IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

ASKSQL: AN AGENTIC TEXT-TO-SQL SYSTEM USING LLM, POSTGRESQL, AND OLLAMA

[1]Harshitha P, [2]AP Chandrru

[1]Student, [2]Student

[1]Department of Artificial Intelligence and Machine Learning, Jain University, Bangalore, India
[2]Department of Artificial Intelligence and Machine Learning, Jain University, Bangalore, India

Abstract: AskSQL is a Text-to-SQL system designed to help users interact with relational databases using natural language. Instead of manually writing SQL queries, users can enter questions in plain English, and the system converts them into executable SQL statements. The project uses a locally hosted Large Language Model through Ollama along with PostgreSQL for query execution. The workflow includes schema retrieval, SQL generation, validation, execution, and result explanation. To improve security, only safe read-only queries are allowed. Since the model runs locally, the system avoids cloud dependency and provides better privacy with lower operational cost. Experimental testing showed reliable query generation and satisfactory response time for common database operations.

Index Terms - Text-to-SQL, Large Language Model, Ollama, PostgreSQL, FastAPI, Natural Language Processing.

I. INTRODUCTION

In recent years, the amount of digital information stored in databases has increased rapidly across different industries. Retrieving useful information from these databases usually requires knowledge of SQL, which can be difficult for users who do not have a technical background. Because of this, there is a growing interest in systems that allow users to interact with databases using natural language instead of writing SQL queries manually.

Text-to-SQL systems are designed to convert natural language questions into executable SQL statements. Earlier approaches mainly depended on rule-based methods and predefined templates, which often failed when users entered flexible or complex queries. With the development of Large Language Models (LLMs), it has become possible to generate more accurate SQL queries from user input.

The proposed system, AskSQL, was developed to provide a simple and secure way of accessing relational databases through natural language. The system uses a locally hosted LLM through Ollama along with PostgreSQL for database operations. Unlike many cloud-based solutions, the model runs completely offline, which helps improve privacy and reduce operational cost.

AskSQL follows a structured workflow that includes schema retrieval, SQL generation, query validation, execution, and result explanation. Additional safety measures such as regex filtering, read-only query access, and retry handling are included to improve reliability during execution. The overall objective of the system is to simplify database interaction and make data access easier for non-technical users.

In addition, recent research has focused on improving query accuracy using Retrieval-Augmented Generation (RAG) techniques and schema-aware prompting methods. These approaches help language models better understand database structure and relationships between tables. Despite these improvements, many existing systems still face challenges related to query validation, scalability, and handling ambiguous user input in real-world applications.

II. LITERATURE REVIEW

Text-to-SQL systems have been widely researched to simplify the interaction between users and relational databases. Earlier systems mainly depended on rule-based techniques and predefined templates for SQL generation. Although these approaches worked for simple queries, they often failed when users entered flexible or complex natural language inputs.

Recent developments in machine learning and Large Language Models (LLMs) have significantly improved Text-to-SQL performance. Models trained on benchmark datasets such as Spider are capable of generating more accurate SQL queries by understanding context and database schema relationships. These advancements have increased the practicality of natural language database interaction systems.

Several modern solutions provide cloud-based AI services for query generation and data analytics. However, many of these systems depend heavily on internet connectivity and external APIs, which may increase operational cost and create privacy concerns when sensitive data is involved.

Research on Retrieval-Augmented Generation (RAG) and schema-aware prompting has further improved SQL generation accuracy. These methods allow language models to retrieve relevant database schema information before generating queries, reducing errors caused by incorrect table or column selection.

The proposed AskSQL system differs from many existing approaches by using a locally hosted LLM through Ollama along with PostgreSQL for secure database interaction. The system also integrates validation mechanisms and retry-based query correction to improve reliability during execution. By operating fully offline, AskSQL reduces dependency on cloud infrastructure while maintaining efficient query generation performance.

III. PROPOSED SYSTEM

The proposed AskSQL system is designed to convert natural language queries into executable SQL statements using a locally hosted Large Language Model. The system aims to simplify relational database interaction for users who may not have prior knowledge of SQL syntax or database structures.

The workflow of the system begins when the user enters a query in natural language through the frontend interface. The backend retrieves schema information from the PostgreSQL database, including table names and column details. This schema context is then provided to the language model to improve SQL query generation accuracy.

After the SQL query is generated, the system performs validation checks before execution. Regex-based filtering is used to block unsafe operations such as UPDATE, DELETE, and DROP statements. In addition, SQL EXPLAIN validation is performed to verify query structure and syntax correctness.

Once validated, the generated SQL query is executed on the PostgreSQL database, and the results are returned to the user through the frontend interface. If the query execution fails due to syntax or schema-related issues, the error message is used to regenerate an improved query through a retry-based mechanism.

The AskSQL system follows an agentic pipeline consisting of schema retrieval, SQL generation, validation, execution, and result explanation. This modular design improves reliability, security, and overall usability for natural language database interaction.

IV. METHODOLOGY

The methodology of the proposed AskSQL system focuses on converting natural language input into secure and executable SQL queries using a locally hosted Large Language Model. The workflow is designed to improve query accuracy while maintaining system reliability and database security.

The process begins when the user enters a query through the frontend interface developed using React. The request is sent to the FastAPI backend, where schema information such as table names, column names, and relationships is retrieved from the PostgreSQL database. This schema context helps the language model understand the database structure before generating SQL queries.

The retrieved schema details and user query are provided to the LLM running through Ollama. Based on the provided context, the model generates an SQL query that matches the user request. To improve output quality, prompt engineering and schema-aware retrieval techniques are used during query generation.

Before execution, the generated query passes through multiple validation stages. Regex filtering is applied to block unsafe operations including DELETE, UPDATE, DROP, and ALTER commands. The query is also checked using SQL EXPLAIN statements to verify syntax correctness and execution feasibility.

After successful validation, the SQL query is executed on the PostgreSQL database, and the results are returned to the frontend interface. If execution errors occur, the error message is passed back to the language model to regenerate a corrected query. This retry-based mechanism helps improve overall query success rate and system reliability.

The system also includes schema-aware prompting to improve SQL generation accuracy. By providing table names, column details, and database relationships along with the user query, the language model is able to generate more contextually relevant SQL statements. This helps reduce errors related to incorrect table selection and missing attributes.

To improve performance during repeated operations, optimized prompt handling and lightweight caching mechanisms were incorporated into the backend workflow. These techniques help reduce unnecessary processing overhead and improve response consistency during continuous user interaction.

The modular architecture of the proposed system also improves scalability and future extensibility. Since the frontend, backend, language model, and database layers are separated, additional features such as multi-database support, conversational memory, and advanced query optimization can be integrated more efficiently in future versions.

The complete workflow of the proposed system is illustrated in Figure 1, which represents the interaction between the frontend, backend, language model, validation layer, and PostgreSQL database.

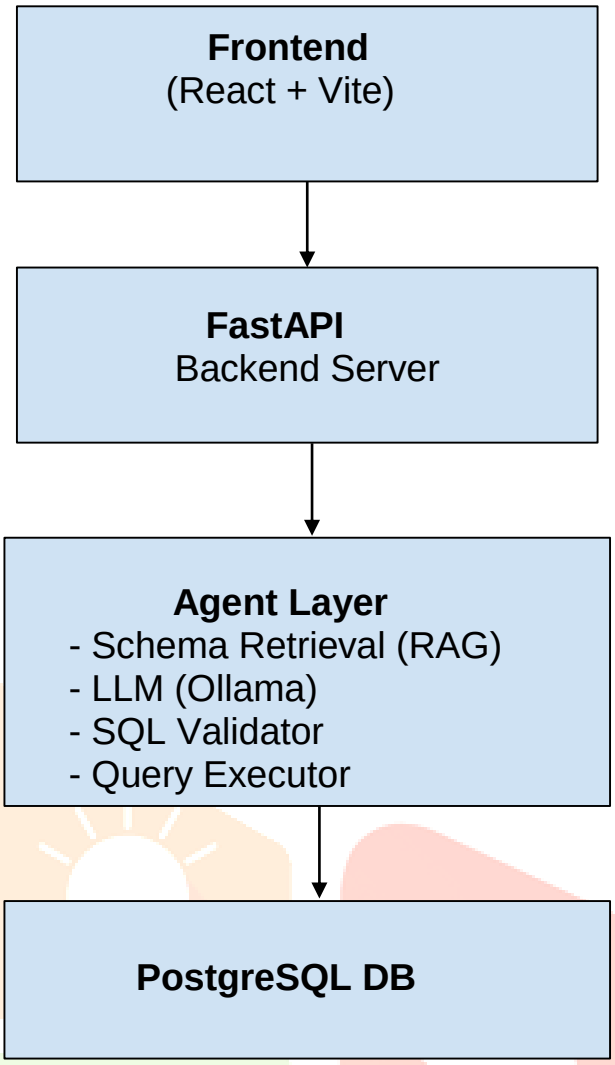

Figure 1: System Architecture of AskSQL

V. Implementation

The AskSQL system was implemented using a combination of modern web development frameworks and database technologies. The frontend interface was developed using React and Vite to provide a simple and responsive user experience for entering natural language queries and displaying query results.

The backend of the system was built using FastAPI, which handles API communication, schema retrieval, query validation, and interaction with the language model. SQLAlchemy was used for database connectivity and query execution within the PostgreSQL environment.

For SQL generation, the system uses the qwen2.5-coder model running locally through Ollama. Since the language model operates entirely on the local system, the proposed approach reduces dependency on external cloud APIs and improves data privacy. The local deployment also helps reduce operational cost while providing more consistent response performance during query execution.

To improve reliability and security, multiple validation layers were integrated into the system. Regex-based filtering blocks unsafe SQL operations, while read-only query restrictions prevent unauthorized database modifications. In addition, SQL EXPLAIN validation is used to verify query correctness before execution.

Caching techniques and optimized prompt handling were also implemented to improve response time and reduce unnecessary processing overhead. The modular structure of the system allows future enhancements and easier integration with additional databases or advanced AI models.

VI. Results and Discussion

The AskSQL system was tested using multiple natural language queries related to customer information, sales records, and order history. The generated SQL queries were executed on the PostgreSQL database, and the system successfully returned accurate results for most test cases.

The integration of schema-aware retrieval improved query generation accuracy by reducing errors related to incorrect table names and missing columns. The retry-based correction mechanism also improved system reliability by regenerating failed queries using database error feedback.

Performance evaluation showed that the locally hosted LLM was capable of generating SQL queries with relatively low latency while maintaining consistent execution accuracy. Since the model operates offline through Ollama, the system avoids external API dependency and reduces operational cost.

The validation layer played an important role in improving database security. Unsafe SQL operations were successfully blocked through regex filtering and read-only query restrictions. SQL EXPLAIN validation further helped detect invalid queries before execution. These validation mechanisms improved system reliability and reduced the chances of executing unsafe or malformed SQL queries during runtime.

The proposed system was also tested with queries of varying complexity to evaluate overall stability and response consistency. Simple queries produced faster responses, while comparatively complex queries involving filtering and aggregation required slightly more processing time. The results showed that the system maintained reliable performance across different query types.

Table 1: Performance Evaluation of AskSQL System

Metric	Value	Description
Average Latency	1.2 seconds	Time taken to process a query
Query Success Rate	90%+	Percentage of successfully executed queries
Retry Count	1-2 attempts	Average retries for failed queries
Accuracy	High	Correctness of generated SQL queries
Response Time	Fast	Improved due to local LLM
Error Handling Efficiency	High	Ability to detect and correct errors

VII. Conclusion

The AskSQL system presents an effective approach for converting natural language queries into executable SQL statements using a locally hosted Large Language Model. The system was developed to simplify database interaction for users who may not have prior knowledge of SQL or relational database structures.

By integrating schema-aware retrieval, query validation, and retry-based correction mechanisms, the proposed system was able to generate reliable SQL queries with satisfactory accuracy and response time. The use of Ollama for local model deployment also improved privacy and reduced dependency on external cloud services.

Security was improved through the use of regex filtering, read-only query restrictions, and SQL EXPLAIN validation before execution. These validation mechanisms helped prevent unsafe database operations while maintaining stable query performance during testing.

Experimental evaluation showed that the system can effectively support natural language database interaction in practical scenarios. Overall, AskSQL demonstrates the growing potential of AI-driven systems in improving accessibility, efficiency, and usability in database management applications.

VIII. Future Work

Although the proposed AskSQL system demonstrated reliable performance during testing, several improvements can be implemented in future versions to enhance functionality and usability. One possible extension is support for multiple database systems such as MySQL, MongoDB, and SQLite in addition to PostgreSQL.

Future versions of the system can also include conversational memory to support multi-turn interactions, allowing users to refine queries through continuous conversation. This would improve the user experience while handling complex database requests.

The integration of voice-based interaction can further improve accessibility for users who are unfamiliar with typing SQL-related queries. In addition, fine-tuned language models trained on domain-specific datasets may improve SQL generation accuracy for specialized applications such as healthcare, finance, and education.

Further improvements may include advanced schema learning, real-time monitoring, role-based access control, and dashboard integration for enterprise-level usage. The system can also be extended to support federated querying across multiple databases while maintaining secure and efficient query execution.

Future research can also focus on improving query optimization techniques to reduce execution time for large-scale databases. The integration of feedback-based learning mechanisms may help the system continuously improve SQL generation accuracy based on user interactions and execution history.

REFERENCES

[1] X. Chen, C. Liu, and D. Song, “Text-to-SQL in the Wild: A Naturally Occurring Dataset Based on Stack Exchange Data,” Proceedings of ACL, 2018.

[2] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, and others, “Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task,” Proceedings of EMNLP, 2018.

[3] X. Liu, “A Survey of Text-to-SQL in the Era of LLMs,” IEEE Transactions on Knowledge and Data Engineering, 2024.

[4] Z. Hong, “Enhancing SQL Generation with Data Expert LLM,” Findings of ACL, 2024.

[5] PostgreSQL Global Development Group, “PostgreSQL Documentation,” Available: https://www.postgresql.org/docs/

[6] Ollama, “Ollama: Run Large Language Models Locally,” Available: https://ollama.com/

[7] FastAPI, “FastAPI Framework Documentation,” Available: https://fastapi.tiangolo.com/

[8] T. Brown, B. Mann, N. Ryder, M. Subbiah, and others, “Language Models are Few-Shot Learners,” Proceedings of NeurIPS, 2020.