

NATURAL LANGUAGE DATABASE HELPER: AN INTELLIGENT AGENTIC WORKFLOW FOR AUTOMATED SQL QUERY GENERATION

Samina Anjum¹, Sarahan Sheikh², Sayyed Fardeen³, Mohammad Salman⁴, Krish Gajbhiye⁵, Syed Eltefatuddin⁶

¹²³⁴⁵⁶*Department of Computer Science & Engineering
Anjuman College of Engineering & Technology, Nagpur, Maharashtra, India*

Abstract - The Natural Language Database Helper (NLDH) is a smart web-based application that aims to provide a technical interface between human language and complicated database management systems. In the current data intensive environment, retrieving data may demand expertise in Structured Query Language (SQL) and this is a major challenge to the non-technical stakeholder. This project enhances easy interaction with the database by enabling users to query relational databases using plain English. The system is based on the Gemma2:9B model using Ollama by processing large language models (LLM) locally, which guarantees high performance and data privacy. The fundamental novelty is a Two-Step Agentic Workflow: the LLM will find the tables that are relevant to the query and schema overview provided by the user; it will then use the finer metadata of the respective tables to come up with an exact and executable SQL query. The application is a decoupled architecture with an interactive JavaScript front-end and a strong back-end based on PHP to secure communications via API. This helps to increase the efficiency of operations, minimize human syntax errors, and democratize the data access.

Key Words: - Natural Language Database Helper, Text-to-SQL, Gemma2:9B, Ollama, Agentic Workflow, PHP, Decoupled Architecture, SQL Automation, Large Language Model, NLP.

1.INTRODUCTION

The development of the information era is entrenched in the capacity to store, manage and access information efficiently and effectively. RDBMS MySQL, PostgreSQL, and Oracle have enabled important industries, including finance, healthcare, education, and governance, to maintain consistency, integrity and access to data.

The Structured Query Language (SQL) has historically been the main interface to interact with databases and it has been initially developed at IBM in the 1970s. SQL is very effective where trained professionals are involved however, it has steep learning curve among non-technical users. To build a query, one needs to know the syntax, as well as the inner workings of the database (schema). As an example, to retrieve monthly sales data, one may need to know several tables, JOIN statements, filtering and aggregation functions.

This complexity instills a reliance on technical staff members and impedes decision-making. The study of Natural Language Interfaces to Databases (NLIDB) started as early as the 1970s with rule-based systems like LUNAR project; but these were inflexible and too strict. Later machine learning methods enhanced flexibility, but did not have great contextual awareness.

The advent of Large Language Models (LLMs) has been a breakthrough, as now machines can interpret complex and ambiguous queries with high accuracy. Local LLM-hosting models, e.g. Ollama, additionally focus on issues of data privacy, latency, and reliance on 3rd-party clouds.

It is against this background that the Natural language Database Helper (NLDH) is suggested, as a cross-platform, smart web application that enables users to query relational databases in natural language, thereby making SQL a non-condition to interacting with data. This system will integrate LLM-driven knowledge with a safe, on-premises pipeline and thus make the data accessible to both technical and non-technical user groups democratically.

2.LITERATURE REVIEW

Creation of NLIDBs has been a key study topic at the crossroads of Database Management Systems and Artificial Intelligent. The surveyed studies below can be used to track the development of rule-based parsers to the present-day LLM-based systems and outline the most significant research gaps that the proposed NLDH fills.

Table 1. Summary of Related Literature

Sr.	Author & Year	Paper / Book Title
1	C. J. Date (2004)	An Introduction to Database Systems
2	Russell & Norvig (2020)	Artificial Intelligence: A Modern Approach
3	Goodfellow et al. (2016)	Deep Learning
4	Dong et al. (2025)	A Survey on LLMs for NL to SQL (IEEE)
5	Hampp & Mirylenka (2024)	Leveraging LLMs for NL to SQL Conversion (IBM)
6	Mohammadjafari et al. (2024)	Review of LLM-based Text-to-SQL Systems
7	Yu et al. (2018)	Spider: Large-Scale Cross-Domain Semantic Parsing
8	Li et al. (2023)	ResSQL: Decoupling Schema Linking and Skeleton Parsing

A. Drawbacks of Existing Systems

An in-depth analysis of the current NL-to-SQL systems demonstrates that they have six key limitations that the NLDH successfully eliminates.

- **Rigidity and Domain Dependency:** Early rule-based interfaces are strictly domain-dependent, and need manual linguistic rules to deal with each new database.
- **Critical Privacy Risks:** Cloud-hosted LLMs (e.g., GPT-4, Gemini) need the full schema of a database to be sent to third-party servers, which is against HIPAA, PCI-DSS, and other data sovereignty laws.
- **Computational Latency:** State-of-the-art deep learning models are costly to maintain gpus infrastructure and create unacceptable latency in typical business settings.
- **SQL Hallucination:** LLMs often generate non-existent column names or use the incorrect aggregation functions, generating silent logical errors.
- **Context Window Overflow** Single-step stuff-all-tables prompting of the enterprise database (hundreds of tables) gets beyond the AI context limits, leading to the lost in the middle effect.
- **Dependence on labelled datasets:** Machine learning systems need thousands of manually labeled NL-SQL pairs per schema, which cannot be retrained at prohibitive cost.

3.PROPOSED SYSTEM

A. Problem Statement

An intelligent, secure, and user-friendly system is required that allows users to interact with relational databases using natural language, automatically converts user queries into accurate SQL statements, and executes them securely while maintaining data privacy and system efficiency. The gap between the current state (SQL expertise required) and the desired future state (natural language accessibility) is the absence of an intelligent, privacy-preserving, locally deployable interface.

B. System Architecture

The NLDH adopts a multilayered decoupled architecture comprising four distinct components:

- **User Interface Layer (Frontend):** Developed in HTML5, CSS3, JavaScript (ES6+), and Tailwind CSS. It takes natural language as an input and gives a structured query output.
- **Application Layer (Backend):** PHP written with PDO to provide secure database connectivity.

Handles API calls, validation of queries and interaction between the front end and AI layer.

- AI Processing Layer: Deploys Gemma 2:9 B model using the Ollama framework of local, privacy-preserving LLM inference. Adopts a Two-Step Agentic Workflow.
- Database Layer: MySQL, PostgreSQL and SQLite. Runs verified SQL and yields the structured results.

C. Two-Step Agentic Workflow

The main innovation of NLDH is the Two-Step Agentic Workflow, which aims to address the context-overflow and hallucination constraint of single-step prompting.

Step 1 - Table Identification (Filtering Agent): The user query is joined with a high-level overview of the schema (table names and brief descriptions). The LLM is able to isolate only the required tables needed to respond to the query and reject noisy schema elements.

Step 2 - SQL Generation (Generator Agent): LLM is fed with detailed metadata (column names, data types, primary/foreign key relationships) of the tables of interest only. This refined context leads to the use of the model to produce accurate, syntactically correct and semantically correct SQL statements.

This two-step filtering method maximizes utilization of the LLM context window, minimizes hallucinations and remarkably enhances the accuracy of JOIN in sophisticated multi-table queries.

D. System Flowchart

The operational pipeline of a single query lifecycle proceeds as follows: (1) User inputs natural language query → (2) Schema Extraction Engine retrieves metadata → (3) Prompt Synthesis combines query + schema → (4) Filtering Agent identifies relevant tables → (5) Generator Agent produces SQL → (6) PHP validation layer filters harmful commands (DROP, DELETE, TRUNCATE) via PDO → (7) Validated query executes on the database → (8) Results rendered as interactive HTML table.

E. Objectives

- Design an application in web based that takes input in natural language.
- Implement a Two-Step Agentic Workflow that transforms NL queries to SQL.
- Have dynamic schema extraction in all relational databases that are connected.
- Secure query execution by validation and filtering keywords.

- Privacy-preserving processing built-in with a local LLM (Gemma2:9B via Ollama).
- Scale to MySQL, PostgreSQL and SQLite environments.

4.METHODOLOGY

The approach is based on a Modular Agile Development Lifecycle that is organized in eight iterative cycles:

A. Requirements Analysis

Determination of user requirements, system goals and difficulties in translating natural language to SQL. Patterns of queries, database layouts and application requirements are examined with particular reference to usability by non-technical users.

B. Natural Language Understanding Module.

The NLU component extracts the intent, entities and contextual relationships to interpret the user input. Semantic understanding was done using the transformer-based Gemma2:9B model relying on part-of-speech context and schema-aware prompting.

C. Query Generation and Mapping.

A hybrid schema-conscious prompting method translates the input natural language to syntactically valid SQL. The schema mapping, the identification of tables through the Filtering Agent, attribute matching, and condition extraction were carried out to provide an accurate query formulation.

D. Layer Security and Privacy.

SQL injection and unauthorized modification of data is prevented by input validation, SQL keyword filtering and PDO prepared statements. Local execution through Ollama has the benefit of being able to deliver no schema metadata or query data to external servers.

E. Agile Iteration and Testing

Detailed testing is done on the query translation efficiency, response time, ability to tolerate ambiguous input and user satisfaction. Execution errors are captured by a continuous feedback loop and fed back to LLM refinement of prompts, thus enabling a self-correcting mechanism.

F. System Requirements

Minimum hardware: Intel Core i5 (8th Gen), 8 GB RAM, 10 GB SSD, and in-built graphics. Suggested: Intel Core i7, 16-32 GB RAM, 20 GB SSD, and NVIDIA CUDA. Stack software: XAMPP/LAMP (Apache, PHP ≥ 7.4 , MySQL), Ollama, Gemma2:9B, Tailwind CSS, and Git.

G. Hypothesis

The research hypothesis is that due to higher accuracy in more complicated queries - especially multi-table JOIN operations - Two-Step Agentic Workflow will be significantly more accurate than traditional single-step prompting, as it reduces irrelevant context and limits hallucinations.

5.IMPLEMENTATION

A. Frontend

JavaScript (ES6+) offers real-time querying and dynamic rendering of the DOM using the Fetch API that offers asynchronous communication with the backend. Tailwind CSS is a utility-based framework that allows quick, responsive UI development that works across desktop, tablet, and mobile screens.

B. Backend

PHP provides a secure interaction between the user interface and database and guarantees that there is no direct client-database interaction. PDO (PHP Data Objects) offers database abstraction, MySQL, PostgreSQL and SQLite, and implements prepared statements to prevent SQL injection attacks.

C. AI Integration

The Gemma2:9B model was implemented locally on the Ollama framework, avoiding external API calls. The model is given structured prompts that include query given by the user and metadata about the relevant schema and generates syntactically and semantically correct SQL output. Local execution ensures that the sensitive schema information does not go outside the internal network of the organization.

D. API Communication

RESTful APIs are used to communicate between the front-end and the backend using the HTTP GET/POST requests. All the data are communicated using JSON, with the benefit of being lightweight, platform-neutral, and keeping a strict separation between system layers.

E. Security Mechanisms

The query validation engine blocks malicious SQL keywords (DROP, DELETE, TRUNCATE and ALTER) prior to execution. The PHP layer, which sanitizes input, prevents injection of malicious payload. The role-based access control is offered as an optional enterprise extension.

6.RESULTS AND DISCUSSION

A. Experimental Setup

The system was installed on a local XAMPP/LAMP server with Intel Core i7, 16 GB RAM, and the Gemma2:9B model using it via Ollama. The testing was done against a normalized relational database with the following tables Users, Products, Orders, Order items and Sales and 200 test queries that were divided into three levels of complexity.

B. Sample Output Results

Table 2. Sample System Outputs

User Query (Natural Language)	Generated SQL	Execution Status
Show all employees	SELECT * FROM employees	Success
List sales from last month	SELECT * FROM sales WHERE date >= DATE_SUB(NOW(), INTERVAL 1 MONTH);	Success
Show top 5 products by revenue	SELECT product_name, SUM(revenue) AS total FROM sales GROUP BY product_name ORDER BY total DESC LIMIT five	Success

Get customers with no orders	SELECT * FROM customers c LEFT JOIN orders o ON c.id = o.user_id WHERE o.id IS NULL;	Success
------------------------------	--	---------

C. Performance Analysis

Table 3. Accuracy and Response Time by Query Complexity

Query Complexity Level	Accuracy Range	Avg. Response Time
Simple (single-table)	95–100%	0.8–1.2 s
Intermediate (filter, sort, aggregate)	85–95%	1.2–2.0 s
Complex (multi-table JOIN, nested)	75–90%	2.0–3.0 s

D. Discussion

The Two-Step Agentic Workflow demonstrated measurable superiority over single-step prompting, particularly for complex queries involving multiple tables. By first narrowing down the relevant tables before generation, this approach reduces hallucinated column names and incorrect function usage. Local LLM deployment via Ollama confirmed sub-3-second response times even on mid-range hardware, validating the system's real-time usability.

Limitations include reduced accuracy on vague temporal terms (e.g., 'recent', 'best') and occasional failures on deeply nested subqueries. Databases with hundreds of tables may require further optimization of the Filtering Agent prompt to remain within the model context window. These limitations provide clear directions for future research.

7. APPLICATIONS

The NLDH system has broad applicability across domains in which data accessibility and efficiency are critical.

- **Business Intelligence:** Enables managers to generate reports and analyze KPIs without SQL expertise, reducing their dependency on IT teams.
- **Healthcare:** Medical staff can retrieve patient records and generate compliance reports locally, preserving HIPAA compliance via on-premises deployment.
- **Finance and Banking:** Supports transaction analysis, fraud detection, and regulatory reporting queries without requiring DBA involvement.
- **Education:** Serves as a teaching aid for demonstrating real-time SQL generation from natural language, making database concepts accessible to students.
- **E-commerce and Retail:** Facilitates quick analysis of inventory, customer behavior, and sales trends for strategic planning.
- **Government and Public Sector:** Simplifies data access for administrative officials managing large citizen-facing data sets.
- **SMEs:** Empowers small organizations without dedicated technical teams to extract actionable data insights efficiently.

8. CONCLUSION AND FUTURE SCOPE

The Natural Language Database Helper successfully demonstrates a solution that bridges the gap between human language and relational database systems. The integration of the Gemma2:9B Large Language Model via Ollama, combined with the Two-Step Agentic Workflow, achieves high query accuracy, particularly for simple and intermediate queries, while maintaining complete data privacy through local execution. The decoupled PHP/JavaScript architecture ensures scalability, security, and ease-of-maintenance.

Experimental results confirm the hypothesis that multi-step context filtering reduces hallucinations and improves JOIN accuracy compared with single-step prompting. The system meets its success criteria of ease of use, efficiency, reliability, and data privacy, and represents a significant step towards intelligent conversational database interfaces.

Future Scope

- Multi-Language Support: Extend NL input to regional languages for global accessibility.
- Voice-based query input: Speech recognition is integrated for hands-free database interaction.
- Real-time data visualization: Graphical dashboards and charts are embedded for query result interpretation.
- NoSQL Database Support: Expand compatibility with MongoDB and other document-oriented stores.
- Adaptive learning: User correction feedback loops should be implemented to continuously improve LLM prompt accuracy.
- Mobile application: A dedicated mobile interface is developed for broader deployment scenarios.
- Enhanced Security: Multi-factor authentication and audit logging should be added for enterprise-level deployment.

ACKNOWLEDGMENT

The authors express profound gratitude to their guide, Prof. Samina Anjum for her valuable guidance and encouragement throughout this project. They also thank Prof. Nazish Khan (Head of Department) and Prof. Dr. K.S. Zakiuddin (Principal), Anjuman College of Engineering & Technology, Nagpur, for providing necessary facilities. Thanks, are also extended to all faculty, staff, and family members who contributed directly or indirectly to the successful completion of this work.

REFERENCES

- [1] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.
- [2] L. Dong et al., "A Survey on Large Language Models for Natural Language to SQL," IEEE Trans. Knowl. Data Eng., vol. 37, no. 1, pp. 112–130, Jan. 2025.
- [3] C. J. Date, An Introduction to Database Systems, 8th ed. Boston, MA, USA: Addison-Wesley; 2004.
- [4] N. Hampp and K. Mirylenka, "Leveraging Large Language Models for NL to SQL Conversion," in Proc. Workshop on ML for Language (MML), IBM Research, Jun. 2024.
- [5] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Upper Saddle River, NJ, USA: Prentice Hall; 2020.
- [6] T. Yu et al., "Spider: A Large-Scale Cross-Domain Semantic Parsing and Text-to-SQL Benchmark," in Proc. EMNLP, 2018.
- [7] B. Wang et al., "RAT-SQL: Relation-Aware Schema Encoding for Text-to-SQL," in Proc. ACL, 2020.
- [8] J. Li et al., "ResSQL: Decoupling the Schema Linking and Skeleton Parsing for Text-to-SQL," in Proc. AAAI, 2023.
- [9] OpenAI, "GPT-4 Technical Report," OpenAI Research.[Online].Available:<https://openai.com/research/gpt-4> (accessed Apr. 2026).
- [10] Oracle, "MySQL 8.0 Reference Manual," MySQL Documentation.[Online].Available:<https://dev.mysql.com/doc/refman/8.0/en/>(accessed Apr. 2026).
- [11] The PHP Group, "PHP 8.0 Release Announcement." [Online].Available:<https://www.php.net/releases/8.0/en.php> (accessed Apr. 2026).
- [12] Tailwind Labs, "Tailwind CSS Documentation." [Online].Available:<https://tailwindcss.com/docs> (accessed Apr. 2026).