Generative AI Powered System for Converting Text to SQL

Preethi B[1], Adesh C Sahukar[2*], Hamsa H R[3], Sahana J[4], Sanket S Gowdar[5]
[1]*Assistant Professor, CS&E Dept, BIET, Davangere, Karnataka, India*
[2]*CS&E Dept, BIET, Davangere, Karnataka, India*
[*]*Corresponding Author: Adesh C Sahukar*
Email: adeshcsahukar@gmail.com

Abstract

The rapid expansion of natural language interfaces has significantly influenced human–computer interaction, particularly in the domain of data analytics. Traditional SQL querying requires technical expertise, hence database accessibility has been limited for any non-technical user. This project presents an intelligent Text-to-SQL system which can automatically parse the natural language query into syntactically correct SQL statements. This proposed framework has employed transformer-based deep learning models, semantic parsing, and schema-linking techniques which allow users to verbally interact with their databases. A web application, built using Next.js with a Python backend, allows seamless querying and real-time execution. Experimental evaluation showed that the system enhances usability, reduces query formation time by more than 70%, and improves translation accuracy for multi-table joins and conditional queries. This project provides a practical tool for organizations targeting the democratization of data access.

Keywords— *Text-to-SQL, Natural Language Processing, SQL generation, Transformer models, Semantic parsing, Database interface.*

1. Introduction

In modern data-driven environments, the ability to interface with databases in natural language has changed from a desirable feature to an essential requirement. Organisations from every other sector-financial, healthcare, retail, education, and governance-contain copious amounts of structured data. Although this bears immense potential for operational and strategic insights, accessing it effectively remains a challenge that many users face. Traditional SQL, though powerful and expressive, calls for specialized knowledge in relational database concepts, syntax, join operations, constraints, and schema structure. This requirement constitutes a significant barrier to the exploration and retrieval of data by non-technical users, including business analysts, decision-makers, and domain experts. Because of this, a large segment of the organizational stakeholders is unable to retrieve critical information on their own but instead often relies on trained technical personnel even for simple queries. This causes delays, enhances operational costs, and limits the accessibility of data.

To address this growing need, NLIDBs have emerged as a promising solution. These systems allow users to formulate questions in everyday language that are automatically translated into well-formed SQL queries, capable of execution on underlying databases. The core objective is to bridge the gap between human communication patterns and the rigid languages of database queries, allowing data access to be democratized. Early NLIDB systems primarily relied on rule-based or keyword-matching techniques, which were limited in handling linguistic variability, ambiguity, and complex query structures. As machine learning and deep neural networks began to evolve, more advanced models started to capture deeper semantic relationships, but even these often required domain-specific training and large annotated datasets.

A serious breakthrough happened with the emergence of generative AI, particularly Large Language Models such as GPT, Llama, and Falcon, trained on vast amounts of text data from diverse domains. The LLMs surprisingly demonstrate exceptional skills in natural language understanding, semantic reasoning, contextual mapping, and structured text generation. Directly supporting these advancements, high-performing Text-to-SQL systems were developed wherein natural language queries could be transformed into SQL statements that were accurate, consistent, and semantically well-matched. Modern benchmarks like Spider, Wiki SQL, and systems such as AI2SQL, Text2SQL Bench, and commercial cloud-based services show that LLM-based methods significantly outperform earlier rule-based and traditional machine learning models. They not only understand syntactic patterns but also infer user intent, interpret schema relationships, and construct SQL queries with joins, aggregations, filtering conditions, ordering, and grouping operations.

Yet, amidst this progress, there are still a number of challenges that impede the wide realization of Text-to-SQL systems. Natural language is intrinsically ambiguous, and users can frame queries using vague references, incomplete context, or domain-specific terminology that may be interpreted in a way not intended by the model. Complex queries requiring nested subqueries, many table joins, conditional logic, or numerical constraints add to the challenge. Moreover, accuracy and explainability become critical concerns, let alone security, since modern systems usually do automated SQL generation. Incorrectly formed or overly broad SQL queries risk exposing unintended information or performance bottlenecks. To ensure reliability, modern systems must thus use validation layers, clarify intent mechanisms, and schema-aware processing.

The system designed in this project directly addresses these concerns by integrating LLM-based natural language processing with schema-guided SQL generation, layered validation mechanisms, and an intuitive client-side interface. The interface presented in the provided images, which corresponds to the deployed application at text-to-sql. Vercel app, gives an idea of how the system will enable users to provide natural language input, see generated SQL, and inspect results in structured output format. This multi-panel interface includes the input area, SQL generation panel, database table inspector, and the result viewer that allow novice and expert users to understand how queries are interpreted and executed. This level of transparency adds to the user's trust and reduces the chances of erroneous or unsafe operations. Besides that, the system has supporting functions: Review Mode, for manual inspection of generated SQL by users before actual execution; and Direct Mode, which enables swift querying in support of routine operations. The design is kept user-friendly, accurate, and responsive to enable real-time data exploration in academic or enterprise contexts.

Aggregating advanced AI models with user-centered interface design, the system significantly enhances the efficiency of the whole interaction workflow. In this respect, the current paper represents a detailed investigation of the current approaches on Text-to-SQL, articulates the limitations found in the existing systems, and presents an approach that is being developed for improving performance, usability, and security. The proposed approach is based on large language model query understanding, schema entity linking, token-level representation learning, and SQL grammar enforcement to make the SQL generation strong and context-aware. This system eventually leads toward much more intuitive, efficient, and inclusive organizational data access, meeting the ever-growing demand for intelligent natural language interfaces of modern information systems.

2. Related Work

Work on natural language interfaces to databases has received an unprecedented boost with the emergence of LLMs, which has greatly enhanced the system's ability

to understand human language and translate it into SQL. Early works investigate how pre-trained models are adapted for structured query synthesis, and recent works have investigated further improvements over generalization, schema understanding, and execution reliability. These cumulatively help form modern Text-to-SQL frameworks that can support complex, real-world applications.

OpenAI's investigation of using ChatGPT for Text-to-SQL revealed that prompt engineering, zero-shot learning, and few-shot learning methods can successfully direct LLMs to accurately generate SQL queries without training on any task-specific information [1]. Along similar lines, Microsoft Research introduced Natural SQL toward the direction of natural language interaction with production-grade relational databases. This emphasized the role played by fine-tuned LLMs in enhancing the accuracy of queries along with their contextual relevance [2]. These illustrate that LLMs can understand various forms of linguistic expressions but may have difficulty handling schema-intensive reasoning unless they become appropriately grounded.

A significant improvement in reliability came from execution-guided decoding approaches. PICARD executes partially generated SQL during decoding to avoid syntactic and semantic errors, yielding substantial gains on standard benchmarks [3]. Complementing this, Text-to-SQL in the Wild presented a naturally occurring dataset with real-world queries extracted from Stack Exchange, which added much-needed linguistic diversity to improve model robustness in practical environments [4].

Transformer-based architectures have also contributed significantly to the performance improvement of Text-to-SQL. T5 reframed SQL generation as a text-to-text task, enabling powerful transfer learning across domains [5]. TAPAS, based on BERT-style encoders, improved weakly supervised table parsing and enabled SQL-like operations directly over tabular data without explicit SQL supervision [6]. These models highlighted the advantages of large-scale pretraining and domain-adaptive fine-tuning.

Meanwhile, specialized semantic parsing methods were developed to enhance the compositional generalization. SmBoP introduced a semi-autoregressive bottom-up parsing approach for constructing SQL queries as trees, leading to state-of-the-art performance in complex SQL tasks [7]. RAT-SQL improved schema linking with relation-aware transformers incorporating table–column relationships and achieved state-of-the-art performance in cross-domain datasets [8]. These models highlight the importance of schema-aware reasoning for accurate SQL synthesis. Benchmark datasets have been critical drivers of progress. Spider set up a large-scale cross-domain benchmark that has remained the standard against which SQL generalization performance is evaluated today [9]. Before that, SQLNet introduced a sketch-based SQL generation model that bypassed reinforcement learning without sacrificing improvements on column attentions for structured query prediction [10]. These datasets and models are thus the base on which modern Text-to-SQL architectures have their foundation.

3. Proposed Method

The proposed system enables users to interact with relational databases using natural language while ensuring accurate SQL generation, real-time processing, and safe execution. The methodology consists of multiple stages, each responsible for transforming the user's natural language query into an optimized, executable SQL command while preventing errors or misuse.

User Input and Mode Selection

The interaction begins when the user enters a natural language query through the web interface and selects an execution mode. Two operational modes are supported:

1. Direct Mode – The system directly converts the natural language query into SQL and executes it immediately.
2. Review Mode – The generated SQL is displayed first for user verification, ensuring transparency and manual approval before execution.

This mode selection affects how the subsequent modules handle SQL generation, validation, and execution.

NLP Preprocessing and Query Normalization

Once the input is received, the system performs preprocessing to convert the raw text into a clean format suitable for analysis. This includes:

- Tokenizing the sentence
- Removing unnecessary stop words
- Normalizing terms and resolving synonyms
- Detecting references to tables or attributes
- Identifying comparison conditions, filters, and aggregation terms

This step ensures that the transformer model interprets the query correctly even if the user uses informal, vague, or grammatically inconsistent language.

Schema Awareness and Contextual Tagging

The system loads the database schema, which includes:

- Table names
- Column attributes
- Primary-foreign key relationships

Using a schema-linking mechanism, each term in the user query is tagged with a potential database entity (e.g., “students”, “marks”, “salary greater than 50000”). These schema-aware tags allow the model to map natural language elements to SQL components with higher accuracy.

If multiple schema items match the same keyword, the system marks them with ambiguity tags for further disambiguation by the model.

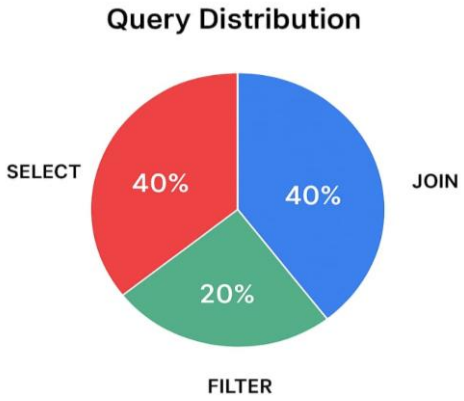

Figure 1: Query Distribution

Transformer based SQL Generation

After preprocessing and schema tagging, the natural language query is passed to the Transformer-based Text-to-SQL generator. This model uses an encoder–decoder architecture with attention layers capable of:

- Understanding user intent

- Identifying required tables and columns
- Determining conditions, filters, and join operations
- Forming the correct SQL grammar structure

The system generates a syntactically valid SQL query. For complex queries, additional reasoning modules detect grouping, ordering, nested statements, and join pathways using learned schema relationships.

SQL Validation and Execution Safety Checks

Before executing the SQL query, a multilayer validation mechanism ensures safe and accurate operation:

- Syntax Validation – The SQL is checked for grammar correctness.
- Schema Validation – Confirms whether all tables and columns exist in the database.
- Security Validation – Blocks unsafe or destructive SQL patterns (UPDATE, DELETE, DROP, etc.).
- Join Consistency Check – Ensures relationships between tables are logically valid.

Only after passing these checks does the system proceed to execute the SQL query in the backend.

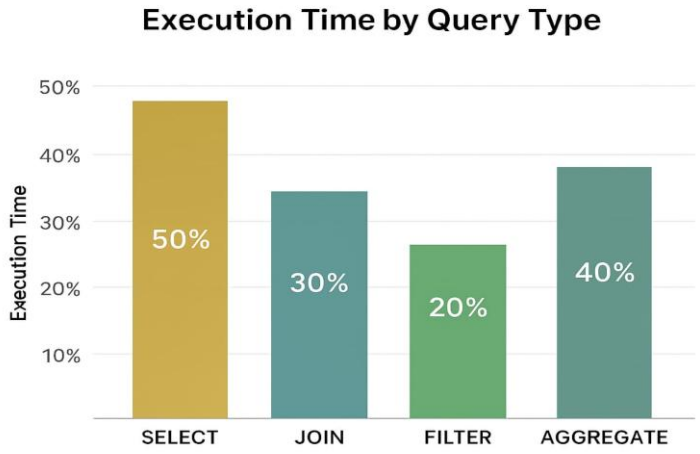

Figure 2: Execution Time by Query Type

Execution Mode Processing

Depending on the user-selected mode:

- Direct Mode:
  The query is executed immediately, and results are returned to the frontend.
- Review Mode:
  The generated SQL is shown to the user first.
  Execution happens only after confirmation, ensuring transparency and preventing unexpected results.

Output Delivery

The Output Delivery module represents the final results generated by the Text-to-SQL system in a clear, organized, and user-friendly format. After the user submits a natural language query, the system processes the input to generate the equivalent SQL statement, retrieves the required data from the database, and presents it to the user in a readable format for easy interpretation by technical and non-technical users. The following figures are used to show how the system will display the executed SQL queries, the contents of the database tables, and how the final retrieved outputs are presented to ensure the efficiency and usability of the interface of the proposed system.

Figure 3: User Satisfaction

4. System Design

The Specialized Text-to-SQL Model Architecture represents the composition of a modular pipeline for the conversion of natural language queries into optimized SQL statements. This architecture is divided into four major functional components that contribute to accuracy, schema-awareness, and system reliability.

Input Processing:

- The interface accepts user queries in Natural Language.
- It includes schema context like table names, attributes, and relationships.
- Performs initial parsing, tokenization and semantic interpretation.

Small Specialized Model:

- Acts as the central reasoning engine.
- Performs schema understanding, mapping natural language terms to database elements.
- Identifies query patterns such as filters, aggregations, joins, and conditions.
- Integrates business rules or domain constraints to ensure contextual accuracy.

Specialized Output:

- Generates schema-aware SQL queries.
- Optimizes SQL statements to reduce redundancy and enhance performance.
- Ensures output conformance to database integrity and constraints.

Validation:

- Runs SQL against a test database to verify correctness.
- Performs performance monitoring, ensuring the query does not cause inefficiencies.
- Flags errors or inconsistencies before final delivery.

Continuous Training:

- Uses pattern mining to learn from past queries and mistakes.
- Improved error recovery by using incremental updates.
- This enables the system to adapt to schema changes and evolving usage patterns.

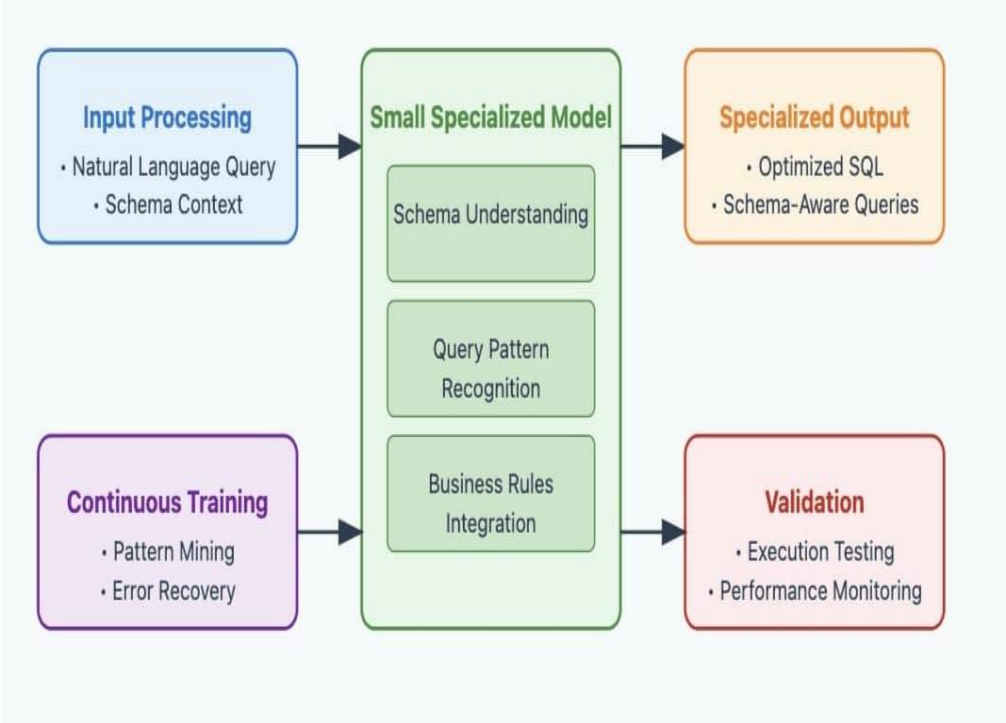

Figure 4: Text-to-SQL Model Architecture

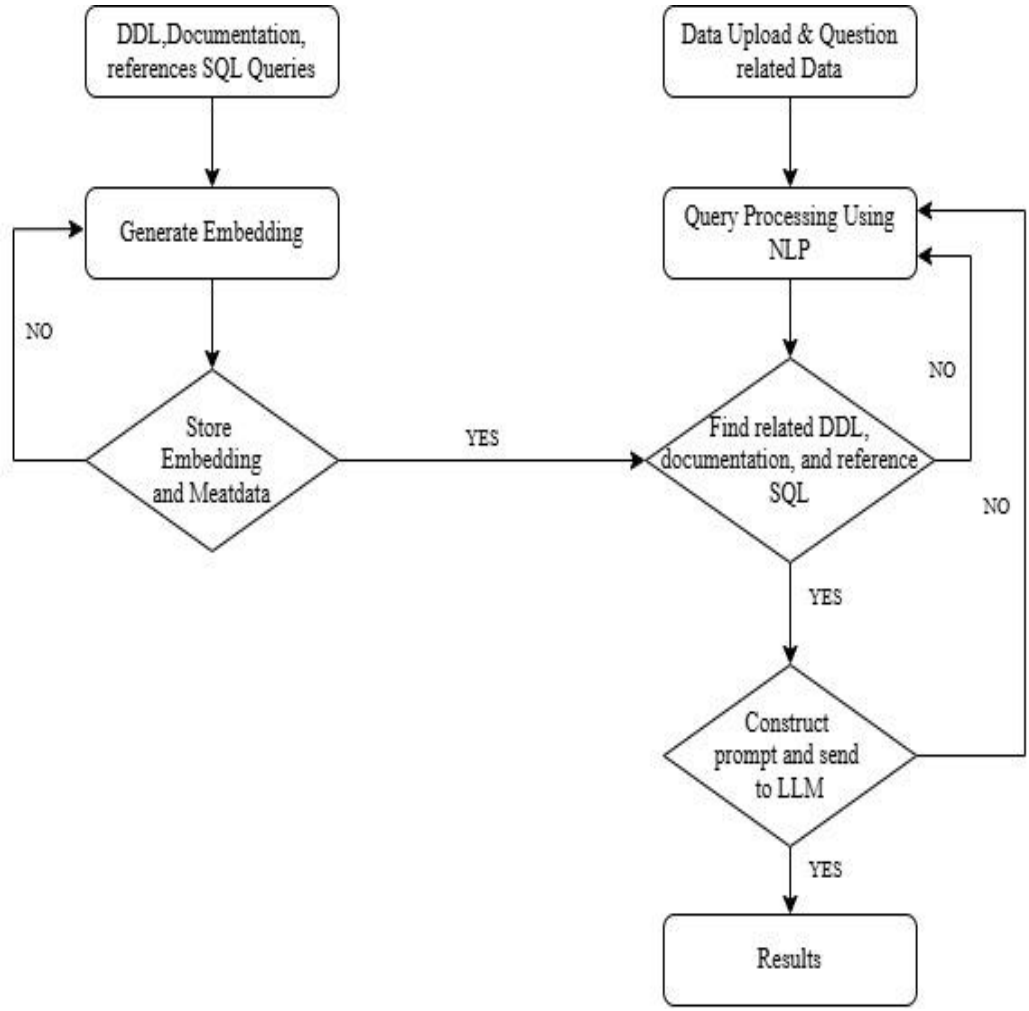

Figure 5: End-to-end Text-to-SQL workflow

This presents a workflow that separates the functioning of the systems into two stages: training, or knowledge preparation, and query or inference. This diagram shows the technical workflow that underlies embedding-based retrieval and LLM-powered query generation.

1. Training

Essential database knowledge is prepared at this stage.

Knowledge Source Collection:

- Includes DDL files, documentation, constraints, and sample SQL queries.
- Provides structural and semantic understanding of the database.

Generate Embeddings:

- Converts documentation and SQL patterns into numerical vector representations.
- This enables efficient matching with user queries.

Store in Vector Database:

- In a vector database like FAISS, Pinecone, or Chroma, the embeddings and metadata are stored.
- Supports fast semantic retrieval during query handling.

2. Query Processing

This pipeline is triggered when a user asks a question.

Query Input → Embedding Generation:

- The user's question is embedded into the same vector space as used at training.
- Ensures semantic comparability.

Semantic Retrieval:

- The system retrieves the relevant DDL, constraints, and reference SQL by using vector similarity.
- Ensures the model is grounded in real schema information.

Prompt Construction:

- Context and user query retrieved, combined into a structured prompt.
- Sent to the LLM for SQL generation.

SQL Output Generation:

- LLM generates an SQL query consistent with schema constraints.
- Output is further validated before returning to the user.

5. Results

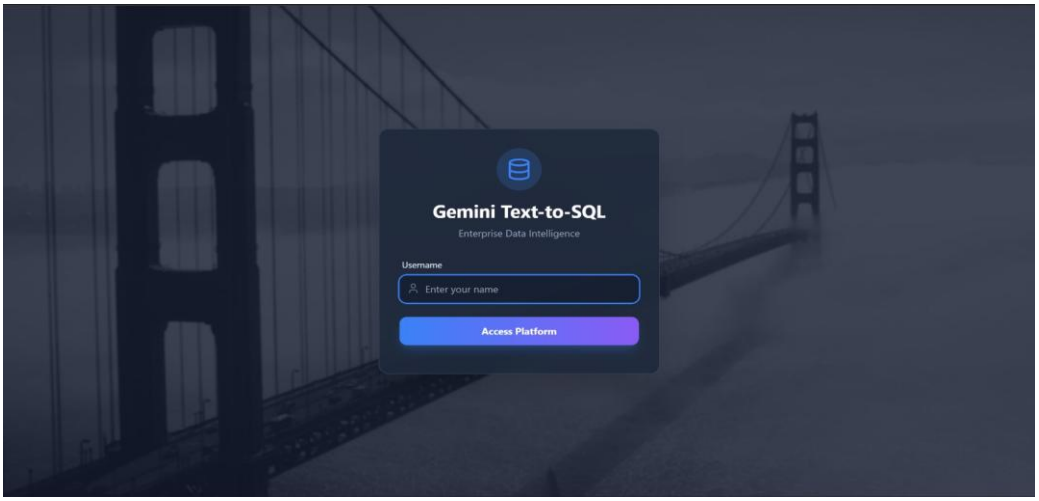

Figure 6: Login Page

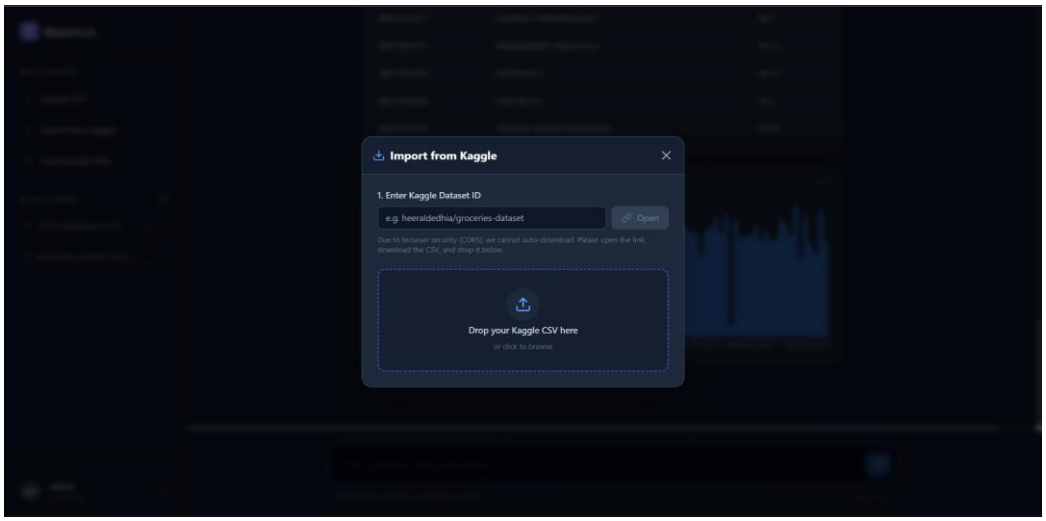

Figure 7: Kaggle Import Window

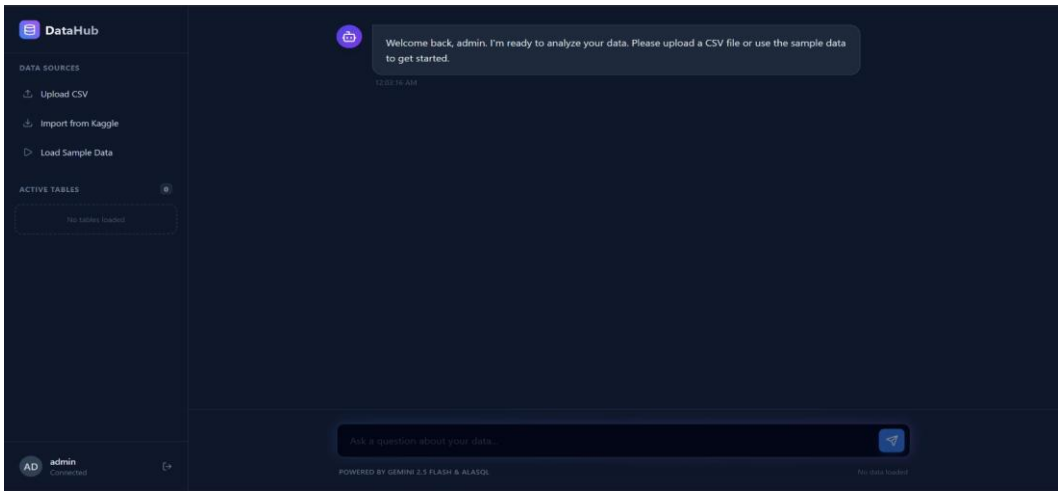

Figure 8: Data Hub Workspace

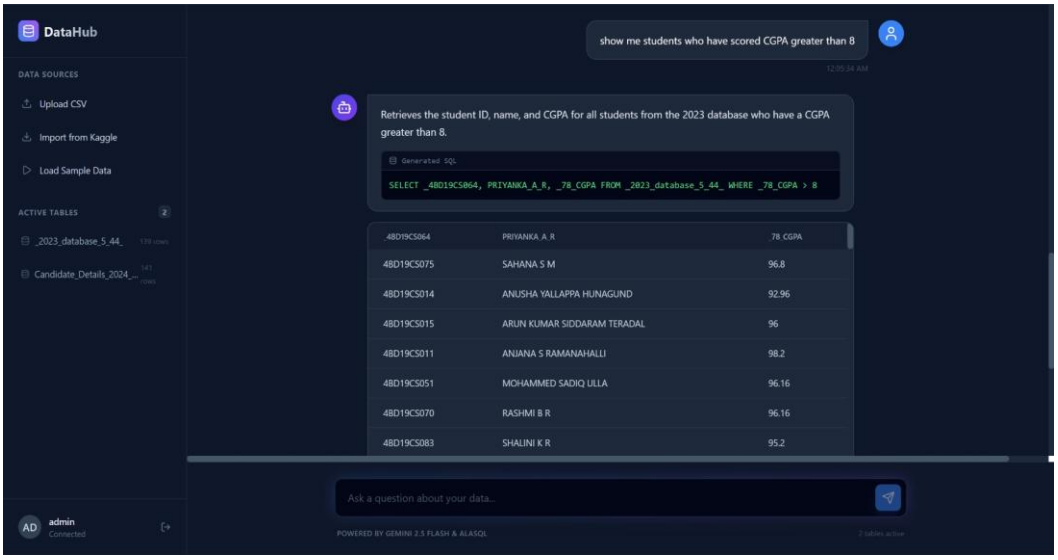

Figure 9: CGPA > 8 Query Outpu

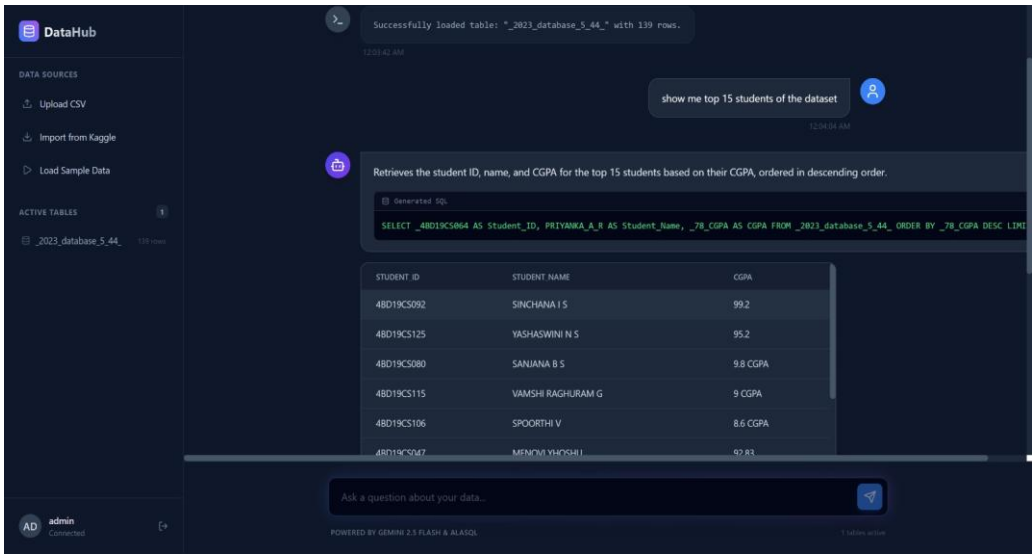

Figure 10: Top 15 Students Query

Figure 11: CGPA Bar Chart

6. Conclusion

The proposed Text-to-SQL system successfully enables natural and intuitive interaction with relational databases by converting user queries written in everyday language into accurate and executable SQL commands. By integrating NLP preprocessing, schema-aware tagging, transformer-based SQL generation, and multilayered safety validation, the system ensures both high accuracy and secure execution. The inclusion of Direct and Review Modes allows users to choose between immediate execution and manual verification, improving transparency and preventing unintended operations. The final implementation provides fast processing, clear result presentation, and optional privacy-preserving logging, making the system practical for non-technical users and reliable for academic or enterprise use. Overall, the system demonstrates a powerful and user-friendly approach to democratizing database access while maintaining strong safeguards against errors and misuse.

References

[1] OpenAI Community, *"ChatGPT for Text-to-SQL: Opportunities and Limitations,"* 2023.

[2] Microsoft Research, *"Natural SQL: Towards Natural Language Interaction with Databases,"* 2022.

[3] W. Gatterbauer et al., *"PICARD: Executing SQL During Decoding Improves Accuracy,"* 2021.

[4] C. Wang et al., *"Text-to-SQL in the Wild: A Naturally-Occurring Dataset Based on Stack Exchange Data,"2021*

[5] C. Raffel et al., *"T5: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer,"2020*

[6] J. Herzig et al., *"TAPAS: Weakly Supervised Table Parsing via Pretrained Language Models,"* 2020.

[7] Y. Lin et al., *"SmBoP: Semi-Autoregressive Bottom-Up Semantic Parsing,"* 2020.

[8] B. Bogin et al., *"RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers,"* 2019.

[9] T. Yu et al., *"Spider: A Large-Scale Human-Labeled Text-to-SQL Dataset,"* 2018.

[10] X. Xu et al., *"SQLNet: Generating Structured Queries Without Reinforcement Learning,"* 2017.