Natural Language to SQL using LLMs

Rajiv Hemant Shah
Association for the Advancement of Artificial Intelligence
2275 East Bayshore Road, Suite 160
Palo Alto, California 94303

Abstract

I present a system that allows non-technical users to query databases using natural language. This system combines intent-aware prompt engineering and supervised finetuning on the Spider dataset to form an end-to-end solution for SQL query generation. The system utilizes Microsoft Phi 1.5 for intent extraction and a finetuned 'Codet5p' model for SQL generation. The model achieves a 30% execution match, indicating good performance in practical scenarios. I will discuss architectural design, technical challenges, evaluation, and future improvements in this report.

Introduction

Translating natural language to SQL (NL2SQL) is a longstanding task in NLP and a core component in many real-world analytical platforms enabling access to data for non-technical users. Traditional approaches rely on manually engineered grammar or rule-based systems, which fail to generalize and often require a level of technical understanding to query the system correctly.

Recent advances in large language models (LLMs) have shown promise for open-domain question answering and structured query generation. However, general LLMs often struggle with schema understanding, foreign key joins, and SQL-specific syntax. This project addresses these challenges by building an end-to-end system that augments LLM-based SQL generation with schema-aware prompt engineering and supervised fine-tuning.

I combined prompt engineering, schema-intent inference using a pre-trained LLM (Phi-1.5), and fine-tuning of the CodeT5p model on the Spider dataset to generate semantically correct SQL queries. I evaluated performance using two metrics - Exact Match and Execution Match. Incorporation of schema and intent information had significant impact on performance. The final system supports schema uploads, metadata storage, natural language interface, and query execution, offering a strong foundation for multi-turn conversational SQL agentic systems.

Background

Natural Language to SQL

NL2SQL refers to the task of converting natural language questions into structured SQL queries that can be executed against relational databases. This task is challenging as it requires a deep understanding of natural language semantics as well as database schema and SQL syntax. It becomes even more complex when we have multiple tables, joins, or aggregate functions involved. Since, the same output can be achieved using completely different SQL approaches, evaluation is also a challenge.

Seq2Seq Large Language Models

Pretrained sequence to sequence (Seq2Seq) transformer models are models that leverage the entire transformer architecture i.e. encoder and decoder. Since these models are already trained on natural language, they are generally capable of adapting to SQL generation by finetuning on domain specific datasets like **Spider** and **CoSQL**.

Retrieval-Augmented Generation(RAG)

RAG systems augment LLMs by retrieving relevant information and injecting this information into prompts dynamically. This allows systems to add only the relevant context to the input prompt helping in controlling input prompt lengths which especially important when working with smaller models.

Related Work

Natural Language to SQL systems aim to translate natural language to SQL queries enabling non technical users to interact with databases without requiring knowledge of SQL syntax or schema structure. This task is at the intersection of natural language understanding, semantic parsing, and structured query generation.

Early approaches were rule-based and had trouble generalizing beyond predefined input patterns. These systems relied on manually defined grammar which limited their adaptability and scalability.

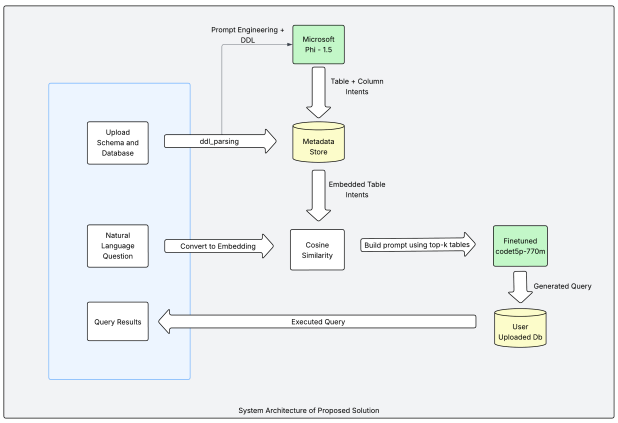
The introduction of deep learning methods, especially sequence to sequence models, marked a significant shift in NL2SQL solutions. Seq2Seq models enhanced with Attention mechanism, allowed systems to learn mappings from

natural language to SQL in a data driven manner. However, these models often struggled with generating syntactically valid SQL and understanding schema-specific constraints without explicit encoding.

Recent advancements in large language models have helped further enhance the performance of NL2SQL tasks. These models benefit from being trained on massive amounts of language data and instruction tuning to perform NL2SQL tasks with limited or even no task-specific training. These models show strong performance in zero-shot and few-shot settings, but often require prompt engineering and are sensitive to schema grounding.

My solution builds upon these developments by incorporating schema understanding through LLM-based intent inference and supervised fine-tuning using a lightweight transformer model (CodeT5p)

Project Description



The proposed system is composed of the following components:

- **Data Preprocessing:** I have used the Spider SQL Dataset which is processed and split into 90% training , 5% validation, and 5% test sets
- **Model Finetuning:** I have finetuned a Salesforce/codet5p-770m model using the preprocessed Spider dataset. This model generates SQL query from plaintext question combined with intent enriched table information.
- **Metadata Intent Inference:** I extract table and column intents using prompt-engineered examples and the Microsoft Phi-1.5 model. The outputs are then stored in a metadata table like SQLite
- **Natural Language Question processing:** The user question is converted into embeddings which is then used to find the most relevant tables for the question using cosine similarity and picking the top k matches
- **Metrics:** Traditional metrics like ROUGE and BLEU score have little relevance when evaluating SQL queries generated by the model. We therefore employ a more robust way of evaluation - **Exact Match** against gold sql as well as **Execution Match**

Data Preprocessing

To fine-tune the model for natural language to SQL translation, I have used the Spider dataset which is a widely adopted benchmark for this task. The dataset provides a collection of databases with complex, cross-domain schemas, along with paired natural language queries with corresponding SQL queries. The first step was to parse the provide schema details and create a consistent mapping from table to columns.

I then perform Intent augmentation as explained below. These table and column level intents are stored in the metadata store along with the parsed schema structures like table name, column name , and data types.

I now convert each sample into a structured prompt that includes:

- Database name
- Tables Schema with Intent
- User question
- The expected SQL query - target during training process

This dataset is then split into 90% - 05% - 05% for training, validation, and testing.

Model Fine-Tuning

Due to limited resources, I started with codet5-small, codet5-base, and eventually the codet5p-770m. This is a pretrained encoder-decoder transformer model optimized for code generation tasks.

This model was fine-tuned using the prompt-gold sql pairs prepared in the above step. I additionally ran experiments with these models where I also used CoSQL dataset to sequentially finetune a model i.e. finetuning a base model on spider data set and then fine tuning this new model using the CoSQL dataset. However, due to compute resource limitations, training proved extremely difficult. I therefore stuck only to Spider dataset and used the codet5p-770m model for final finetuning.

The finetuning was carried out by using Huggingface Trainer with a batch size of 8 per device and for 5 epochs. Training is tracked using MLFlow to capture and monitor evaluation loss, exact match, and execution match metrics over epochs.

Metadata Intent Inference

Understanding the semantic context of the database is critical for accurate SQL generation. To address this, I did the following:

- I used a Microsoft/Phi-1.5 model along with a custom few-shot prompt containing annotated examples of table and column intent descriptions.
- After a little post processing the model was able to generate fairly accurate table and column intents.

- These intents along with column name, table name, and data type are stored in a metadata store to support fast semantic retrieval and dynamic schema construction during inference time.

Natural Language Question Processing

Once the user enters a query, we use a semantic search strategy to identify relevant tables:

- The question is encoded using all-MiniLM-L6-v2, a SentenceTransformer model.
- All table-level intents in the metadata are embedded similarly.
- **Cosine similarity** is computed between the query and each table intent.
- The *top-k* matching tables are selected as contextual support for prompt construction.

This approach is domain-agnostic and allows the system to scale across databases and schema configurations while controlling the input prompt token length as required for the query generation model.

Metrics

Standard text generation metrics like **BLEU** and **ROUGE** are not appropriate for SQL generation tasks because:

- They penalize minor lexical variations that do not affect execution.
- They fail to capture logical equivalence or result-set equivalence.

Instead I used **Exact Match** and **Execution Match** as metrics for training and evaluation.

Exact Match: Compares the model-generated SQL to the gold-standard SQL string. While useful, it is strict and overly sensitive to reordering and formatting.

Execution Match: Executes both the predicted and gold SQL against the same database and compares the result sets. This is a more robust and semantically grounded metric.

My model achieved 20% Exact Match and 30% Execution Match on the test dataset.

Empirical Results

I observed that the Execution Match metric is higher than Exact Match, which indicates that the model can often generate semantically correct queries even when the syntax differs from the ground truth.

Epoch	Training Loss	Validation Loss	Execution Match	Exact Match	Invalid Sql
1	0.013900	0.018471	0.240000	0.205714	0.154286
2	0.008500	0.016957	0.248571	0.220000	0.140000
3	0.006600	0.015505	0.248571	0.228571	0.180000
4	0.005300	0.015258	0.254286	0.234286	0.177143
5	0.005200	0.014858	0.254286	0.234286	0.217143

Figure 1: Model Finetuning Metrics

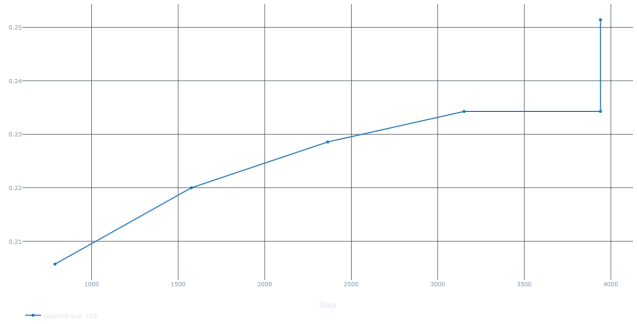


Figure 2: Exact Match during Training and Testing

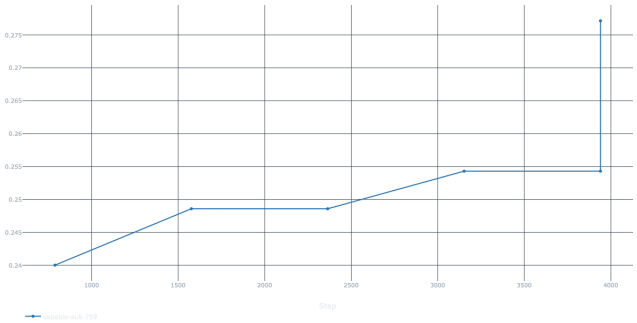


Figure 3: Execution Match during Training and Testing

Notably, inclusion of table and column intents improved performance compared to a baseline version that used schema names alone. In qualitative analysis, I found that the model struggled most with complex joins and multi-table reasoning, likely due to context length limitations (model size) as well as incorrect understanding of columns. These issues could have been fixed possibly with few-shot prompting. However, due to resource constraints, I was limited to a smaller model which could not accept longer input sequences.

I also tracked training and evaluation using MLflow and exported all model predictions for error analysis. Manual inspection revealed that errors often stemmed from:

- Failure to understand Schema/ Intent correctly
- Incorrect JOIN conditions
- Incomplete SQL generation

Broader Implications

This project contributes toward democratizing access to structured data and enabling non-technical users to retrieve insights from SQL databases. While this can prove to be quite helpful, care must be taken when deploying such systems to production in order to adhere to access control mechanisms and sensitive data protection. Since, generated queries can some times produce incorrect results, software level guardrails must be established as well as perhaps a human in the loop approach would make the system more reliable.

Integrating a LLM judge could perhaps address some of the challenges like evaluating the generated query before execution, interacting with user and prompting for more information in case generated query/ results seem ambiguous, etc.

Conclusion and Future Direction

I was able to build an end to end NL-to-SQL assistant using a mix of prompt engineering, metadata extraction, and model fine-tuning. I believe the incorporation of semantic table and column level intent to enable RAG search as well as enriching the prompts during finetuning and inference was novel and significantly improved performance.

While the overall performance was not as good as I expected, this project provided significant insights and enabled me to focus more on prompting, and data preprocessing than simply picking the largest available model. Working with limited resources allowed me to explore new ways both successful and unsuccessful to extract the best out of the system. Future work for this project may include:

- Adding an LLM Judge module for self-verification and query clarification
- Sequential finetuning on CoSQL data
- Supporting multi-turn conversations using dialogue state tracking
- Leveraging larger models for handling larger input size and enabling few-shot performance improvements

This project helped me understand the power of combining LLMs with structured metadata and the importance of grounding LLM outputs in execution. For future students, I would recommend focusing on prompt design and evaluation early and modularizing your architecture for easier iteration.

Github link

https://github.com/RajivShah1798/NL2SQL

References

@miscZhongEtAl2017, author = Zhong, Victor and Xiong, Caiming and Socher, Richard, title = Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning, year = 2017, eprint = 1709.00103, archivePrefix = arXiv, primaryClass = cs.CL, url = https://arxiv.org/pdf/1709.00103

@miscYuEtAl2018, author = Yu, Tao and Zhang, Rui and Yang, Kai and Yasunaga, Michihiro and Wang, Dongxu and Li, Zifan and Ma, James and Li, Irene and Yao, Qingning and Roman, Shanelle and Zhang, Zilin and Er, Dragomir, title = Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task, year = 2018, eprint = 1809.08887, archivePrefix = arXiv, primaryClass = cs.CL, url = https://arxiv.org/abs/1809.08887

@miscYuEtAl2019, author = Yu, Tao and Zhang, Rui and Er, Heyang and Li, Suyi and Xue, Eric and Pang, Bo and Lin, Xi Victoria and Tan, Yi Chern and Shi, Tianze and Li, Zihan and Jiang, Youxuan and Yasunaga, Michihiro and Yang, Sungrok and Zhang, Zilin and Wang, Dongxu and Li, Zifan and Er, Dragomir, title = CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Conversational Semantic Parsing, year = 2019, eprint = 1909.05378, archivePrefix = arXiv, primaryClass = cs.CL, url = https://arxiv.org/abs/1909.05378