Natural Language to SQL with Fine-Tuned T5 and Retrieval-Augmented Generation

Harish Padmanabhan [*], **Sabari Mathavan Ramasamy Balasubramanian**[*]
[1]Department of Data Science, Northeastern University, Boston, MA, USA
padmanabhan.h@northeastern.edu, ramasamybalasubram.s@northeastern.edu

GitHub Repository

Abstract

Natural Language to SQL (NL2SQL) systems enable users to access and query relational databases using plain English, eliminating the need to learn SQL syntax. However, in cross-domain settings where the target database is not specified, the challenge lies in both selecting the correct schema (*schema grounding*) and generating an executable query (*execution correctness*). We propose an NL2SQL framework that integrates a fully fine-tuned `T5-base` model with Retrieval-Augmented Generation (RAG) for automatic schema selection. Database schemas are encoded using a SentenceTransformer (`BAAI/bge-small-en-v1.5`) and stored in a FAISS index, enabling efficient retrieval of the most relevant schema at inference time. The retrieved schema is concatenated with the user's query and passed to the T5 model for SQL generation. Trained on the Spider dataset, our system achieves **31.13%** exact match accuracy, **44.96%** execution match accuracy, **63.34%** simple query execution match accuracy, and **78.53%** schema retrieval accuracy. These results demonstrate that coupling schema retrieval with generative modeling improves grounding, reduces hallucinations, and enhances execution correctness for unseen databases, offering a scalable approach for real-world, multi-database environments.

Introduction

Relational databases store and power a vast majority of enterprise, governmental, and public data systems, serving as the backbone for analytics, reporting, and decision-making across industries. While SQL remains the standard language for interacting with these databases, its steep learning curve and rigid syntax often limit access for non-technical users, creating a barrier between the data and those who could benefit from it most. Natural Language to SQL (NL2SQL) systems aim to bridge this gap by translating natural language queries into executable SQL statements, enabling intuitive, language-based access to structured data.

In real-world, cross-domain scenarios, NL2SQL presents two primary challenges. *Schema grounding* involves correctly identifying the relevant database and the specific tables and columns required to answer the query. *Execution correctness* requires generating SQL queries that not only follow proper syntax but also accurately capture the user's intent, producing correct and meaningful results. Both challenges are amplified in settings where the system encounters databases it has never seen before, and the user provides no explicit schema or database identifier.

Our work addresses these challenges in precisely such a setting: the user supplies only a natural language question, and the system must autonomously determine the correct schema and produce a valid, executable SQL query. To achieve this, we integrate a fully fine-tuned `T5-base` model for SQL generation with a Retrieval-Augmented Generation (RAG) step that uses dense sentence embeddings to automatically select the most relevant schema. By embedding and indexing all available database schemas with a SentenceTransformer and searching them via FAISS, our system dynamically retrieves the correct schema at inference time. This hybrid design improves schema grounding, reduces irrelevant context, and enhances execution accuracy, while remaining scalable to multi-database environments and adaptable to unseen domains without additional retraining.

Background

Spider Dataset

Our work is trained and evaluated on the Spider dataset, a large-scale, cross-domain benchmark designed to test generalization to unseen schemas. The dataset contains over 6,000 natural language questions paired with SQL queries across 133 databases spanning diverse domains such as business, academia, and government. Importantly, the training, validation, and test sets are split such that no database appears in more than one split.

Spider queries vary widely in complexity, from simple single-table lookups to nested aggregations, `GROUP BY` clauses, and multi-table joins. This diversity makes the dataset an ideal benchmark for assessing both schema generalization and execution correctness in NL2SQL systems.

Sequence-to-Sequence Models for NL2SQL

The Natural Language to SQL (NL2SQL) task can be formalized as a conditional sequence generation problem, where the input sequence consists of a natural language question (optionally accompanied by a database schema description), and the output sequence is the corresponding

[*]These authors contributed equally.

SQL query. Sequence-to-sequence (seq2seq) architectures, particularly those based on the Transformer model, have proven highly effective for this task due to their ability to capture long-range dependencies and model complex structural relationships between tokens.

The T5 (Text-to-Text Transfer Transformer) framework is particularly well-suited to NL2SQL because it treats all NLP problems in a uniform "text-to-text" format. In this paradigm, translating natural language into SQL becomes analogous to translating between human languages: the encoder processes the input question and schema, and the decoder generates the SQL query token-by-token. This approach enables leveraging pre-trained knowledge while adapting to SQL-specific syntax and semantics during finetuning.

Retrieval-Augmented Generation (RAG)

While powerful, pure seq2seq approaches face limitations in multi-database, cross-domain settings where relevant schema information may be absent or diluted in large inputs. Retrieval-Augmented Generation (RAG) addresses this by coupling a neural generator with an external retriever. Instead of relying solely on internal model parameters to store and recall schema information, RAG dynamically retrieves relevant documents (in our case, database schemas) from an external knowledge store at inference time.

In our system, database schemas are first transformed into plain-text descriptions capturing table names, columns, primary keys (PKs), and foreign key (FK) relationships. These descriptions are embedded into dense vector representations using the SentenceTransformer model `BAAI/bge-small-en-v1.5`. A FAISS L2 index enables efficient nearest-neighbor search over these embeddings. At query time, the system encodes the user's question into the same vector space, retrieves the most relevant schema, and conditions SQL generation on it. This retrieval step ensures that the generator operates with schema-specific context, improving grounding and reducing hallucination.

Related Work

Research in NL2SQL has evolved rapidly over the past decade, moving from rule-based and statistical methods to advanced neural architectures and large language models. These approaches vary in how they represent schemas, generate queries, and ensure syntactic and semantic correctness. Below is a summary of key contributions that have influenced our work:

- **Early Neural NL2SQL Models** – Seq2SQL (Zhong et al. 2017) and SQLNet (Xu et al. 2017) used sketch-based decoding to fill SQL slots, but struggled with unseen databases.
- **Schema-Aware Transformers** – RAT-SQL (Wang et al. 2020) introduced relation-aware self-attention for better schema linking, inspiring later models such as BRIDGE and SmBoP to handle complex queries.
- **Constrained Decoding** – PICARD (Scholak et al. 2021) enforced SQL syntax during generation, while execution-guided decoding (Wang et al. 2018) pruned invalid queries at runtime.
- **Industrial Systems** – Uber's QueryGPT combines LLM-based generation with execution validation for production NL2SQL, demonstrating how enterprise adaptation improves usability.
- **Schema Retrieval Approaches** – Prior work often used TF-IDF/BM25 keyword search; our approach leverages dense embeddings (`BGE-small-en-v1.5`) with FAISS for more robust semantic retrieval.

Our work builds upon these foundations by explicitly combining dense schema retrieval with a fine-tuned generative model. Unlike purely generative approaches, our pipeline decouples schema selection from SQL generation, reducing hallucination and improving execution correctness in multi-database, cross-domain settings.

Project Description

Our system translates natural language queries into executable SQL in a multi-database environment where the user does not specify the target database. It combines dense schema retrieval for automatic schema selection with fine-tuned T5-based SQL generation for query construction. This approach improves schema grounding, reduces irrelevant context, and increases execution correctness.

Schema Embedding

Each database schema is converted into a structured plain-text description containing table names, column names, data types, primary keys (PKs), and foreign key (FK) relationships. This schema text is embedded into a dense vector s_i using the SentenceTransformer model `BAAI/bge-small-en-v1.5`, which captures semantic meaning beyond keyword overlap.

Indexing

The schema embeddings s_i are stored in a FAISS L2 index, enabling fast nearest-neighbor search even with hundreds or thousands of schemas. This indexing step allows the retrieval stage to operate in milliseconds at inference time.

Schema Retrieval

A user's natural language query q is embedded into a vector q using the same embedding model. The system retrieves the schema with the smallest Euclidean distance to q:

$$S^* = \arg\min_{S_i} \|q - s_i\|_2$$

This ensures that even if the question uses synonyms or different phrasing from the schema, the most relevant database is still selected.

Prompt Construction

The retrieved schema S^* is concatenated with the user's question in the following format:

```
translate to SQL: <QUESTION>
<SCHEMA TEXT>
```

This format mirrors the style used during training, making the model more consistent at inference.

SQL Generation

The formatted prompt is passed to the fully fine-tuned `T5-base` model, which has been trained to produce syntactically correct and semantically accurate SQL. Beam search ($k = 5$) is used to explore multiple candidate outputs and choose the highest-scoring one.

Front-End (Web UI: Flask + HTML/JS)

Purpose: End-to-end, database-agnostic querying without SQL. Users can upload `.sqlite` databases, view schemas, ask questions, and get generated SQL (plus the exact model prompt used).

Stack: Flask (routes + JSON APIs), vanilla HTML/CSS/JS (no frameworks), SentenceTransformer + FAISS in the service layer, fine-tuned T5 for generation.

System Configuration

Component	Value
Base Model	T5-base (full fine-tuning)
Optimizer	AdamW
Learning Rate	5×10^{-5}
Warmup Steps	10% of total
Precision	FP16
Embedding Model	BGE-small-en-v1.5
Index Type	FAISS L2
Retrieval Top-k	1 schema

In summary, this modular architecture decouples retrieval from generation, allowing the model to always work with schema-relevant context. This design improves accuracy, reduces hallucinations, and makes the system highly scalable—adding a new database requires only computing and indexing its schema embedding, with no retraining required.

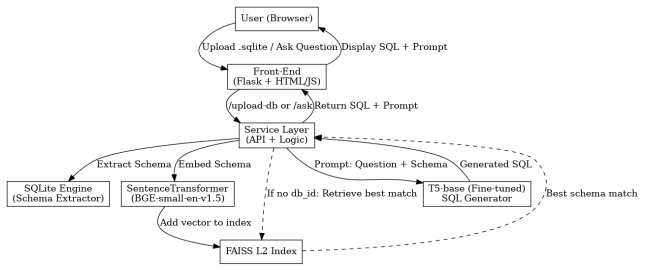

User (Browser)
Upload .sqlite / Ask Question
Display SQL + Prompt
Front-End (Flask + HTML/JS)
/upload-db or /ask
Return SQL + Prompt
Service Layer (API + Logic)
Extract Schema
Embed Schema
Prompt: Question + Schema
Generated SQL
SQLite Engine (Schema Extractor)
SentenceTransformer (BGE-small-en-v1.5)
If no db_id: Retrieve best match
T5-base (Fine-tuned) SQL Generator
Best schema match
Add vector to index
FAISS L2 Index

Figure 1: System design

Empirical Results

We evaluated our NL2SQL system on the Spider benchmark to measure the effect of integrating Retrieval-Augmented Generation (RAG) with a fully fine-tuned `T5-base` model. The metrics include Exact Match Accuracy (EM), Execution Match Accuracy (EMExec), and Schema Retrieval Accuracy. Results are also broken down by simple and complex queries to assess the strengths and weaknesses of the approach.

Training and Validation Loss

Epoch	Train Loss	Val Loss
1	2.5908	0.8172
2	0.7154	0.4072
3	0.4405	0.2958
4	0.3360	0.2359
5	0.2754	0.2069

Table 1: Training and validation loss across epochs.

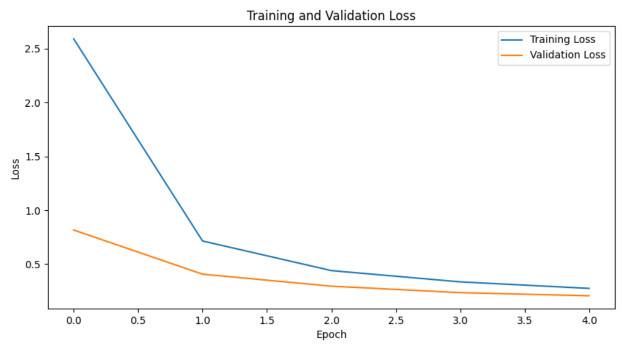


Figure 2: Train and Validation Losses

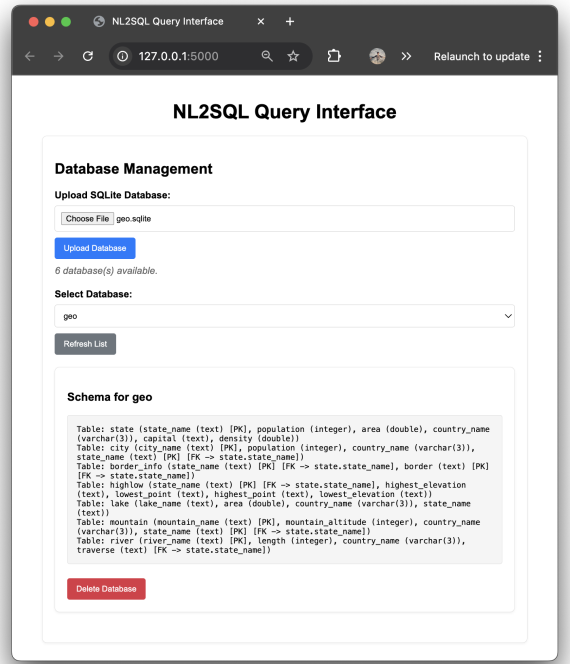
Test Performance (Spider Dataset)

Metric	Fine-tuned T5 + RAG
Exact Match Accuracy	31.13%
Execution Match Accuracy	44.96%
Simple Query Execution Match Accuracy	63.34%
Complex Query Execution Match Accuracy	24.62%
RAG DB Identification Accuracy	78.53%

Table 2: Test performance on the Spider benchmark.

User Interface

Below are screenshots of the user interface developed using Flask, HTML, and JavaScript. The interface allows users to upload, delete, and view the contents of a database provided as a .sqlite file. When a question is entered, it is sent to the RAG component to retrieve the most relevant schema, which is then passed to the model to generate the corresponding SQL query. The generated SQL syntax is subsequently displayed to the user through the same interface.



NL2SQL Query Interface

Database Management

Upload SQLite Database:

Choose File geo.sqlite

Upload Database

6 database(s) available.

Select Database:

geo

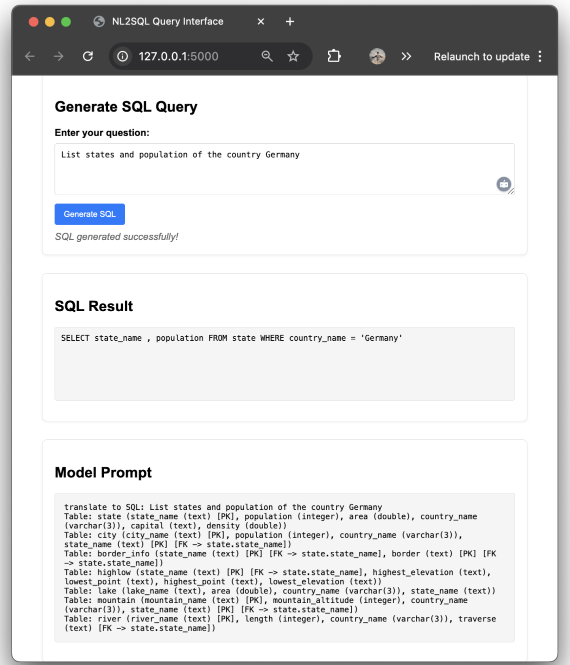
Refresh List

Schema for geo

```
Table: state (state_name (text) [PK], population (integer), area (double), country_name
(varchar(3)), capital (text), density (double))
Table: city (city_name (text) [PK], population (integer), country_name (varchar(3)),
state_name (text) [PK] [FK -> state.state_name])
Table: border_info (state_name (text) [PK] [FK -> state.state_name], border (text) [PK]
[FK -> state.state_name])
Table: highlow (state_name (text) [PK] [FK -> state.state_name], highest_elevation
(text), lowest_point (text), highest_point (text), lowest_elevation (text))
Table: lake (lake_name (text), area (double), country_name (varchar(3)), state_name
(text))
Table: mountain (mountain_name (text) [PK], mountain_altitude (integer), country_name
(varchar(3)), state_name (text) [PK] [FK -> state.state_name])
Table: river (river_name (text) [PK], length (integer), country_name (varchar(3)),
traverse (text) [FK -> state.state_name])
```

Delete Database

Figure 3: Upload, View, Delete DBs



Generate SQL Query

Enter your question:

List states and population of the country Germany

Generate SQL

SQL generated successfully!

SQL Result

```
SELECT state_name , population FROM state WHERE country_name = 'Germany'
```

Model Prompt

```
translate to SQL: List states and population of the country Germany
Table: state (state_name (text) [PK], population (integer), area (double), country_name
(varchar(3)), capital (text), density (double))
Table: city (city_name (text) [PK], population (integer), country_name (varchar(3)),
state_name (text) [PK] [FK -> state.state_name])
Table: border_info (state_name (text) [PK] [FK -> state.state_name], border (text) [PK] [FK
-> state.state_name])
Table: highlow (state_name (text) [PK] [FK -> state.state_name], highest_elevation (text),
lowest_point (text), highest_point (text), lowest_elevation (text))
Table: lake (lake_name (text), area (double), country_name (varchar(3)), state_name (text))
Table: mountain (mountain_name (text) [PK], mountain_altitude (integer), country_name
(varchar(3)), state_name (text) [PK] [FK -> state.state_name])
Table: river (river_name (text) [PK], length (integer), country_name (varchar(3)), traverse
(text) [FK -> state.state_name])
```

Figure 4: Question and SQL result

Analysis

The retrieval step consistently improved execution accuracy by ensuring the generator received only schema-relevant context. Gains were most pronounced for simple queries, where irrelevant schema noise was largely removed. Complex multi-join queries remain a challenge, likely due to input length constraints and the need for multi-hop reasoning.

Broader Implications

This work demonstrates how combining large language models with retrieval-based schema grounding can make relational databases accessible to non-technical users. By eliminating the need to manually select a database or write SQL, the system lowers the barrier for data access and can be applied across domains such as business intelligence, government open data portals, and education. Its modular design also enables easy adaptation to new databases without retraining, making it scalable for enterprise environments. However, broader deployment must address potential risks such as query misinterpretation, exposure of sensitive schema information, and ensuring generated SQL is both safe and accurate.

Future Work

- **Larger and More Specialized Models** – Explore models such as `Flan-T5-Large` or domain-specific LLMs to improve complex query handling.
- **Execution-Guided Generation** – Integrate runtime feedback during decoding to catch and correct invalid SQL before returning results.
- **Multi-Schema Queries** – Extend support to queries that require combining data from multiple databases or schemas.
- **Interactive Disambiguation** – Add a clarification step when multiple schema interpretations are possible.
- **Enhanced RAG Filtering** – Introduce a confidence threshold or re-ranking step to avoid incorrect schema retrievals.

Conclusion

We developed an end-to-end NL2SQL system that integrates fine-tuned T5 SQL generation with dense retrieval-based schema selection. Evaluated on the Spider benchmark, the approach achieved **31.13%** exact match accuracy, **44.96%** execution match accuracy, and strong schema retrieval accuracy (**78.53%**), with notable gains in simple query execution performance. The retrieval step effectively reduced hallucinations by ensuring the model worked only with relevant schema context. While complex multi-join queries remain challenging, this architecture offers a scalable and adaptable foundation for real-world, multi-database environments, bridging the gap between human language and structured database querying.

References

Zhong, V., Xiong, C., & Socher, R. (2017). Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, 342–351.

Xu, X., Liu, C., & Song, D. (2017). SQLNet: Generating Structured Queries from Natural Language Without Reinforcement Learning. *Proceedings of the VLDB Endowment*, 11(3), 238–251.

Wang, B., Shin, R., Liu, X., Polozov, O., & Richardson, M. (2020). RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 7567–7578.

Scholak, T., Schucher, N., & Bahdanau, D. (2021). PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 9895–9901.

Wang, C., Shin, R., Liu, X., Polozov, O., & Richardson, M. (2018). Execution-Guided Neural Program Decoding. *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 9133–9140.

Guo, J., Cao, C., Deng, X., He, S., Wu, H., & Wang, H. (2019). IRNet: Interpretable Reasoning Network for Text-to-SQL. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 5349–5358.

Lin, X. V., Socher, R., & Xiong, C. (2020). Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing. *Findings of the Association for Computational Linguistics: EMNLP 2020*, 4870–4888.

Rubin, O., & Berant, J. (2020). SmBoP: Semi-autoregressive Bottom-up Semantic Parsing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 8952–8965.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Riedel, S. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems (NeurIPS)*, 33, 9459–9474.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., ... & Radev, D. (2018). Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 3911–3921.