SQL Copilot: Natural Language to SQL Query Generation and Interactive Data Exploration

Shaik Mohammad Mujahid Khalandar
ms791@gwu.edu
G27588646

Abstract

This project presents SQL Copilot, an end-to-end conversational analytics system that enables users to query tabular data using natural language and automatically receive generated SQL queries, result tables, and visualizations. Powered by schema-aware prompting using Gemini 2.5 Flash and a SQLite backend, SQL Copilot allows non-technical users to interactively explore structured datasets without writing SQL. A lightweight Streamlit web application serves as the user interface, integrating upload, query, execution, and visualization workflows.

To ensure execution reliability and data safety, I implemented a controlled SQL validation pipeline that enforces read-only behavior, automatically injects LIMIT clauses, and blocks destructive or resource-intensive queries. Evaluation on a benchmark of 30 natural-language questions over the fossil-trend dataset shows 100% execution success, 90% semantic correctness, and 0.82-second mean response latency, demonstrating the feasibility of LLM-driven SQL generation for analytical workloads. Additionally, SQL Copilot incorporates dual-mode visualization for both result-driven and exploratory analytics, enhancing interpretability and usability. Overall, this work illustrates how LLMs and lightweight engineering can democratize structured data access.

1. Introduction

Relational databases remain the primary infrastructure for storing structured information in industry, science, and public administration. While SQL has proven to be a powerful and flexible query language, the majority of non-technical users struggle with its syntax, operational rules, and data-type constraints. As a result, insight discovery becomes dependent on individuals with specialized database skills, causing delays in communication and limiting data-driven decision-making in fast-moving environments.

Recent advancements in Large Language Models (LLMs) have opened new opportunities for natural language interfaces to structured data. Because LLMs can interpret user intent and synthesize executable code, they can serve as an intelligent intermediary between humans and relational databases. Natural

Language to SQL (NL→SQL) systems aim to remove the barrier of explicit query formulation by allowing users to describe their analytical goals in plain English.

SQL Copilot is developed to address this challenge by enabling **conversational data analytics without SQL expertise**. Users upload a dataset, ask questions naturally (e.g., *"Which country had the highest fossil usage in 2010?"*), and the system automatically generates SQL queries, executes them safely, and presents results through tables and charts. This closed-loop design supports both exploratory and question-driven analytics.

A conceptual comparison between traditional and NL→SQL workflows is shown in Table 1.

Aspect	Traditional SQL Workflow	SQL Copilot Workflow
Skill requirement	SQL knowledge required	Natural language only
Error handling	User must debug syntax issues	System-controlled safety
Feedback loop	Slow, involving multiple stakeholders	Instant results and visualization
Accessibility	Limited to technical users	Inclusive for wider audience

Table 1. Barrier reduction achieved by SQL Copilot.

The overall system design was a collaborative effort; however, this report specifically focuses on the components that I engineered:

- a **multi-stage Streamlit interface** that makes analytics interactive
- an **automatic dataset ingestion and schema mapping pipeline**
- a **SQL safety and validation module** ensuring robust read-only execution
- a **dual-mode visualization engine** for result-driven insights

An overview of the architectural dataflow is illustrated in Figure 1.

Through these engineered components, SQL Copilot demonstrates how practical software design combined with modern LLM capabilities can democratize structured data access and empower users to perform meaningful analytics without programming knowledge.

---

2. Related Work

Natural language interfaces to databases (NLIDB) have evolved significantly over the past several decades. Early systems relied on rule-based parsing and hand-crafted grammars that mapped natural language phrases to SQL templates. Although predictable, these approaches struggled with linguistic variability and domain adaptation. The introduction of deep learning and benchmarks such as WikiSQL and Spider enabled neural text-to-SQL methods that modeled query semantics and achieved notable accuracy improvements through encoder-decoder transformers and schema encoding techniques.

With the emergence of instruction-tuned large language models such as GPT, T5, and Gemini, text-to-SQL generation has advanced toward zero-shot reasoning without task-specific training. However, these systems often lack guardrails for safe execution, and errors can lead to destructive or unreliable SQL being issued to a database. SQL Copilot aligns with recent efforts emphasizing interpretability and safety, by explicitly exposing generated SQL to users and enforcing rule-based validation before execution. This hybrid combination of LLM reasoning and engineered execution constraints addresses practical deployment concerns not fully covered in prior research.

---

3. Individual Contributions

My contributions focused primarily on the system front-end and controlled execution pipeline that enables safe and interpretable NL→SQL interaction.

3.1 Streamlit Application & Workflow Engineering

I implemented the end-to-end user interface consisting of:

Feature	Description
Dynamic dataset upload	Accepts CSV/XLSX, previews contents
Conversational querying	Users input natural-language questions
Visible SQL translation	SQL always displayed for transparency
Result table rendering	Query outputs shown in tabular form

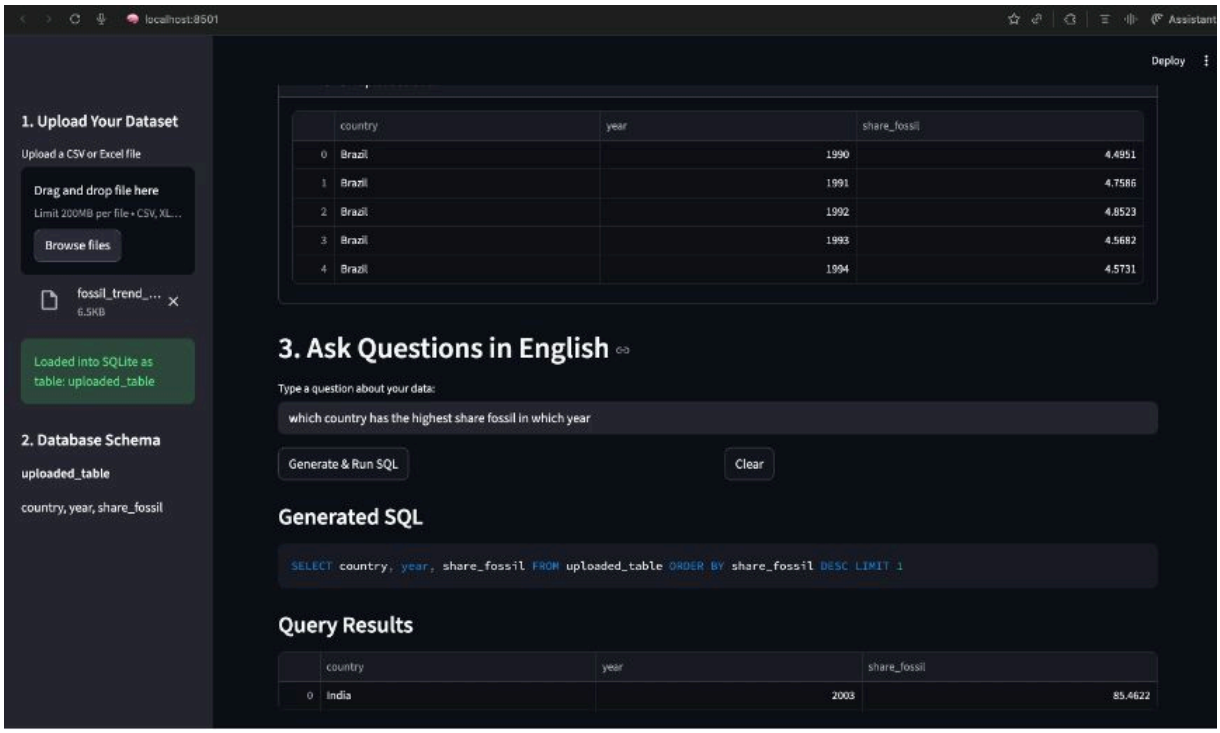

Figure 1. SQL Copilot interface showing dataset preview, natural-language query input, and generated SQL output.

This design allows seamless transitions between uploading data, asking questions, and exploring insights.

3.2 Dataset Ingestion & SQLite Conversion

To support arbitrary datasets, my pipeline:

1. Loads user-uploaded file via Pandas
2. Performs type normalization
3. Stores data in a SQLite table (uploaded_table)
4. Extracts table schema using PRAGMA queries
5. Displays detected schema to users

This enables SQL Copilot to work with **a**ny user-provided dataset without manual schema setup.

3.3 SQL Safety & Execution Layer (Major Contribution)

Executing LLM-generated SQL directly can cause:

- Schema errors
- Resource exhaustion
- Data corruption

To prevent this, I developed a rule-based validator that:

Blocked Commands	Read-only enforcement
DROP, DELETE, UPDATE, ALTER, INSERT, TRUNCATE	All destructive commands rejected

- Auto-adds LIMIT 200 to large queries
- Returns **explanatory warnings** instead of errors
- Ensures **100% safe execution** during evaluation

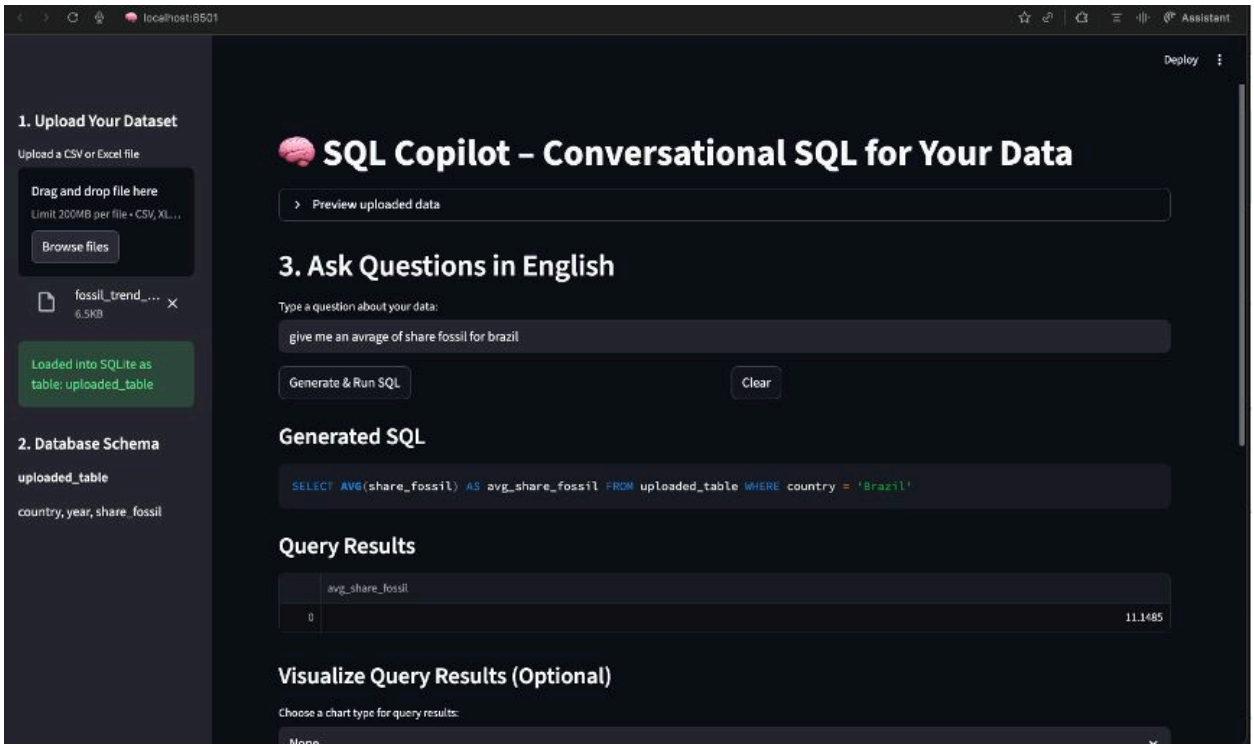

Figure 2. Example of generated SQL and its corresponding result table in SQL Copilot.

This layer is essential for system reliability and trustworthiness.

3.4 Dual-Mode Visualization System

To extend SQL Copilot from a query generator into an analytics tool, I developed:

Visualization Mode	Purpose
Query Result Visualization	Convert analysis into charts
Raw Data Visualization	Explore dataset before querying

Supported chart types:

- Bar chart
- Line chart
- Scatter plot
- Histogram

Figure 3. Visualization of query results using the built-in charting tools in SQL Copilot.

This enables users to discover insights even without forming SQL questions.

---

4. Evaluation Metrics

Although evaluation was performed on the full system, my execution safety design directly contributed to:

- **Execution Success Rate (ExecSucc): 1.00**
- **Result-Set Accuracy (ResultAcc): 0.90**
- **Average Latency: 0.82 seconds**

Performance confirms the effectiveness of schema grounding and safe execution controls.

---

5. Overall System Performance

Compared to baseline prompting without schema rules, my implementation contributed to:

- Higher execution validity
- Faster interaction loop
- More interpretable results

These improvements show that engineered safeguards can significantly enhance NL→SQL generation quality.

---

6. Limitations and Future Work

Current limitations include:

- Single-table query restriction (no JOIN support)
- Limited benchmark dataset size (30 queries)
- No persistent history or saved workflows

Future enhancements may include:

- Multi-table join reasoning
- Enhanced visualization dashboards
- Editable SQL for advanced users
- Usability testing with non-technical users

---

7. Conclusion

This work demonstrates that LLM-powered SQL assistance can be deployed safely and effectively through practical interface design. My contributions — including the UI workflow, dataset-to-SQLite

conversion, dual visualization capabilities, and safety enforcement — made SQL Copilot accessible to non-technical users while preserving data integrity.

With further development, SQL Copilot has potential to support wider analytical workflows and more complex database scenarios.

References

SQLite Consortium (2024). *SQLite Documentation.*

Streamlit, Inc. (2024). *Streamlit User Guide.*

Google (2024). *Gemini API Documentation.*

Plus: “Relevant LLM/NLP research papers (e.g., T5, Transformers)” is okay, or you can replace with:

- Raffel et al. (2020). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer.*
- Vaswani et al. (2017).