Natural Language to SQL with Fine-Tuned T5 and RAG

Sabari Mathavan Ramasamy Balasubramanian
Harish Padmanabhan

High-Level Overview

Goal: Enable users to query structured databases using plain English without knowing SQL.

Input: A natural language question (e.g., “What is the average salary of employees in Sales?”).

Output: An executable SQL query tailored to the correct database schema.

Key Components:

1. Fine-tuned T5-base model for SQL generation.
2. RAG with Sentence Transformers + FAISS to select the most relevant database schema.
3. Web interface for database upload, schema viewing, and query generation.

Technical Problem Formulation

Task Type: Text-to-SQL generation with database schema grounding.

Features/Inputs:

- Natural language question.
- Corresponding database schema (tables, columns, PK/FK info).

Output: SQL query string that can be executed on the database.

Dataset:

- Spider dataset—166 databases across multiple domains, with train/validation/test splits.
- Complex & simple SQL queries.

Problem Challenges:

- Schema linking—mapping natural language to correct tables & columns.
- Generalization – unseen databases in test time.
- Ambiguity—vague questions requiring correct table selection.

System Architecture

Model Fine-Tuning:

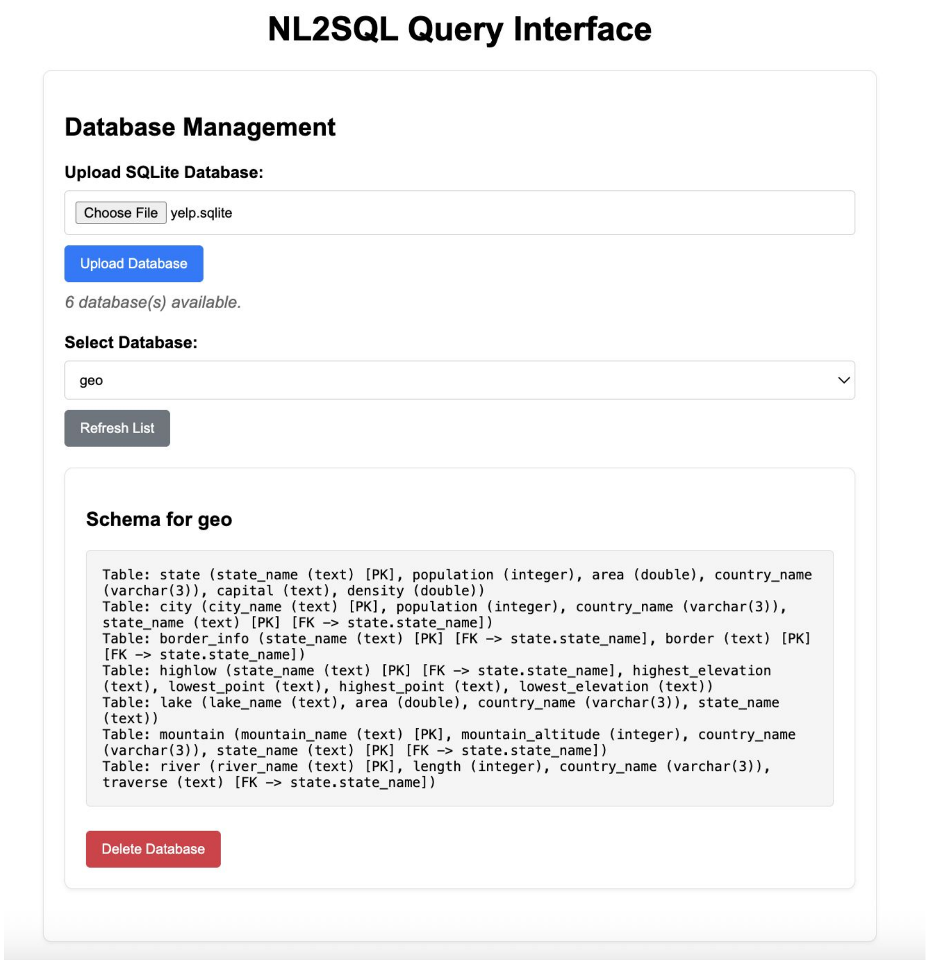
- Base model: T5-Base with gradient updates to all parameters
- Input format: “Translate to SQL: <QUESTION> \n <SCHEMA TEXT>”
- Target: Corresponding SQL query.
- Training parameters:
  - AdamW optimizer
  - learning rate 5e-5
  - linear scheduler with 10% of training steps as warmup
  - gradient accumulation
  - mixed precision
  - checkpoint for best models

Retrieval-Augmented Generation (RAG):

- Schema Embeddings: SentenceTransformer (BAAI/bge-small-en-v1.5) encodes DB schemas into dense vectors.
- Vector Search: FAISS L2 index retrieves the most relevant schema.
- Dynamic Retrieval: Auto-selects schema if DB is not specified in query.
- Generation: Retrieved schema + question → Fine-tuned T5 → SQL
- Advantages
  - Reduces hallucination
  - scales to new DBs without retraining.
  - The user doesn't need to pass schema (or connection between tables in the database) while giving input.

Web Interface (Flask + HTML, JS):

- Upload SQLite DBs, auto-extract schema, and store embeddings.
- Ask questions, get SQL & see prompt.
- View/delete DBs.



NL2SQL Query Interface

Database Management

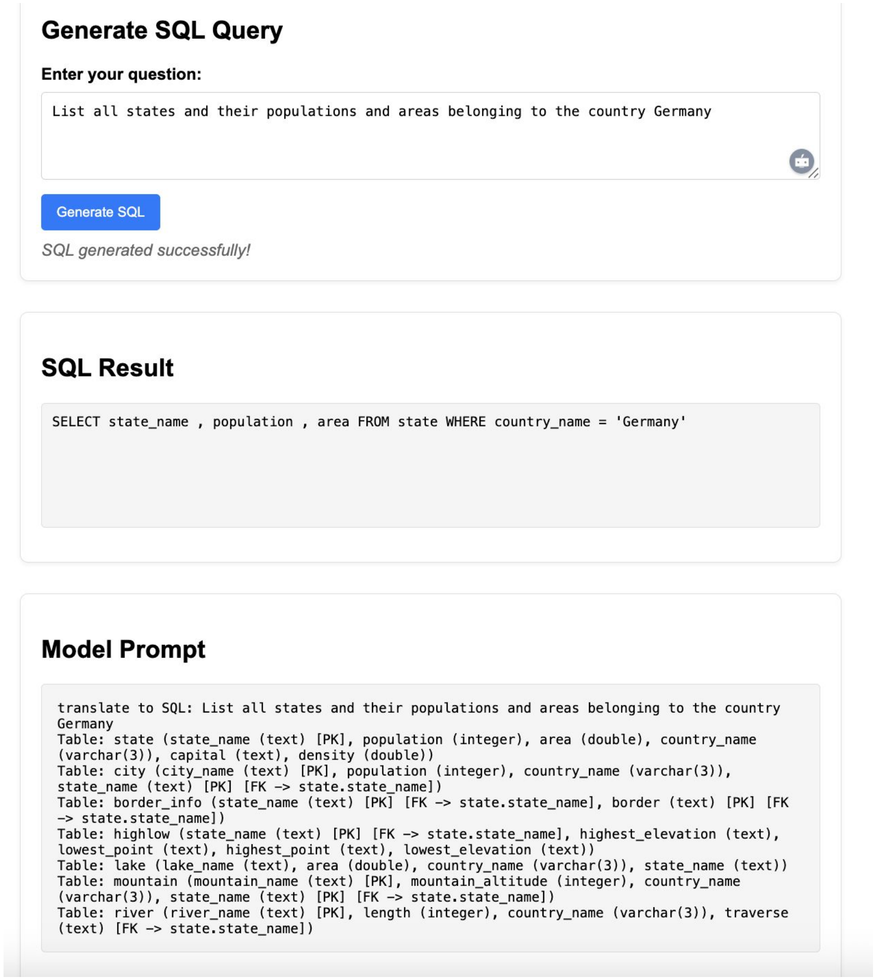
Upload SQLite Database:

6 database(s) available.

Select Database:

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])
```



Generate SQL Query

Enter your question:

```
List all states and their populations and areas belonging to the country Germany
```

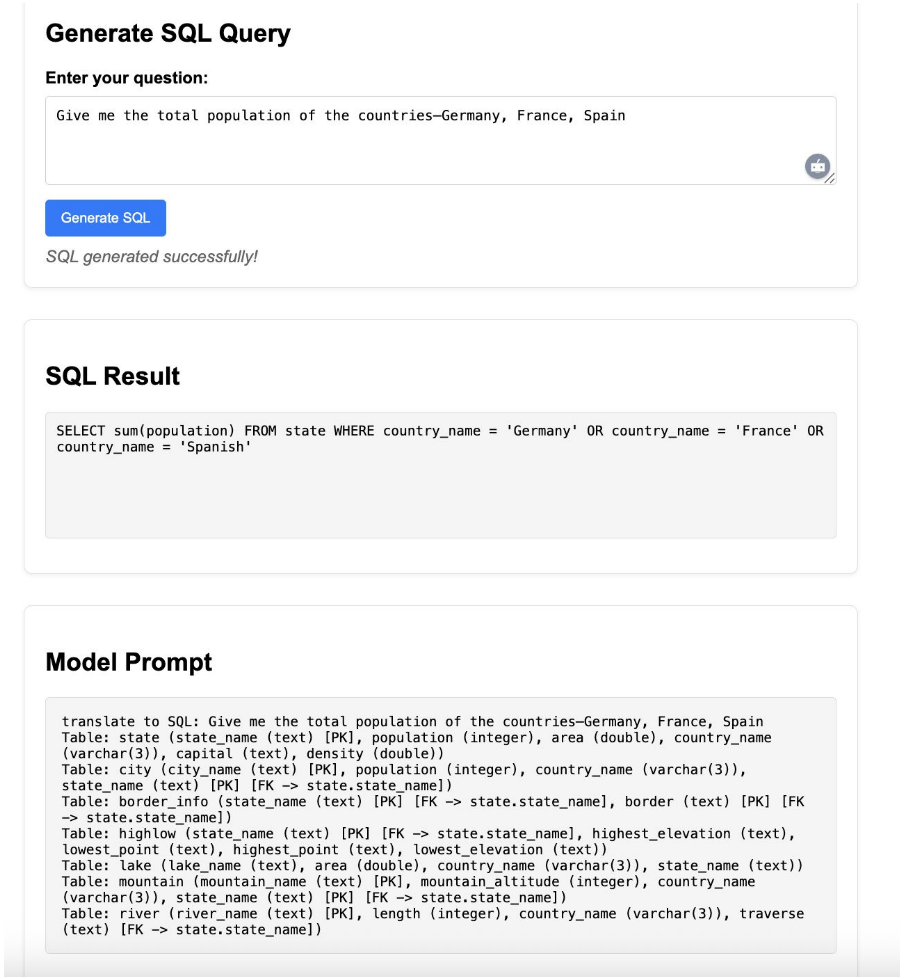
SQL generated successfully!

SQL Result

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

Model Prompt

```
translate to SQL: List all states and their populations and areas belonging to 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])
```



Generate SQL Query

Enter your question:

```
Give me the total population of the countries—Germany, France, Spain
```

SQL generated successfully!

SQL Result

```
SELECT sum(population) FROM state WHERE country_name = 'Germany' OR country_name = 'France' OR
country_name = 'Spanish'
```

Model Prompt

```
translate to SQL: Give me the total population of the countries—Germany, France, Spain
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])
```

Results & Evaluation

Baseline model: The baseline vanilla T5-base model is an encoder-decoder architecture, and this model does not output data in SQL format unless fine-tuned.

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%

Future Work & Conclusion

Future Work

- Boost Model Accuracy: Explore larger models (T5-Large or Flan-T5) and domain-specific pretraining to improve exact and execution match scores.
- Error Analysis & Debugging: Implement automated SQL correctness checks and semantic error detection before execution.
- Complex Query Handling: Complex queries still remain problematic.
- RAG Threshold: RAG retrieves the closest match, even if a proper DB is not found.

Conclusion

- We developed an end-to-end NL2SQL system combining fine-tuned T5 with RAG schema retrieval.
- Achieved 31.13% exact match accuracy and 44.96% execution match accuracy, with strong performance on simple queries (63.34%) and solid schema retrieval (78.53%).
- RAG significantly improved database selection, helping in identifying the appropriate database without the need for a schema to be passed as input.