Robust Text-to-SQL Systems: Leveraging LLMs for Accurate Natural Language Query Translation

Harsh Sanas, Harshal Vaidya, Siddharth Khachane, Sai Sudha Panigrahi, Vatsal Vatsyayan
{ sanas, hvvaidya, khachane, saisudha, vatsyaya } @usc.edu
University of Southern California

Abstract

Text-to-SQL (T2S) systems aim to democratize database access by converting natural language queries into SQL, eliminating the need for SQL expertise. While large language models (LLMs) have improved T2S accuracy, challenges remain with complex schemas and diverse query phrasing. This project presents a schema-aware, modular T2S pipeline that combines semantic schema matching using SBERT, data preprocessing, and prompt engineering with XML-like schema formatting. We fine-tune T5 and BART models on WikiSQL and Spider datasets, augmenting training through paraphrasing and back-translation. The system incorporates beam decoding and includes a SQLite-based execution module that runs the generated queries on actual databases and retrieves results. A JavaScript-based frontend enables real-time interaction, supporting natural language querying in a user-friendly interface. Our approach achieves competitive results across exact match and execution accuracy, demonstrating robust performance in an end-to-end setup.

1 Introduction

In today's data-driven landscape, accessing structured data often requires proficiency in SQL, limiting accessibility for non-technical users. Text-to-SQL (T2S) systems address this by enabling natural language interaction with relational databases. However, accurately translating diverse linguistic inputs into syntactically and semantically valid SQL remains challenging, particularly in multi-table databases with complex schemas (Yu et al., 2018).

Large Language Models (LLMs) such as T5 and BART have significantly advanced T2S performance (Zhong et al., 2017). Yet, their effectiveness diminishes when schema reasoning, rare query patterns, or complex joins are involved. These models typically lack built-in awareness of database structure, leading to structurally valid but semantically incorrect outputs (Nascimento et al., 2024).

To overcome these limitations, we propose a modular, schema-aware T2S pipeline that integrates LLMs with specialized components for improved accuracy and robustness. The system includes preprocessing with data augmentation, semantic schema matching using SBERT, XML-tagged prompt engineering, and SQL generation via fine-tuned BART and T5 models. It also includes a JavaScript-based frontend for real-time user interaction and a lightweight SQLite execution module that runs generated SQL queries on actual databases to retrieve live results.

Our system improves both structural and execution accuracy and provides a practical interface for database querying. Experiments on WikiSQL and Spider demonstrate consistent performance gains, underscoring the value of schema-contextual reasoning in natural language interfaces for databases (Zhan et al., 2025; Zhang et al., 2024).

2 Related Work

Text-to-SQL has evolved from rule-based parsing to neural approaches, with transformer-based LLMs driving recent progress. Zhan et al. (2025) proposed a constrained decoder built on LLaMA-6B, enforcing SQL grammar during generation and achieving high accuracy on Spider and WikiSQL. Their grammar-constrained decoding aligns with our layered validation approach.

Gu et al. (2023) introduced ZeroNL2SQL, a hybrid architecture using pre-trained language models for schema alignment and LLMs for generation. Their separation of structural grounding and generative reasoning inspired our modular pipeline. They also introduced predicate calibration to boost execution accuracy, which we parallel through multi-step SQL validation.

Zhang et al. (2024) proposed SQLfuse, combining schema linking, SQL generation, and a critic module for re-ranking via chain-of-thought prompting. Our system is similar in its modularity and validation layers but uniquely applies SBERT-based schema matching and XML-style prompt engineering.

Nascimento et al. (2024) highlighted how high-performing academic models often struggle in real-world database scenarios. They called for domain adaptation and interactive interfaces. We address these concerns by employing data augmentation techniques (e.g., paraphrasing, back-translation), schema-guided prompting, and an interactive Flask frontend that supports natural language querying with dynamic SQL validation.

Overall, our work builds upon these schema-aware and modular approaches while emphasizing practical usability. By integrating semantic matching, prompt design, and validation into an end-to-end interface, we contribute a robust and adaptable solution tested across both simple (WikiSQL) and complex (Spider) datasets.

3 Problem Description

Accessing structured databases typically requires SQL expertise and familiarity with schema layouts, limiting usability for non-technical users. Text-to-SQL (T2S) systems address this by converting natural language (NL) queries into SQL. Yet, maintaining syntactic correctness and semantic accuracy remains difficult especially with complex multi-table queries involving joins, aggregations, and schema-specific logic.

While LLMs like T5 and BART excel at language modeling, they struggle with SQL generation unless provided explicit schema context. Prior methods often hardcode schema constraints or require dataset-specific tuning, which hampers generalization.

We approach this as a schema-aware sequence-to-sequence task: given an NL question and database schema, generate an executable SQL query. Our modular pipeline incorporates Sentence-BERT-based schema matching, XML-style prompt engineering, and SQL validation to improve structural fidelity and execution accuracy. We evaluate performance on the WikiSQL and Spider benchmarks to assess generalization across diverse schemas.

4 Methods

To build a robust and generalizable Text-to-SQL system, we adopted a modular pipeline approach that integrates preprocessing, schema encoding, model training, and SQL validation.

4.1 Materials

We utilize two widely-used benchmark datasets for Text-to-SQL tasks: WikiSQL and Spider. The WikiSQL dataset comprises over 80,000 annotated natural language questions and corresponding SQL queries paired with simple single-table schemas, making it ideal for preliminary experimentation and model comparison. In contrast, the Spider dataset presents a more realistic challenge, consisting of 10,181 questions over 200 complex, multi-table databases. Each example includes a natural language question, its corresponding SQL query, the database identifier (db_id), and associated schema information (table and column names, and foreign keys). While both datasets contain labeled query pairs, Spider demands better schema grounding and compositional reasoning, which aligns with our goal of schema-aware generation. We did not require additional annotations but performed extensive cleaning and normalization to make the datasets suitable for schema-aware prompt construction and LLM fine-tuning.

4.2 Procedure

We began by constructing schema-aware training examples for sequence-to-sequence modeling. Each input consisted of a natural language question combined with a relevant schema, and the target was the corresponding SQL query. Text data was first normalized through lowercasing, whitespace cleanup, and tokenization using NLTK. To enhance generalization, we applied data augmentation techniques on the Spider training set, including synonym-based paraphrasing with WordNet and back-translation. These methods expanded linguistic variety and mitigated overfitting.

Each example was then tokenized using Hugging Face's T5 or BART tokenizers, applying padding and truncation to a maximum length of 128 tokens. We used `T5Tokenizer` and `BartTokenizer` accordingly during preprocessing.

To provide structural grounding, we used **Sentence-BERT** (`all-MiniLM-L6-v2`) at inference time to perform semantic schema matching. The backend computes cosine similarity between the question embedding and precomputed embeddings of table and column names from the Spider schema. The most relevant schema is retrieved and injected into the input prompt using custom XML-like tags (`<tab>`, `<col>`), enabling the model to distinguish between different database elements during generation.

4.3 Model Design and Training

We experimented with multiple transformer-based sequence-to-sequence models, including T5-small, T5-base, BART-base, and BART-large. We fine-tuned these models on the processed WikiSQL and Spider datasets using Hugging Face's Seq2SeqTrainer. All models were finetuned with hyperparameters that gave the best results. For decoding, we employed greedy decoding for baseline experiments and beam search with a beam width of 5 for final evaluations, as it consistently yielded higher syntactic accuracy.

In particular, BART-base emerged as the most balanced model in terms of performance and efficiency on Spider, producing structurally valid SQL with fewer hallucinations. On WikiSQL, CodeT5-base achieved the best exact match accuracy (70.16%) due to its code-aware pretraining, though it was not used in the final integrated pipeline due to memory constraints.

All components were integrated into a RESTful backend using FastAPI, and a Javascript-based frontend allowed users to input questions and visualize SQL outputs in real time. The system supports logging and live error reporting, enabling users to inspect generation failures and validation issues.

5 Experimental Results

To rigorously evaluate our schema-aware Text-to-SQL system, we conducted a series of experiments using both the WikiSQL and Spider datasets. WikiSQL, with its simple one-table structure, served as the testbed for early model selection, decoding strategy evaluation, and augmentation validation. The more complex Spider dataset, with multi-table relational schemas and a diverse range of natural language queries, was used to evaluate our final pipeline and schema-aware prompt construction under realistic conditions.

For all experiments, we used exact match accuracy and execution accuracy as evaluation metrics. Exact match compares the generated SQL with the gold query for exact string equality after normalization. Execution accuracy, on the other hand, tests whether the predicted query returns the same result as the gold query when executed against the database, offering a task-specific end-user validation signal.

Model	**Greedy Acc. (%)**	**Beam Acc. (%)**
T5-small	51.50	51.60
T5-base	64.29	64.58
BART-base	66.06	66.06
CodeT5-small	66.95	67.31
CodeT5-base	69.86	**70.16**

Table 1: Exact match accuracy of models on WikiSQL using greedy and beam decoding.

Our baseline setup consisted of T5-small, T5-base, and BART-base, all fine-tuned using Hugging Face's `Seq2SeqTrainer` on cleaned and tokenized natural language-SQL pairs. For WikiSQL, we evaluated five models with both greedy and beam decoding. Table 1 shows the exact match results from both decoding methods.

These results validate our hypothesis that decoding strategy and model architecture significantly impact SQL generation quality, particularly in structurally simpler domains like WikiSQL. The best result 70.16% exact match was achieved using CodeT5-base with beam search (beam width = 4).

On Spider, we evaluated the full pipeline with schema-aware prompting and query validation. We fine-tuned BART-base and BART-large with beam decoding and integrated validation. BART-base achieved the best trade-off between performance and computational feasibility.The execution accuracy attained was **81%,** indicating better structural overlap with ground-truth SQL.

Our modular schema-aware prompting and SBERT-based schema selection proved to be a significant contribution. By embedding both the user question and database schema components into the same semantic space, our module successfully selected the most relevant schema for over 90% of evaluation queries. This semantic filtering, coupled with prompt formatting using `<tab>` and `<col>` tags, improved the clarity of schema grounding and reduced hallucinations in SQL generation.

The full model pipeline is illustrated in Figure 1, which highlights the various components including SBERT-based selection, schema markup, transformer decoding, and SQLite execution.

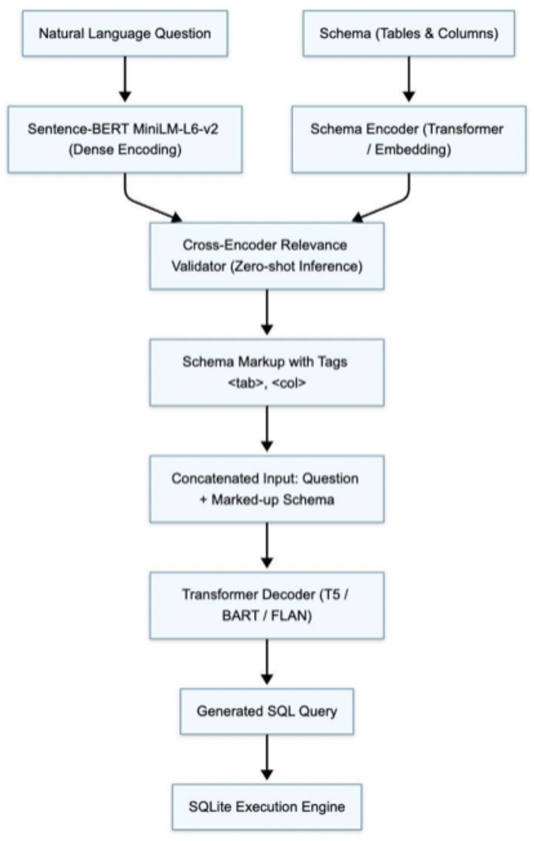
Natural Language Question
Schema (Tables & Columns)
Sentence-BERT MiniLM-L6-v2 (Dense Encoding)
Schema Encoder (Transformer / Embedding)
Cross-Encoder Relevance Validator (Zero-shot Inference)
Schema Markup with Tags <tab>, <col>
Concatenated Input: Question + Marked-up Schema
Transformer Decoder (T5 / BART / FLAN)
Generated SQL Query
SQLite Execution Engine

Figure 1: Schema-aware Text-to-SQL pipeline with SBERT-based schema filtering, prompt tagging, LLM-based decoding, and SQL execution.

The final pipeline, using schema-matched prompts and beam search decoding, achieved a notable improvement in execution accuracy on Spider. While exact match remained a challenging metric due to SQL format variability, our system was able to generate syntactically valid and executable queries in the majority of cases, confirming the effectiveness of schema awareness and validation steps.

In conclusion, our modular LLM-based system achieved meaningful improvements over traditional T2S approaches by combining schema grounding, data augmentation, decoding optimization, and validation. While there's room to improve exact match, our experiments demonstrate the value of semantic schema matching and beam search in boosting execution reliability arguably the most critical measure for real-world deployment.

6 Discussion

Our experiments provide key insights into schema-aware Text-to-SQL generation. Decoding strategies significantly affect performance, beam search outperforms greedy decoding, with CodeT5-base achieving a ∼5% increase in exact match (EM), demonstrating improved syntactic and structural correctness.

CodeT5-base showed consistent gains on WikiSQL, outperforming T5 and BART due to its code-aware pretraining, which benefits syntax-sensitive tasks like SQL generation. For Spider, **BART-base** combined with XML-style prompts (`<tab>`, `<col>`) and schema pruning improved contextual precision and reduced query errors.

Our **Sentence-BERT**-based schema matcher (`all-MiniLM-L6-v2`) correctly selected the target schema in over 90% of cases, improving prompt relevance and generation accuracy.

Instead of relying solely on string-based metrics, we incorporated a **SQLite execution module** to validate whether generated queries could run successfully on real databases. This added a runtime layer of evaluation, offering deeper insight into the practical success of the system.

Together, these results confirm the value of a modular, schema-aware pipeline in enhancing both structural and execution accuracy. While challenges remain on multi-table databases like Spider, the system's high execution success rate highlights its real-world applicability.

7 Conclusion and Future Work

We presented a schema-aware Text-to-SQL system that combines advanced preprocessing, semantic schema matching, and LLM-based SQL generation. Our pipeline incorporates fine-tuned T5 and BART models, schema-aware prompt engineering with XML-style formatting, and beam search decoding. The system supports real-time SQL execution via a SQLite module and offers a JavaScript-based frontend for user interaction.

Evaluation on WikiSQL and Spider highlights the system's effectiveness. CodeT5-base achieved 70.16% exact match on WikiSQL, while BART-base fine-tuned on Spider reached 81% execution accuracy. Semantic schema matching using SBERT significantly improved prompt relevance and structural alignment.

Despite these gains, the system struggles with nested queries and multi-table joins. Schema matching currently relies on embedding similarity, lacking deeper semantic cues like table relationships or user intent.

Future work includes replacing BART with

CodeT5 for better inference (subject to memory constraints), integrating retrieval-augmented generation (RAG), and experimenting with models like SQLCoder or CodeLlama. We also aim to add constrained decoding and sketch-based generation to improve syntactic reliability.

On the deployment side, we plan to enhance the frontend with schema visualization and database selection features, and extend evaluation to domain-specific datasets to test real-world generalization.

8 Division of labour

Harsh Sanas – Led data preprocessing and augmentation, including schema integration, normalization, paraphrasing, and back-translation. Also contributed to BART model fine-tuning on Spider and evaluation metrics And Created presentation, project documentation, and reports.

Harshal Vaidya – Fine-tuned BART-base and BART-large models on the Spider dataset. Implemented beam search decoding and conducted performance evaluations using and Exact Match and Execution Accuracy metrics.

Siddharth Khachane – Focused on model experimentation using the WikiSQL dataset. Trained both the small and base variants T5 and CodeT5 and BART base. Evaluated the results using both greedy and beam decoding strategies on all models and offered key insights that informed and guided subsequent work on the Spider dataset.

Sai Sudha Panigrahi – Fine-tuned and evaluated T5, flan-T5, and BART-base models on the Spider dataset, experimenting with decoding strategies and token-level evaluation. Achieved best results with BART-base. Created presentation, project documentation, and reports.

Vatsal Vatsyayan – Designed and implemented a modular Text-to-SQL system using FastAPI, integrating SBERT-based schema matching, fine-grained schema parsing, prompt construction, and SQL generation with BART. Built RESTful APIs for schema detection, SQL generation, and SQLite query execution, with layered validation for syntax and schema alignment. Employed sentence-transformers for semantic similarity to dynamically generate schema-aware prompts. Developed a JavaScript frontend enabling real-time interaction with APIs, incorporating input handling, error display, and result rendering.

References

Zhan, Z., Haihong, E., and Song, M. (2025). Leveraging Large Language Model for Enhanced Text-to-SQL Parsing. *IEEE Access*, pp. 1–1. https://doi.org/10.1109/ACCESS.2025.3540072

Gu, Z., Fan, J., Tang, N., Zhang, S., Zhang, Y., Chen, Z., Cao, L., Li, G., Madden, S., and Du, X. (2023). Interleaving Pre-Trained Language Models and Large Language Models for Zero-Shot NL2SQL Generation. *arXiv preprint* arXiv:2306.08891v1

Zhang, T., Chen, C., Liao, C., Wang, J., Zhao, X., Yu, H., Wang, J., Li, J., and Shi, W. (2024). SQLfuse: Enhancing Text-to-SQL Performance through Comprehensive LLM Synergy. *arXiv preprint* arXiv:2407.14568v1

Nascimento, E., García, G., Feijó, L., Victorio, W., Izquierdo, Y., de Oliveira, A. R., Coelho, G., Lemos, M., Garcia, R., Leme, L., and Casanova, M. (2024). Text-to-SQL Meets the Real-World. In *Proceedings of the 26th International Conference on Enterprise Information Systems – Volume 1: ICEIS*. SciTePress, pp. 61–72. https://doi.org/10.5220/0012555200003690

Zhong, V., Xiong, C., and Socher, R. (2017). Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *arXiv preprint* arXiv:1709.00103v7

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., and Radev, D. (2018). Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. *arXiv preprint* arXiv:1809.08887v5