engineering proceedings

MDPI

Proceeding Paper

Development and Evaluation of Learning Portfolio Query System Based on LangChain Framework †

Nien-Lin Hsueh * and Wei-Ting Wang

Department of Information Engineering and Computer Science, Feng Chia University, Taichung 407, Taiwan; m1107421@o365.fcu.edu.tw

* Correspondence: nlhsueh@fcu.edu.tw

† Presented at the 2024 IEEE 6th Eurasia Conference on IoT, Communication and Engineering, Yunlin, Taiwan, 15–17 November 2024.

Abstract: With the increasing popularity of online education platforms, the use frequency of students and teachers has gradually increased. A large volume of data is generated and analyzed daily on these platforms including course information and student learning status. However, traditional analysis methods often require substantial manpower and expertise. Large language models Chat GPT-4 offer a potential solution to this problem. This study aims to address this challenge by utilizing the large-scale language model framework LangChain and the database of the OpenEdu online education platform. We designed an interface capable of querying educational data in natural language. When a user queries in natural language, the large language model generates structured query language to query the database and converts the query back into natural language to respond to the user's question. Two different query methods were developed based on LangChain components: the sequential query version and the Agent query version. Based on these methods, four different versions of prompt and model combinations were created. The accuracy in converting natural language to SQL was estimated, and error type analysis was conducted to enhance the system's performance and accuracy. The execution accuracy reached up to 85.7%, with the primary error type in natural language-generated SQL being Schema Linking. By integrating large-scale language models into conversational query systems, a promising approach was developed for handling large-scale data queries on educational platforms.

Keywords: large language models; LangChain; natural language processing (NLP)

check for updates

Academic Editors: Teen-Hang Meen, Chi-Ting Ho and Cheng-Fu Yang

Published: 30 April 2025

Citation: Hsueh, N.-L.; Wang, W.-T. Development and Evaluation of Learning Portfolio Query System Based on LangChain Framework. *Eng. Proc.* **2025**, *92*, 40. https://doi.org/10.3390/engproc2025092040

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

With the advancement of artificial intelligence (AI), ChatGPT has been a significant breakthrough. ChatGPT uses natural language processing (NLP) technology to interact with humans. AI is used in various foundational technologies, such as machine learning and deep learning. Large language models (LLMs) are based on deep learning and handle natural language tasks. These models consist of billions to trillions of parameters, learning the structure and semantics of language through extensive data pre-training to predict the next word or sentence. LLMs are a part of NLP technology. In the evolution of language models, the range of tasks that LLMs can solve has expanded, and their performance in handling tasks has significantly improved [1].

With the development of LLM applications across various fields, a framework named LangChain is used to support LLM applications. LangChain provides an interface that connects different LLMs and external data sources, making it a convenient development

tool. LangChain consists of multiple composable components (LangChain components), including model I/O, retrieval, chains, agents, memory, and callbacks. The modular design of these components facilitates the development and construction of more applications based on LLMs [2]. For example, Pandya and Holia demonstrated how to automate customer service using the LangChain framework and LLMs [3], and Jacob et al. used the LangChain framework to build a ChatGPT for PDF documents, showcasing LangChain combined with LLMs to achieve efficient PDF querying and information extraction [4].

LangChain's components are as follows.

- Model I/O: An interface for LLMs, including prompt optimization and management.
- Retrieval: A component for retrieving and interacting with external data.
- Chains: Provides chains in LangChain to accomplish specific tasks, allowing customization based on application needs.
- Agents: Agents think and reason the actions needed based on user questions, guiding the LLM through a series of actions.
- Memory: Maintains application state between chain runs, especially useful in conversational QA.
- Callbacks: LangChain offers a callback system that allows developers to better understand and monitor the application's operations.

In this study, we applied LLMs to the OpenEdu platform, a Chinese open-education platform. The platform was designed to provide online learning resources for students. Since a large amount of classroom and student data is generated daily on this platform, a fast and efficient query and analysis system is required. However, traditional query and analysis methods require substantial manpower and expertise. Therefore, we designed a natural language query system to allow teachers or developers to obtain course and student performance data and improve the efficiency of course and student management. The system helps teachers effectively analyze course and student performance data to improve teaching strategies.

Using LangChain to connect to the OpenEdu database, we generated SQL queries through LLMs and converted the query results into natural language to describe the data. Finally, we evaluated the accuracy of the generated SQL. We evaluated the execution accuracy of LangChain and GPT query methods in converting natural language to SQL and analyzed the types of errors in SQL generation by LangChain and GPT query methods.

2. Literature Review

2.1. Natural Language in Data

LLMs such as generative pre-trained transformer (GPT) and large language model meta AI (LLAMA) have demonstrated strong capabilities in various natural language processing techniques and provided effective methods for data-related tasks. For example, Cheng et al. used the GPT-4 model for data analysis and compared it with professional data analysts to evaluate whether LLMs could replace data analysts [5]. Zhang et al. proposed Data-copilot based on LLMs, which autonomously manage and analyze data [6]. These studies show the feasibility of using LLMs for data analysis. In education, Yu et al. integrated past MOOC educational data to create a large database called MOOCCube, providing rich resources with natural language processing educational data and promoting research in natural language processing and AI in education [7].

2.2. Natural Language to Structured Query Language (SQL) Conversion

The technique of text-to-SQL is used to translate natural language into SQL queries to retrieve user questions. LLMs trained on large datasets possess strong semantic under-

standing capabilities, making them promising. However, determining whether the SQL matches the user's question is a challenge.

In the study of the execution accuracy of natural language to SQL tasks, Yu and Zhang introduced a large-scale, complex, cross-domain semantic parsing, and text-to-SQL dataset annotated by 11 college students (Spider) [8]. This dataset includes natural language questions from different domains corresponding to SQL queries to develop and test new text-to-SQL methods. Gao and Wang proposed a new text-to- SQL prompt method called DAIL SQL, which achieved 86.6% accuracy, a record for the Spider benchmark [9]. Rajkumar et al. designed various prompt design methods and found that the prompt with CREATE TABLE commands and three pieces of data from the table achieved the highest accuracy of 67.0%, laying the foundation for subsequent prompt design research [10]. Meimarakis and Koutrika highlighted challenges in text-to-SQL conversion, such as the same sentence in natural language potentially having different meanings, or two different sentences having the same meaning, both translating to the same SQL query. User errors, such as typos and syntactical or grammatical errors, complicate the translation task [11].

The SQL syntax is strict, unlike natural language, which has more flexibility. In natural language, sentences with errors can still be understood, but SQL syntax errors result in execution failure as errors cause differences between database column names and user terms.

2.3. Evaluation Metrics

Natural language querying SQL is evaluated using various metrics. Execution accuracy and string matching were used for evaluation. Execution accuracy is calculated based on the results returned by the queries. String matching is the simplest accuracy metric for text-to-SQL. The generated SQL query and the reference SQL query as strings are compared without considering order [12]. Execution accuracy, component matching, and exact matching were used to evaluate the performance of text-to-SQL tasks, too. Component matching is used to better understand which parts of the SQL query are correctly predicted and grouped by different SQL components such as select, where group by, order by, and keywords. Exact matching considers all SQL components and their order [8].

In the literature review, we explored the application of LLMs in natural language processing, particularly in the natural language to SQL conversion technology for data analysis. LLMs enable strong semantic understanding and generation capabilities, effectively converting natural language questions into SQL queries. We explored various prompt designs and methods, such as the CREATE TABLE command and DAIL-SQL which are used to improve the model's execution accuracy. These studies provide theoretical and practical guidance for natural language to SQL conversion while pointing out challenges and directions. We summarized the evaluation metrics for natural language-generated SQL used in previous studies as shown in Table 1.

Table 1. Summary of previous studies on evaluation methods for natural language to SQL conversion.

Name	Description
String Matching [12]	Comparing the generated SQL query and the reference SQL query as strings without considering order.
Execution Accuracy [8,12]	Judging accuracy based on the results returned by the queries.
Component Matching [8]	Grouping by different SQL components to see if they match correctly.
Exact Matching [8]	Comparing all SQL components and considering their order.

3. Methods

3.1. System Design

A system was developed based on the Django web application framework combined with LangChain. In using the LangChain, different component combinations were utilized to develop different versions of operational modes, sequential query, and agent query. The LLM interface interfaces with the OpenAI GPT-4 API and improves query accuracy. Chains are used to combine prompts and LLMs to accomplish tasks, while agents plan how to use tools and prompt the LLM for subsequent actions based on the questions. The memory function was implemented through LangChain's Conversation-BufferMemory to store dialogue messages in a buffer and pass the messages to the prompt template. This allowed the agent to maintain contextual continuity when handling tasks, thereby enhancing the overall system performance and accuracy (Figure 1).

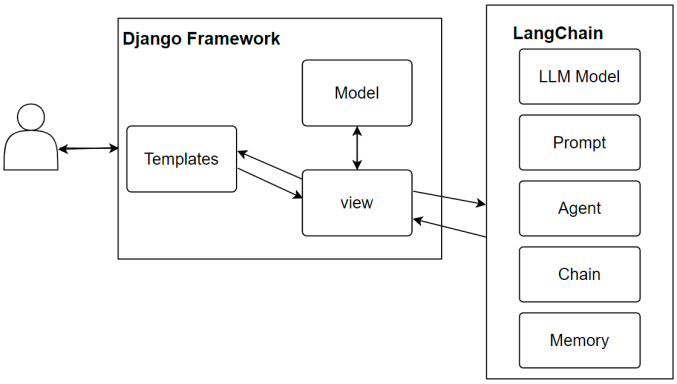


Figure 1. System architecture (D).

After proposing hypotheses, different LangChain components were used to design query systems with different operational principles. Then, the questions that the system needs to query were designed and tested. Subsequently, the accuracy of LangChain and GPT query methods in generating SQL was calculated. Chi-square was used to verify the experimental hypotheses, followed by error-type analysis.

3.2. Prompt Design

The prompts in this study were divided into three categories: information on database tables, natural language descriptions of tables and columns, and SQL query examples. In LangChain, the SQL database class was constructed using table information prompts and the create table command. The first three rows of data from the table were used in the database. To enhance the LLM's recognition and understanding of tables and columns, we created a document containing descriptions of the database tables and their columns and added it as a prompt. To increase the accuracy of SQL queries, we used the few-shot prompting method to add prompts. A few-shot prompt is a small set of input=output examples that enables the model to learn. We created an example set, where each example included a user's input question and the corresponding SQL syntax as shown in Figure 2. By providing these examples, the LLM's accuracy in generating SQL syntax was improved.

```
examples = [
    {"input": "請告訴我資訊安全導論，這門課的註冊人數有多少人?",
     "query": 'SELECT * FROM dashboard.course_overview_course where course_name="資訊
    安全導論";',
     },
    {"input": "生活中的經濟思維與思考創新這門課的註冊人數有多少",
     "query": 'SELECT * FROM dashboard.course_total_data_latest where course_name="生
    活中的經濟思維與思考創新";',
     },
    {"input": "在2021年的11月註冊人數總共有多少",
     "query": 'SELECT count FROM dashboard.register_count where year="2021" and month
    ="11";',
     },
    {"input": "告訴我軟體設計的課程ID和分別的註冊人數",
     "query": 'SELECT course_id,註冊人數 FROM dashboard.course_overview_course WHERE
    course_name="軟體設計";',
     },
    {"input": "請查詢有修課程ID為'FCUx/mooc0002/201410'的學生，並列出課程ID、課程名
    稱、user id、學生姓名",
     "query": "SELECT c.course_id, c.course_name, sc.user_id, sc.name FROM dashboard.
    course_total_data_latest AS c INNER JOIN dashboard.student_course AS sc ON c.
    course_id = sc.course_id where c.course_id = 'FCUx/mooc0002/201410';",
     },
    {"input": "請查詢課程名稱為 '空間資訊與智慧生活' 課程ID為'FCUx/mooc0003/201410'的
    註冊人數及註冊該課程的學生ID，",
     "query": "SELECT c.course_id, c.course_name,c.註冊人數, s.user_id FROM dashboard.
    course_total_data_latest AS c INNER JOIN dashboard.student_total_data_latest AS s
    ON c.course_id = s.course_id WHERE c.course_name = '空間資訊與智慧生活' and c.
    course_id='FCUx/mooc0003/201410';",
    },
    {"input": "課程ID為 NCUx/M0000301/201409 該課程的影片觀看人數有多少人?",
    "query": "SELECT course_id, 影片觀看人數台灣, 影片觀看人數_非台灣 FROM dashboard.
    course_total_data_latest WHERE course_id = 'NCUx/M0000301/201409';",
    },
    {"input": "課程名稱為'軟體設計'的課前和課後的課程滿意度分別是多少？",
     "query": "WHERE course_name = '軟體設計' UNION SELECT course_id, course_name, 課
    程滿意度 FROM dashboard.survey_after_course WHERE course_name = '軟體設計';",
     },
]
```

Figure 2. The query conversion between user and the system, asking the course information in the OpenEdu system.

In the agent setup, we configured a prompt named agent prompt using the chat prompt template. This chat prompt model included three role parameters: system, human, and AI. In the system, we added role settings that first instruct the system that it is an assistant responsible for conversing with the database. Additionally, when the system did not know the answer, the "I don't know" prompt appeared instead of fabricating an answer. This method reduced the occurrence of hallucinations by the LLM as shown in Figure 3.

```
agent_prompt = ChatPromptTemplate.from_messages([
    ("system", "You are an assistant responsible for talking to the database"
    ),
    ("system", "If you don't know the answer, return you don't know, don't
    try to make up the answer "),
    ("system", "Answer in Traditional Chinese"),
    ("system", format_template),
    MessagesPlaceholder(variable_name="history"),
    ("user", "{input}"),
    MessagesPlaceholder(variable_name="agent_scratchpad")
```

Figure 3. A prompt example to the agent.

Finally, to see the SQL syntax generated by the LLM and the natural language response, we provided specified output formats for each operation process and ensured that the responses were in traditional Chinese (Figure 4).

```
answer_prompt = PromptTemplate.from_template(
    """You are a MySQL expert.
    Given the following user question, corresponding SQL query, and SQL
    result, answer the user question.
    Use the following format:
    Question: {question}
    SQL Query: {query}
    SQL Result: {result}
    Answer: Answer in Traditional Chinese"""
)
```

Figure 4. Prompt design for asking a specific output.

3.3. System Interfaces

The system was built on the OpenEdu MOOC platform as its interface provides a dialogue window where users input the information they want to query on OpenEdu and receive immediate responses.

4. LangChain Design

We used the SQLDatabase class provided by LangChain to interface with the OpenAI GPT model API and connect to external data sources that record the OpenEdu online education platform's database.

4.1. Sequential Method

To design of sequential query, we obtained the SQL query syntax, executed the SQL query syntax, and converted the results into natural language. Based on these three steps, we named the method a sequential query. All the steps were linked into a chain to query the database using natural language. The overall process is shown in Figure 5.

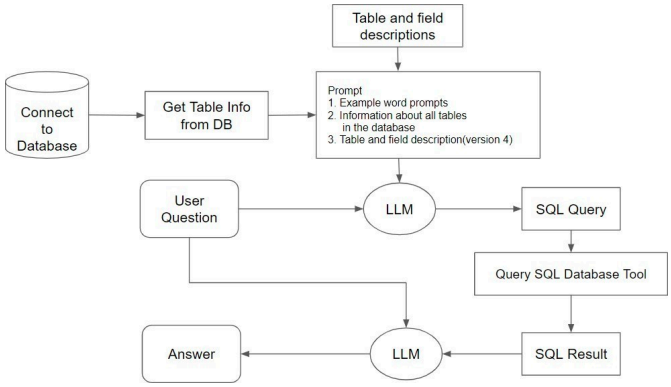


Figure 5. Sequential query process.

First, the LLM receives the user's question. Simultaneously, the LLM, through the LangChain interface connected to the database, obtains the relevant structure and content information. Based on the different prompts, we divided it into two versions: Version 1 and Version 2. Both versions include example prompts and information about all tables in the database, while Version 2 adds natural language descriptions of the database tables and columns. Based on these prompts and the question, the LLM generates the corresponding SQL query syntax.

Next, we use the SQL Database Tool provided by LangChain to execute the generated SQL query syntax. Finally, the LLM uses the execution results and the user's question to generate a natural language answer in the designed output format to respond to the user's question.

4.2. Agent Method

The design of the agent query was based on the concept of agents, of which their core function selected appropriate tools and executed corresponding actions based on the questions. Therefore, we named it agent query. In the agent query method, to improve accuracy, we divided it into two versions based on the LLMs used: Version 3 used the GPT- 3.5-turbo model and Version 4 used the GPT-4-turbo model. Both versions included example prompts, information about all tables in the database, and natural language descriptions of the database tables and columns.

In this study, the SQL Database Chain in LangChain was used in the agent's tool as its internal tool. Each tool needed to be named and described. We named this tool Query

Database which was used to convert the received question into an SQL query, execute the query, and then translate the results into a natural language response.

The operational process of the Query Database tool includes the LLM connecting to the database that receives the database's structure description, table structures, and column formats, and receives the designed prompts. The LLM generates the SQL query and executes it in the database, finally converting the results into a natural language response.

The LLM receives the user's questions, prompts, and historical records. At this point, the agent selects the appropriate tools and executes the corresponding actions based on the user's question. Specifically, the agent constructs a query syntax based on the semantics of the question and passes it to the LLM as a reference for tool use. After the agent arranges all the steps, the Query Database tool generates the SQL query syntax and executes the query to obtain the results. Finally, the LLM converts the query results into a natural language response and stores the question and answer in memory. In the next query, the agent refers to the previous dialogue records to respond. The operation process of the agent query is shown in Figure 6.

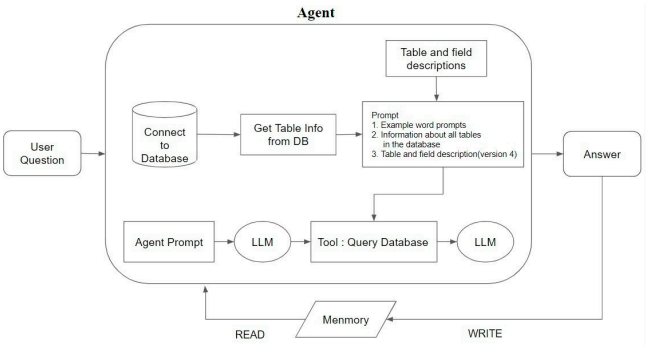


Figure 6. Agent query flowchart.

5. Experiment Design

5.1. Experiment

Version 1 and Version 2 of the sequential query, and Version 3 of the agent query were used with the latest model of GPT-3.5-turbo, specifically the GPT-3.5-turbo-0125 model with 16,385 tokens in the experiment. Version 4 of the agent query was used with the GPT-4-turbo model. The temperature parameter for each version was set to 0. The dataset was obtained from the OpenEdu online education platform's MySQL dataset, which includes 13 tables and a total of 995,975 records.

We designed four query methods based on different components, models, and prompts of LangChain, dividing these systems into four versions. Table 2 provides an overview of each version. To explore the accuracy of these versions under different conditions, we proposed three hypotheses and setup corresponding null hypotheses for verification.

Table 2. Overview of designed versions.

Version	Query	Model	Prompt
V1	Sequential Query	GPT-3.5 turbo	SQL query example; database table info.
V2			
V3			
V4		GPT-4 turbo	

First, we hypothesized that adding natural language descriptions of tables and columns could improve the LLM's understanding of tables and columns. We established the following null hypothesis:

- Natural Language Description

H0: Adding descriptions of tables and columns does not significantly affect the accuracy of responses.

This hypothesis was proposed to determine whether natural language descriptions of tables and columns effectively enhanced the LLM's understanding ability, thereby improving query accuracy.

Next, we explored the impact of the agent's reasoning ability on the accuracy of responses in natural language to SQL conversion and established the following null hypothesis.

- Agent Reasoning Ability Hypothesis

H0: There is no significant difference in response accuracy between sequential query and agent query methods.

This hypothesis was proposed to evaluate whether the agent query method improved the system's response accuracy through its reasoning and planning capabilities.

We investigated the impact of GPT-3.5-turbo and GPT-4-turbo on response accuracy and established the following null hypothesis.

- Model Impact Hypothesis

H0: There is no significant difference in response accuracy between GPT-3.5-turbo and GPT-4-turbo.

Based on these hypotheses in the experiment, we evaluated the performance of different versions of the query system in querying educational platform data.

5.2. Question Design

We designed 42 questions to test the ability of LLMs to convert natural language into SQL. These questions were designed at three difficulty levels: simple, moderate, and difficult. Simple questions required querying a single table, not limited to a single column; moderate questions required querying more than one table or using aggregate functions; difficult questions involved the use of join or group by syntax. We designed 20 simple questions, 11 moderate questions, and 11 difficult questions including course-related questions, student-related questions, student participation questions, and questionnaire-related questions.

Scenarios where a join operation was needed to answer the question were created. The model did not use a join or failed to correctly identify the appropriate tables or columns. This type of error indicated that the model struggled with handling relationships between tables, leading to unsuccessful join operations. Group by errors occur when the model fails to recognize the necessity of grouping and thus does not use group by, or when it groups by incorrect columns. This type of error indicates that the model cannot correctly understand the need for grouping the questions to provide the right answer.

In invalid SQL, syntax errors in the generated SQL were inexecutable. We categorized invalid column queries and invalid table queries from schema linking errors under this error type.

6. Results and Discussions

We evaluated the accuracy and analyzed the error types in each version.

6.1. Discussion of Sequential Method

We tested the four versions with the 42 designed questions and calculated the execution accuracy using (1).

$$Accuracy = \sum_{i=1}^{n} \frac{Q_c}{Qt} \times 100\% \quad (1)$$

where Q_c is the number of correctly executed SQL queries, and Q_t is the total number of queries.

The results showed that the execution accuracy for Sequential Query Version 1 was 83.3%. For Sequential Query Version 2 with added table and column descriptions, it was 85.7%, while for Agent Query Version 3, it was 42.86%. For Agent Query Version 4 using the GPT-4 model, it was 64.29%. The sequential query versions performed significantly better than the agent query versions, with Version 1 having the highest execution accuracy (Figure 7). The agent query versions created an SQL query syntax based on the semantics of the question, leading to errors or queries for columns and tables that did not exist in the database.

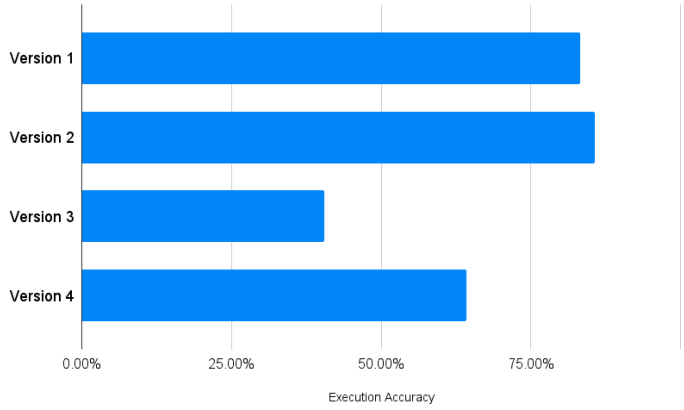


Figure 7. Execution accuracy.

6.2. Evaluation of Our Approach

Error type analysis was conducted for text-to-SQL tasks using the method proposed by Pourreza and Rafiei. SQL errors were classified into schema linking, join, group by, invalid SQL, and wrong answer. We evaluated and analyzed the errors of different complexity levels using the execution accuracy metrics [13].

Schema Linking

Schema linking errors occur when the model fails to recognize the column names, table names, or instances of entities mentioned in the question. For example, in the question "What is the number of registrations in June for the course?", if the database has a table named "register count" recording the total number of registrations for all courses on the platform, the model queries the 'registrations' column in the appropriate table but queries the wrong table instead. We categorized these errors into wrong column query, wrong table query, invalid column query, and invalid table query.

6.3. Hypothesis Verification

We used chi-square tests to analyze and evaluate the impact of different factors such as LangChain components, models, and prompt design on the accuracy of responses across different versions.

In the null hypothesis H_0, it was assumed that adding descriptions of tables and columns does not significantly affect response accuracy. Table 3 shows the cross table of

correct and incorrect responses for these two versions. The chi-square test results showed a *p*-value of 1.0, which was greater than the significance level of 0.05, indicating that adding natural language descriptions of tables and columns did not significantly affect response accuracy. Therefore, the null hypothesis was supported.

Table 3. Cross table for responses in Version 1 and Version 2.

	Version 1	**Version 2**
Correct	35	36
Incorrect	7	6

We analyzed the impact of query methods on response accuracy. We assumed that there is no significant difference in response accuracy between sequential query and agent query methods. Table 4 shows the cross table of correct and incorrect responses for these two versions. The chi-square test results showed a *p*-value of 0.0001, which was less than the significance level of 0.05, indicating a significant difference in response accuracy between these two methods. Therefore, the null hypothesis was rejected.

Table 4. Cross table for responses in Version 2 and Version 3.

	Version 2	**Version 3**
Correct	36	18
Incorrect	6	24

Lastly, we analyzed the impact of the models on response accuracy. We proposed that there is no significant difference in response accuracy between GPT-3.5-turbo and GPT-4-turbo. Table 5 shows the cross table of correct and incorrect responses for these two versions. The chi-square test results showed a *p*-value of 0.0801, which was greater than the significance level of 0.05, indicating no significant difference in response accuracy between GPT-3.5-turbo and GPT-4-turbo. Therefore, the null hypothesis was supported.

Table 5. Cross table for responses in Version 3 and Version 4.

	Version 3	**Version 4**
Correct	18	27
Incorrect	24	15

Adding natural language descriptions of tables and columns in prompts did not impact response accuracy, meaning that table and column descriptions did not significantly change the performance of LLMs in terms of response accuracy. The LangChain components, namely sequential query and agent query methods, significantly affected response accuracy, indicating that the design and use of components improved query accuracy. Finally, there was no significant difference in response accuracy between GPT-3.5-turbo and GPT-4-turbo models, meaning that the choice of model did not impact the query results between these two versions. Table 6 shows the comparison.

Table 6. Chi-square test results.

Comparison	*p*-value	Result
Version 1 vs. Version 2	1.0	Not significant, fail to reject H_0
Version 2 vs. Version 3	0.0001	Significant, reject H_0
Version 3 vs. Version 4	0.0801	Not significant, fail to reject H_0

6.4. Error Type Analysis

Error analysis results are shown in Figure 8. The primary error type across all four versions was schema linking. The frequency of schema linking errors was higher in the agent query versions, specifically including wrong column queries, wrong table queries, invalid column queries, and invalid table queries. In particular, Version 3 showed the highest proportion of errors related to querying incorrect tables or columns, indicating a significant disadvantage in table and column recognition. In contrast, the sequential query versions had a lower frequency of schema linking errors.

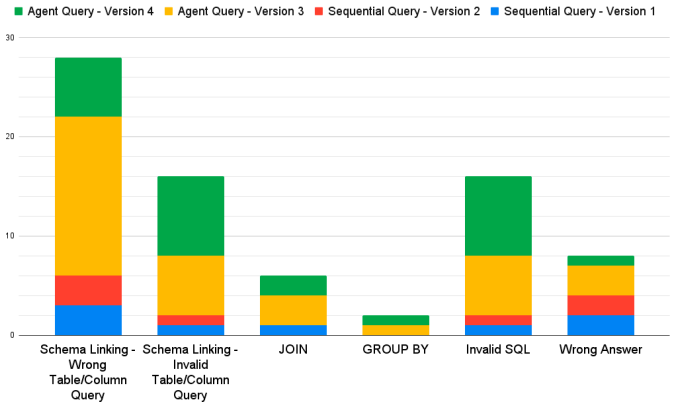


Figure 8. Error type analysis results.

The proportions of join and group by errors were low with no errors in these categories for Sequential Query Version 2, indicating its superior performance in handling inter-table relationships and grouping operations. Additionally, invalid SQL errors occurred in the agent query versions, primarily due to the generation of invalid tables or columns, leading to SQL syntax errors that could not be executed. The wrong answer error type was the least common in Version 4, mainly because most errors in this version were invalid SQL. The proportions of wrong answer errors were similar among the other three versions, indicating that it frequently failed to answer the questions although the SQL syntax was correct.

Overall, the agent query versions exhibit higher error rates across all error types than the sequential query versions, particularly in schema linking and invalid SQL categories. This suggested that agent queries faced more challenges in converting natural language to SQL syntax, especially in correctly identifying tables and columns. The reasoning ability of the agent often confused the LLM, generating tables and columns that did not exist in the database, thereby causing SQL execution errors.

7. Conclusions

We proposed two operational methods and designs based on the LangChain framework, integrating LLMs with an online education platform database to enable interaction with the database. By designing a system workflow for LangChain components,

the database and the few-shot prompting method were combined with descriptions of database tables and columns. We optimized prompt design to improve response accuracy and evaluated the execution accuracy and error types of four versions, each using different combinations of three factors to convert natural language to SQL syntax. The sequential query versions showed higher accuracy than the agent query versions, with errors concentrated in schema linking. This indicated that LLMs often generated SQL syntax inconsistent with database columns and tables based on the semantics of the input questions. Additionally, the phenomenon of hallucinations in LLMs remains an important research topic, requiring further optimization of prompt design to mitigate such issues.

For future improvements and research, it is necessary to optimize the query tools and prompt engineering of the agent at the system level. Utilizing the agent's ability to use various tools and the convenience of LangChain components, it is required to customize and develop operational methods by incorporating additional external tools. Then, the system is optimized, enhancing its accuracy and stability. The agent needs to be integrated into a visual dashboard tool to view educational data quickly and intuitively. To achieve a perfectly integrated data analysis with the OpenEdu database, timely feedback is necessary to enable teachers to analyze course and student performance data efficiently, thereby improving teaching strategies.

Author Contributions: Conceptualization, N.-L.H. and W.-T.W.; methodology, N.-L.H. and W.-T.W.; software, W.-T.W.; validation, W.-T.W.; writing—original draft preparation, W.-T.W.; writing—review and editing, N.-L.H. and W.-T.W.; All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Ministry of Science and Technology, Taiwan, under grant NSTC112-2221-E-035-030- MY2.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to the data contains personal information of participants.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zhao, W.M.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; et al. A Survey of Large Language Models. *arXiv* **2023**, arXiv:2303.18223.
2. Topsakal, O.; Akinci, T.C. Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast. *Int. Conf. Appl. Eng. Nat. Sci.* **2023**, *1*, 1050–1056. [CrossRef]
3. Pandya, K.; Holia, M. Automating customer service using langchain: Building custom open-source gpt chatbot for organizations. *arXiv* **2023**, arXiv:2310.05421.
4. Jacob, T.P.; Bizotto, B.L.S.; Sathiyanarayanan, M. Constructing the chatgpt for pdf files with langchain–ai. In Proceedings of the 2024 International Conference on Inventive Computation Technologies (ICICT), Lalitpur, Nepal, 24–26 April 2024; pp. 835–839.
5. Cheng, L.; Li, X.; Bing, L. Is gpt-4 a good data analyst? *arXiv* **2023**, arXiv:2305.15038.
6. Zhang, W.; Shen, Y.; Lu, W.; Zhuang, Y. Data-copilot: Bridging billions of data and humans with autonomous workflow. *arXiv* **2023**, arXiv:2306.07209.
7. Yu, J.; Luo, G.; Xiao, T.; Zhong, Q.; Wang, Y.; Feng, W.; Luo, J.; Wang, C.; Hou, L.; Li, J.; et al. Mooccube: A large-scale data repository for nlp applications in moocs. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Online, 5–10 July 2020; pp. 3135–3142.
8. Yu, T.; Zhang, R.; Yang, K.; Yasunaga, M.; Wang, D.; Li, Z.; Ma, J.; Li, I.; Yao, Q.; Roman, S.; et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv* **2018**, arXiv:1809.08887.
9. Gao, D.; Wang, H.; Li, Y.; Sun, X.; Qian, Y.; Ding, B.; Zhou, J. Text-to-sql empowered by LLMs: A benchmark evaluation. *arXiv* **2023**, arXiv:2308.15363.

10. Rajkumar, N.; Li, R.; Bahdanau, D. Evaluating the text-to-sql capabilities of LLMs. *arXiv* **2022**, arXiv:2204.00498.
11. Katsogiannis-Meimarakis, G.; Koutrika, G. A survey on deep learning approaches for text-to-SQL. *VLDB J.* **2023**, *32*, 905–936. [CrossRef]
12. Zhong, V.; Xiong, C.; Socher, R. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv* **2017**, arXiv:1709.00103.
13. Pourreza, M.; Rafiei, D. Din-sql: Decomposed in- context learning of text-to-sql with self-correction. *arXiv* **2023**, arXiv:2304.11015.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.