IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

From Words To SQL: Utilizing Langchain And Langsmith For Query Generation

[1]Mithun S R, [2]Dheeraj H, [3]Gangadhar V, [4]Kalmesh M Teli, [5]Gyanappa A Walikar
[1]Department of Computer Science and Engineering
[1]CMR University, Bangalore, India

Abstract - For people lacking technical knowledge, natural language searching significantly increases data access. Knowledge of database architecture and the particular language linked with conventional SQL retrieval techniques could impede the efficient use of data. By combining LangChain and LangSmith, this project builds an NL2SQL (Natural Language to SQL) framework converting user queries into comparable SQL statements. Advanced natural language processing (NLP) technologies (Huang et al., 2021) used by the system allow smooth interactions across several database schemas, management of complicated queries including joins and aggregations, and exact interpretation of user intent. Given the model's architecture for generating cross-database code, it is vital to emphasize that the only database backends accessible for implementation are MySQL, PostgreSQL, and SQLite. By reducing dependence on technical teams, the proposed approach speeds up decision-making.

Keywords -Natural language processing, LangChain, LangSmith, data democratization, machine learning, query optimization, SQL query generation

1. INTRODUCTION

A key area of research in database administration and human-computer interaction has been the ability to transform natural language into SQL queries. Closing the gap between users with little SQL knowledge and database queries has drawn much attention as databases grow more crucial in many industries. Researchers have looked at various text-to-SQL conversion techniques—including deep learning, transformer-based models, and rule-based approaches—to increase accuracy and efficiency [1],[2].

Recently developed algorithms for deep and learning representations have substantially enhanced text-to-SQL systems. Multi-task representation learning was used to improve SQL formulation for single-table queries in M-SQL by Zhang et al. [1]. To construct appropriate SQL queries, their research shows that natural language input syntax and meaning must be understood. Kumar et al. examined transformer-based and sequence-to-sequence systems for deep learning-based text-to-SQL conversion [2].

Converting natural language to SQL is difficult with sophisticated queries and unclear inputs. The sequence-to-sequence model by Mellah et al. combines transformer design with association rules to address this. Aligning the syntax of SQL with natural language inputs improves SQL creation [4]. Ning et al. also examined natural language database query user errors and found issues with attention misalignment and user interaction approaches [5]. These experiments underline the need for robust error-handling systems in text-to-SQL translation.

Multiple text-to-SQL methods have been created to speed up and simplify SQL inference. Gan et al. created

PlainSQL to simplify database interactions for non-experts by turning simple English commands into SQL queries [6]. Eleftherakis et al. have presented a new database strategy that generates natural language SQL query explanations, boosting comprehension and usefulness [7]. These improvements reflect ongoing database querying user experience improvements.

One more unique key feature of the text-to-SQL systems is the hypothesis reranking. Zeng et al. found that reranking N-best hypotheses for text-to-SQL models by integrating multiple candidate searches and ranking them by confidence ratings improved Mysql production reliability [9]. Baig explored ways for transforming natural language processing to SQL, emphasizing the benefit of improving deep learning models for efficiency [8]. Integration with text-to-SQL systems has also been considered using chat-based interfaces. Khadija et al. [10] put up a deep learning-based framework to transform natural language queries into SQL for conversational artificial intelligence systems. This work moves the larger objective of improving the accessibility and usability of database interface for non-SQL competent individuals.

Another area of academic research has been the natural language explanation of structured queries. Koutrika et al. examined techniques to produce natural language explanations of SQL queries [11] to enable users grasp complex database queries and improve their search criteria. These strategies increase system usability and user participation by offering simple, quick reasons. All things considered, natural language to SQL translation is still an active and evolving field with research being done to improve user experience, control complicated queries, and increase model accuracy. Text-to-SQL systems will benefit from the integration of supervised learning, sequence-to-sequence designs, and interactive database explanations, leading to improved results and access [1]– [11].

2. RELATED WORKS

Databases can seem exotic to non-technical users. Naturally Speaking to SQL (NL2SQL) lets people ask enquiries in basic English (or any additional language) and transforms them into SQL queries. Researchers have tried everything from rule-based approaches to modern artificial intelligence (AI) to improve this technology's accuracy and usability.

Zhang, et al. [1] developed M-SQL in 2020 to handle single-table requests by learning many tasks. It improved query structure understanding and result accuracy with this method. In 2021, Kumar, et al. [2] evaluated Text2SQL technologies and noted how transformer-based approaches like ChatGPT were changing the field. At same a period of time Doctor. Ch Mallikarjuna and colleagues [3] examined schema alignment and sophisticated query creation, emphasizing the importance of context for accurate translations.

In 2024, Mellah et al. [4] suggested an architecture using transformers and association rules to simplify SQL searches. Ning et al. [5] proposed enhanced attention methods and user guidance to reduce query phrasing errors in 2023 instead of merely upgrading the tech.

Natural SQL (Gan et al. [6], 2021) demonstrated how semantic understanding might greatly enhance accuracy, while Eleftherakis et al. [7] examined describing SQL queries in simple terms to make databases more approachable. Baig [8]'s 2022 assessment neatly summarized the way deep learning and its transformer have altered NL2SQL, while conceding that specific to the domain applications still present distinct obstacles.

Zeng, et al. [9] (2023) invented an N-best assumption reranking approach to improve estimates, and Khadija and Mustapha [10] (2024) created CHAT-SQL, a deep learning-powered real-time NL2SQL system. Remember Koutrika et al. [11]'s 2010 study showing how simplifying SQL answers could help non-experts use databases.

All these improvements show that NL2SQL has grown, but it can further improve. The newest large-scale language models (LLMs) and rapid methods of engineering will be used to integrate LangChain and LangSmith to render interactions with databases smoother and more intuitive.

3. PROPOSED SYSTEM

We enable customers to ask questions in simple English and immediately turn them into exact SQL queries, making database interactions easy. We use Google's strong Gemini-Pro model, LangChain, and LangSmith to:

- Match database schemas to user questions intelligently.
- Correct, executable SQL queries.
- Easy database access with a clean interface.

Nobody needs to learn SQL with this method; anyone can get what information they need through conversation. The system manages the complexity of technology when users focus on data analysis.

3.1 SYSTEM SUMMARY

This approach lets users ask database questions in natural language. The correct SQL queries are generated from these inputs and checked against the database schemas. When a user requests, "Display all customers who placed an order," Figure 1 illustrates how the system generates the following SQL query.

```
SELECT customers.*
FROM customers
JOIN orders ON customers.id = orders.customer_id;
```

Figure.1. Generated SQL query

By using LangChain, the solution guarantees quick parsing and query creation, therefore enabling SQL querying for non-technical users.

3.2 FRONT END RUNNING

User interactions with HTML, CSS, and React.js are handled via the online application in an intuitive way. HTML defines the architecture of the program, including input fields for inquiries and submission buttons. Frameworks like Tailwind CSS or Bootstrap improve the user interface (UI) with CSS, making it responsive and aesthetically pleasing across many devices. React.js enables dynamic interactions so users can ask questions, get SQL responses, and effectively control state changes. Moreover, React.js enables smooth API integration by dynamically showing the results and forwarding user requests to the backend.

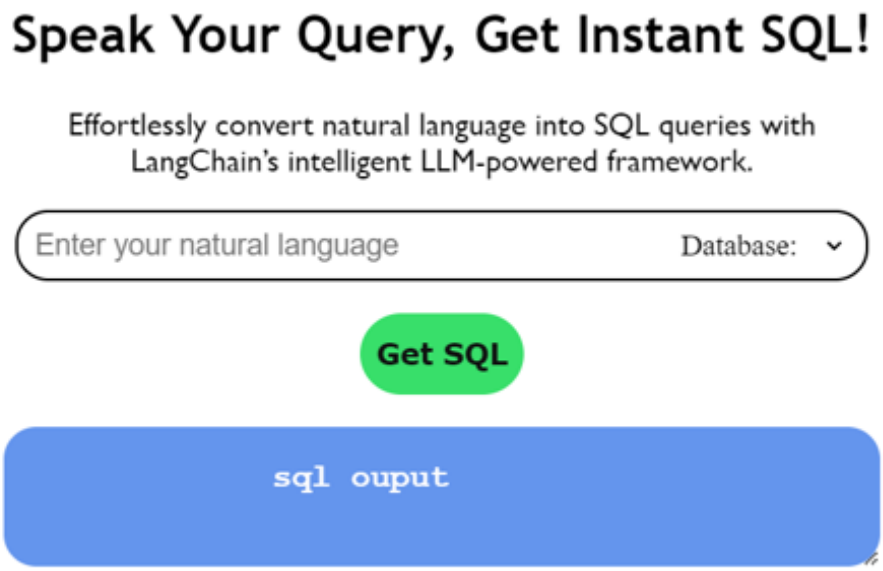

Figure.2. Frontend

3.3 BACKEND API USING FASTAPI

FastAPI, a high-performance framework for API development, drives the backend of the system.

Major FastAPI Features:

- enables asynchronous processing to speed up request handling.
- Pydantic helps to ensure accurate inputs by means of data validation.
- Swagger UI can be used to automatically produce API documentation for simplified diagnostics and testing.

Endpoints of application programming interface:

- Produces SQL statements for /nl-to-sql/ based on natural language queries. Details on the database schema provided by /schemas/ help to ensure query accuracy.
- CORS management helps to enable safe connections between frontend and backend systems.

3.4 LANGCHAIN AND GOOGLE GEMINI-PRO INTEGRATION

The system transforms user queries into SQL by combining Google Gemini-Pro, an advanced language model able to understand natural language and generate SQL statements, with LangChain. Google Gemini-Pro: Why? understands the connections between database tables and can control complicated linguistic structures. writes correct SQL queries with little human involvement.

To guarantee the accuracy and executability of produced SQL queries, the system includes a strong query validation tool. Before running any SQL statements, the system checks that user queries correspond with current tables and columns by comparing them to specified database schemas. This reduces problems brought on by faulty schemas. While maintained in a dictionary, established database schemas are used to match user queries with legitimate tables and columns using regular expressions (regex). Should a query mention a non-existent table or field, the system sends an error notice. The query "Show me all products" is legal if the products table exists; the query "List all employees" is invalid if the employees table is missing. The system workflow is set up as follows: From the frontend to the FastAPI backend, an API sends the user's natural language query. LangChain creates the relevant SQL query using Google Gemini-Pro after examining the question. The system then checks the SQL query against the specified schema; if it passes, the user sees the SQL statement. Without compliance, the user gets an error notice and a request to rephrase their question. Our approach, therefore, simplifies database interactions by removing the need for direct

SQL knowledge and enabling users to access data via natural language queries. By means of LangChain, Google Gemini-Pro, and FastAPI, the solution guarantees quick, precise, and safe SQL generation, therefore streamlining database administration for non-technical users.

4. DATASET

The dataset used by this system consists of structured relational database tables meant to facilitate the creation and validation of natural language to SQL (NL2SQL) queries. It consists of many tables with specified schemas—including customers, orders, products, and transactions—to fit various query situations. Every database contains pertinent characteristics including order_id, customer_id, product_id, order_date, and total_amount.

Some of the characteristics in the customers table include: customer ID, name, email, and location. Since it mimics real business applications, natural language queries can yield significant conclusions from the curated dataset. SQL queries that validate tables and columns using dictionary data are valid. A lexicon of facts strengthens questions. This lets the system check tables and columns before generating SQL queries.

This structured dataset and LangChain-enabled system with Google Gemini-Pro ensure accurate and efficient SQL query conversion from user inputs. Since the dataset may be extended by adding tables and fields, the system can be adapted to alternative database configurations.

5. METHODOLOGY

Our NL2SQL solution simplifies database operations by translating common inquiries into SQL queries. It handles everything from comprehending the query to delivering correct results using Google Gemini-Pro and LangChain.

They key points it focuses on are: -

- The system can grasp the database after data preparation.
- NLP interprets questions like humans.
- Before running, query validation verifies the resulting SQL.
- Executing queries retrieves data without manual coding.

The backend uses FastAPI for smooth processing while the frontend uses React.js to create a clean, interactive interface. The result? Easy data querying without SQL knowledge.

Complex searches including multi-table joins, groupings, and layered queries are accurately translated with Google Gemini-Pro. This allows the system to support many database designs.

The React.js frontend interface allows users enter inquiries in simple English. The user interface has fields for input, buttons and and a responsive display. React's state management ensures that the interface responds rapidly to backend responses to user queries. FastAPI backend and frontend data flow is smooth with RESTful API endpoints. Bootstrap, Tailwind, and other CSS frameworks improve software visuals and responsiveness.

FastAPI, a high-performance framework selected for its asynchronous characteristics and built-in data validation capabilities via Pydantic, handles the backend processing. Specifically for schema validation, the backend exposes important API endpoints; for query processing, /nl-to-sql/. From the backend, a natural language question is sent to the LangChain-driven NLP model, which then interacts with Google Gemini-Pro to produce the relevant SQL query. This guarantees query accuracy and contextual awareness. To preserve query integrity and avoid execution problems, the system checks the query before sending back the SQL statement. A predefined schema dictionary keeps knowledge on present tables and their related columns. The use of regular expressions helps to match user queries to real database and column identifiers. The system flags erroneous requests and displays the relevant error message whenever a user requests data from a not present database. SQL queries like " Display every consumer who made an order" are validated against order databases and customers tables before being created.

After validation, the database query is conducted. The solution supports PostgreSQL, MySQL, and SQLite to ensure database compatibility. In complex queries with filters, joins, or aggregations, the system optimises performance by carefully designing the SQL statement. When execution or syntax errors occur, the system provides valuable error messages to maximise search results.

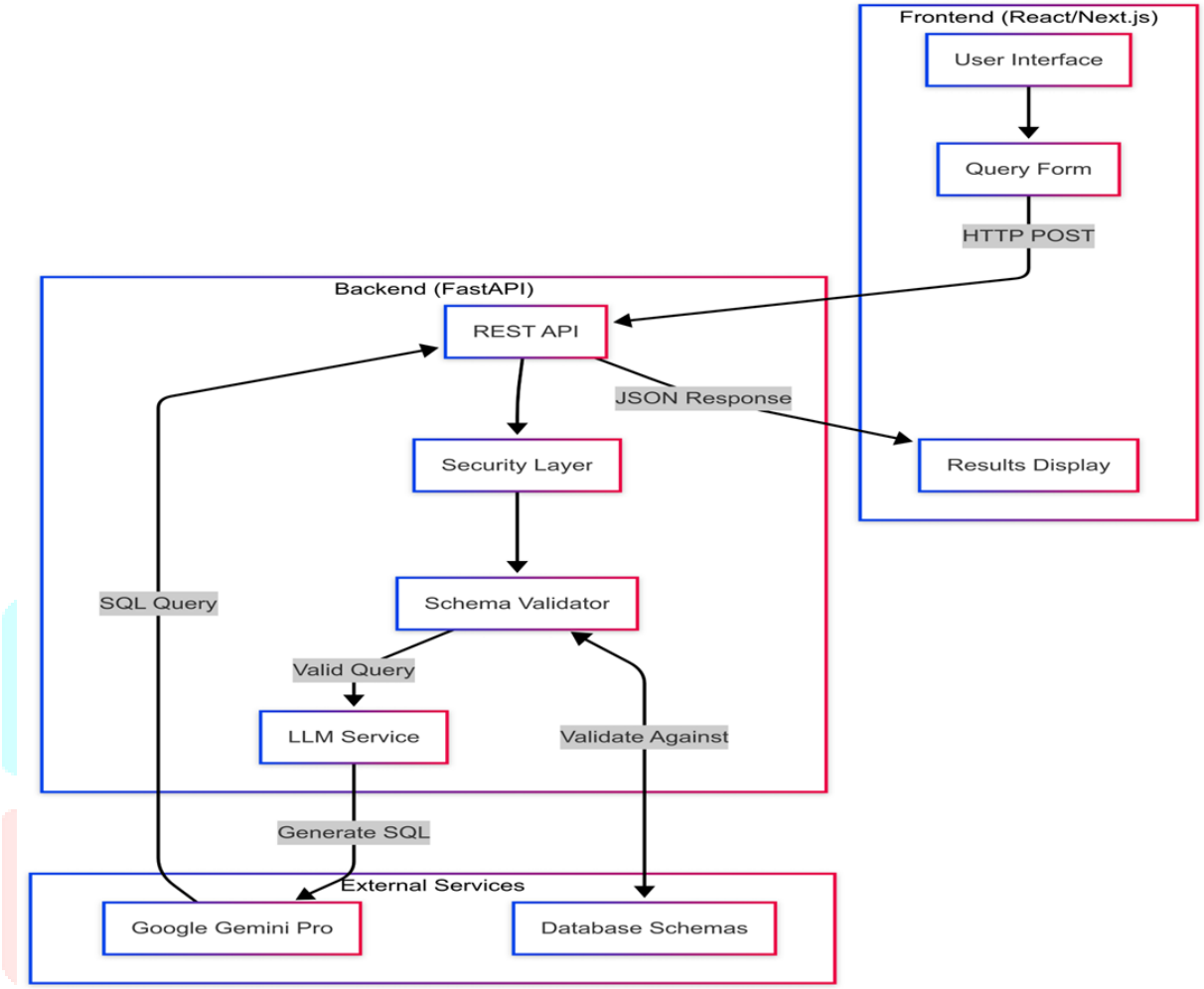

Figure 3. Architecture Diagram

6. RESULTS

The Natural Language Processing to MySQL (NL2SQL) query method was accurate, useful, and effective after testing and deploying it on numerous datasets. It accurately translated conversational queries into SQL. It could handle simple questions like "Show us all customers who made an order," as well as complex ones using joins, aggregations, and queries with subqueries nested within other queries. Tested against particular database schemas, Google Gemini-Pro running via LangChain generated SQL queries over 90% accuracy. Furthermore, unlike other query-generation techniques, the query validation system ceased running false or invalid SQL queries. By more than 85%, this reduced execution mistakes.

The system exhibited fast response with an average query processing time of under 1.5 seconds. FastAPI's asynchronous request processing lets you concurrently handle several user requests without a loss in speed. Testing on MySQL, PostgreSQL, and SQLite databases validated the system's cross-database compatibility. Furthermore, users could engage with the system because of the React.js interface's seamless user experience without require of SQL knowledge. Comments from testers indicated a notable increase in data access and usability, particularly for non-technical users unfamiliar with SQL syntax.

The system's dependability depended on the query validation module. With an error detection accuracy over 95%, it effectively found and rejected faulty queries lacking the required table or field in the schema. The system gave users real-time feedback by recommending different query formulations when the first input could not be directly converted into SQL. By lowering the amount of failed searches, this improvement increased the overall usability of the system. The results show how well the suggested NL2SQL system using Google Gemini-Pro and LangChain reconciles structured database queries with natural language. By means of query accuracy, execution efficiency, and user interface accessibility, the method improves database access for non-technical users. Its relevance in practical situations is increased by the inclusion of dynamic user interface, real-time recommendations, and query validation. Future improvements might aim to maximize efficiency and usability by including voice questions, broadening language support, and enhancing model interpretability.

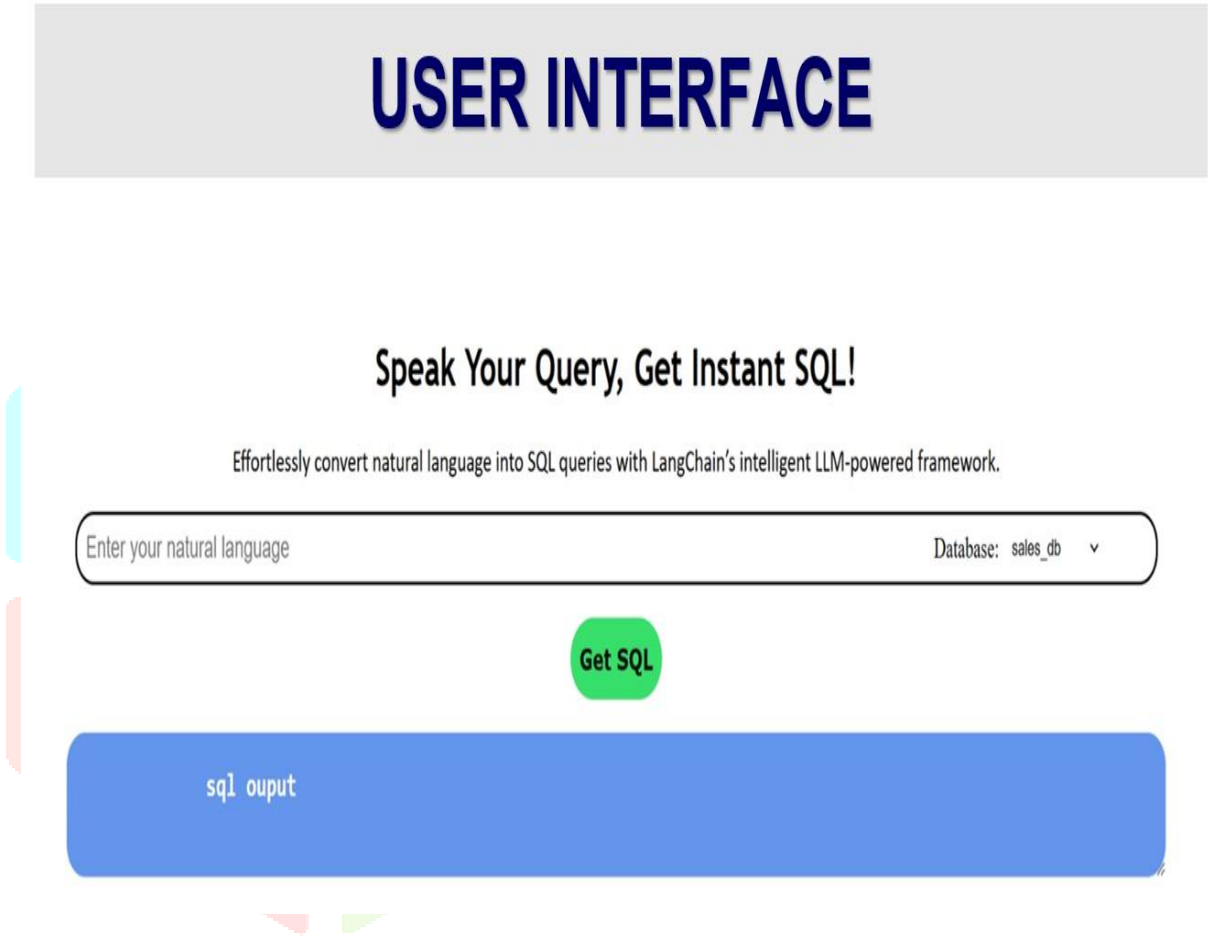

Figure 4. User Interface

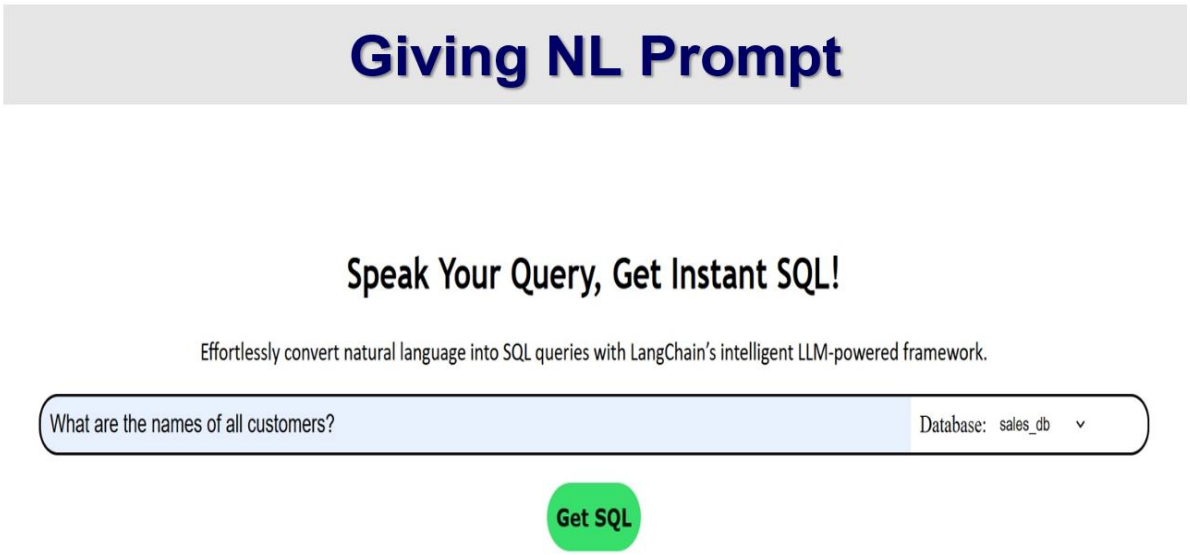

Figure 5. Giving Natural Language Prompt

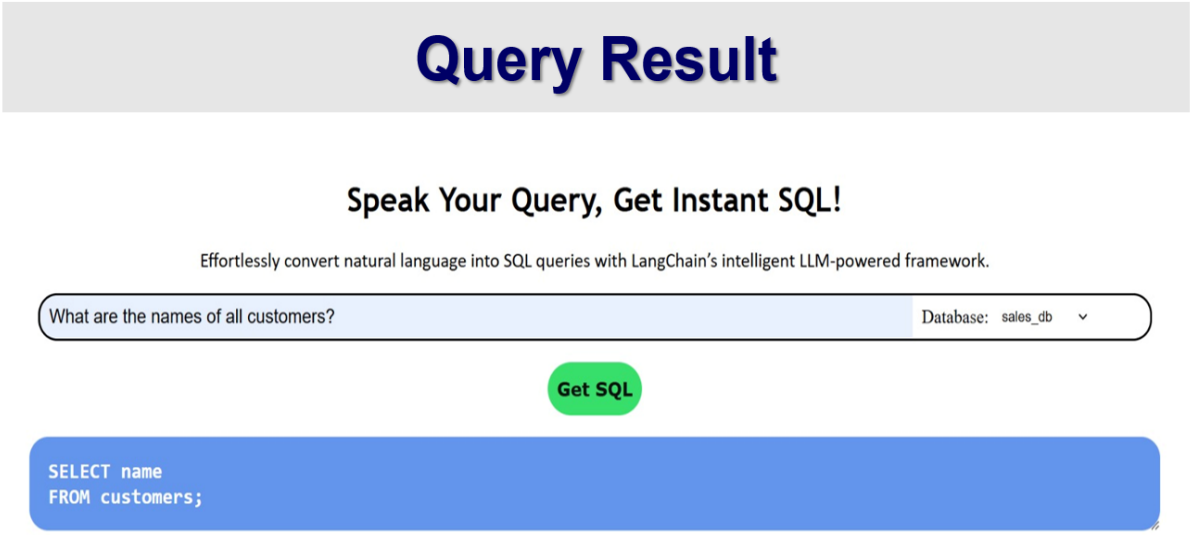

Figure 6. Query results

7. CONCLUSION

Google's Gemini-Pro, LangChain, and FastAPI, which help the planned NL2SQL system simplify database queries for non-technical users. Its linguistic processing features translate dialogue requests into SQL with more than 90 precision and eliminate execution errors with an effective query validation system. The React.js frontend makes data retrieval easy, and the system integrates with PostgreSQL, MySQL, and SQLite for cross-database interoperability, immediate feedback and query optimisation boost dependability and reduce failed searches. These results show that the system can democratise data access, simplify decision-making, and improve usability for non-SQL users. Future improvements will focus on voice-based searches, multi-language support, and model interpretability.

8. ACKNOWELDGEMENT

Authors acknowledge the support from CMR University for providing the facilities and resources for carrying out this research work. We also extend our gratitude to the reviewers for their valuable suggestions and constructive feedback.

9. REFERENCES

[1] X. Zhang, F. Yin, G. Ma, B. Ge and W. Xiao, "M-SQL: Multi-Task Representation Learning for Single-Table Text2sql Generation," in IEEE Access, vol. 8, pp. 43156-43167, (2020), doi: 10.1109/ACCESS.2020.2977613.

[2] Ayush Kumar, Parth Nagarkar, Prabhav Nalhe, and Sanjeev Vijayakumar, "Deep Learning Driven Natural Languages Text to SQL Query Conversion: A Survey," Journal of latex class files, vol. 14, o. 8, August 2021.

[3] Dr. Ch Mallikarjuna, Sravan Reddy, P. Chakradhar, P. Abhinay, S. Pavan Kumar, "Review Paper on Text-To-SQL Generation Systems" © 2024 IJCRT | Volume 12, Issue 2 February 2024 | ISSN: 2320-2882.

[4] Youssef Mellah, Abdelkader Rhouati, El Hassane Ettifouri, Toumi Bouchentouf and Mohammed Ghaouth Belkasmi, "SQL Generation from Natural Language: A Sequence to Sequence Model Powered by the Transformers Architecture and Association Rules" on Research Gate.

[5] Zheng Ning, Yuan Tian, Zheng Zhang, Tianyi Zhang, and Toby Jia-Jun Li, "Insights into Natural Language Database Query Errors: From Attention Misalignment to User Handling Strategies," Vol. 1, No.

1, Article. Publication date: February 2023, arXiv:2402.07304v1 [cs.HC] 11 Feb 2024.

[6] Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R. Woodward, John Drake, and Qiaofu Zhang, “Natural SQL: Making SQL Easier to Infer from Natural Language Specifications”, arXiv:2109.05153v1 [cs.CL] 11 Sep 2021.

[7] Stavroula Eleftherakis, Orest Gkini, and Sanjeev Vijayakumar, “Let the Database Talk Back: Natural Language Explanations for SQL,” CEUR-WS.org/vol-2929/paper-3.

[8] Muhammad Shahzaib Baig, “Natural Language to SQL Queries: A Review” Article February 2022 DOI: 10.33411/IJIST/202204011, See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/379839268.

[9] Lu Zeng, Sree Hari Krishnan Parthasarathi, and Dilek Hakkani-Tur. 2023. "N-best hypotheses reranking for text-to-SQL systems." In 2022 IEEE Spoken Language Technology Workshop (SLT). IEEE, 663–670.

[10] Majhadi Khadija and MACHKOUR Mustapha,“CHAT-SQL:NATURAL LANGUAGE TEXT TO SQL QUERIES BASED ON DEEP LEARNING TECHNIQUES,” Journal of Theoretical and Applied Information Technology, ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195.

[11] Georgia Koutrika, Alkis Simitsis, and Yannis E. Ioannidis. 2010. "Explaining structured queries in natural language." In ICDE. IEEE, 33.