure", "Algorithm", "Database"]),
    ("MA", "Mathematics", ["Advanced Mathematics", "Linear Algebra", "Probability Theory"]),
    ("PH", "Physics", ["Mechanics", "Electromagnetism", "Quantum Physics"]),
    ("CH", "Chemistry", ["Organic Chemistry", "Inorganic Chemistry", "Physical Chemistry"]),
    ("BI", "Biology", ["Molecular Biology", "Ecology", "Genetics"])
]

faculty_data = []
for fid, name, subjects in faculties:
    for subject in subjects:
        faculty_data.append((fid, name, subject))

faculty_insert = "INSERT INTO Faculty (faculty_id, name, subject) VALUES (%s, %s, %s)"
cursor.executemany(faculty_insert, faculty_data)

students = []
for i in range(1, 151):
    student_id = f"S{str(i).zfill(4)}"   # S0001-S0150
    name = fake.name()
    age = random.randint(18, 25)
    faculty = random.choice(faculties)[0]
    students.append((student_id, name, age, faculty))
```

ii. Natural language parser

The natural language parser is the core work of this project, and we are still in the debugging stage. The main difficulty we face is that we cannot ensure the stability of LLM output for the time being. For example, after entering a specific prompt, LLM will have some irrelevant output instead of pure SQL statements, which affects the correct execution of the program. The plan for subsequent projects is to continue to tune the effect of the prompt, control the change of the temperature value, and ensure that the output of the program is fixed. On the other hand, we also need to tell LLM which database is currently being operated, because according to the original design, the ideal situation of the project is that LLM can autonomously identify which database the user wants to operate and automatically generate SQL input. But this plan may be adjusted later.

6. Timeline

Stage	Deadline	Deliverables	Status
Database design	2/10 - 2/15	Completed 2 sample databases (MySQL + MongoDB), including connectable field relationships	Completed
Natural language parsing prototype	2/16 - 2/28	Implemented basic query conversion (such as single table SELECT/FIND), Hugging Face API integration test	Ongoing
Complex query support development	3/1 - 3/15	Supported JOIN/lookup, aggregation operations (GROUPBY/lookup, aggregation operations (GROUPBY/group)	Ongoing
Data operation function development	3/15 - 3/31	Implemented INSERT/UPDATE/DELETE statement generation and execution	Ongoing

Stage	Deadline	Deliverables	Status
System integration and performance optimization	4/1 - 4/15	Completed end-to-end testing, optimized response time to within 2 seconds, error rate <5%, completed demo	To be started
Final testing and documentation submission	4/16 - 5/5	Uploaded code to Google Drive, wrote final report and API documentation	To be started

7. The Next Plan

i. Design for the NoSQL (Mongodb) database and simulate test data that meets the needs of the project. The data should be as complete as possible.
ii. Adjust the natural language processor to ensure that its output is stable enough, and then set it up on our project.
iii. Test the capabilities of the natural language processor to observe whether it can operate two types of databases simultaneously, and make certain adjustments based on the performance of the program.
iv. Further in-depth testing, design some complex test operations to observe the performance of the program.