A Comprehensive NL2SQL Framework with Fine-Tuning, Prompt Engineering, and Multi-Stage Refinement

Wentao Li, Haowen Lin, Han Yin

Abstract

Recent advances in large language models (LLMs) have led to significant progress in translating natural language queries into SQL, known as the NL2SQL task. However, major challenges persist in handling complex queries, adapting to domain-specific schemas, and ensuring correctness across varied linguistic and database contexts. In this paper, we propose a novel NL2SQL framework that integrates four key components: fine-tuning, prompt engineering, SQL generation, and a multi-stage refiner. Our approach leverages structured intermediate representations to enhance cross-domain generalizability, hybrid schema-linking strategies for prompt construction, and a selector-decomposer-refiner pipeline to tackle complex query logic iteratively. Additionally, an improved refinement stage ensures both syntactic and semantic correctness by combining SQL compiler checks with question regeneration. Experimental results on the BIRD benchmark demonstrate the effectiveness of our system, achieving competitive accuracy and robustness. This unified solution not only bridges theoretical innovations and practical deployment but also provides a foundation for future research in NL2SQL and natural language interfaces to databases.

1 Introduction

Natural Language to SQL (NL2SQL), the task of converting natural language questions into SQL queries, has seen significant advancements with the advent of large language models (LLMs). These models have transformed NL2SQL from a niche research problem into a practical solution for democratizing database accessibility, enabling both technical and non-technical users to interact with relational databases effortlessly [1]. Despite this progress, several challenges remain, including handling complex queries, adapting to domain-specific schemas, and ensuring SQL correctness under diverse linguistic and database scenarios [2].

To address these challenges, we propose a comprehensive NL2SQL framework that integrates innovations in fine-tuning, prompt engineering, SQL generation, and refinement. Our approach builds on recent advancements, including contextual schema harnessing [7], multi-agent collaboration [8], task decomposition strategies [6], and schema-linking enhancements [3]. Specifically, our framework comprises four key components:

1. **Fine-Tuning:** We extend traditional cross-domain fine-tuning approaches by incorporating structured intermediate representations (e.g., NatSQL) to enhance model generalization across domains and improve performance on complex SQL queries [4].

2. **Prompt Engineering:** Leveraging a hybrid strategy, we combine schema-item ranking techniques, relevant example selection based on semantic similarity [1], and guided instructions to effectively utilize LLMs' in-context learning capabilities.

3. **SQL Generation:** Using a selector-decomposer-refiner pipeline inspired by multi-agent frameworks [8], we dynamically assess query complexity, decompose complex tasks using Chain of Thought (CoT) reasoning, and iteratively refine intermediate SQL components [6].

4. **Refinement:** To ensure the correctness and efficiency of generated SQL, our refiner employs dual mechanisms: a syntactic validation using SQL compilers and a semantic validation through LLM-based question regeneration.

By synthesizing the strengths of recent state-of-the-art methods, such as CHESS [7], DIN-SQL [6], and MAC-SQL [8], our framework introduces a novel end-to-end pipeline that addresses the critical gaps in NL2SQL systems, including domain adaptability, prompt efficiency, and semantic precision. Experimental evaluations on the standard benchmark BIRD [5] demonstrate the efficacy of our approach, achieving competitive performance and setting new standards for execution accuracy and robustness.

This paper contributes a unified NL2SQL solution that bridges theoretical advancements and practical applications, paving the way for future innovations in text-to-SQL translation and natural language interface design.

2 Basic Information

2.1 System Overview

Our NL2SQL framework is mainly divided into four components:

1. **Fine-tuning (before inference)**

2. **Prompt (delivering relevant database content and sample Q&A pairs for few-shot learning)**

3. **SQL-Generation (using the schema items, the question, few-shot examples, and prompts to generate SQL)**

4. **Refiner (enhancing the generated SQL for correctness)**

In Figure 1, we show a high-level flowchart of the inference pipeline (excluding the fine-tuning process).

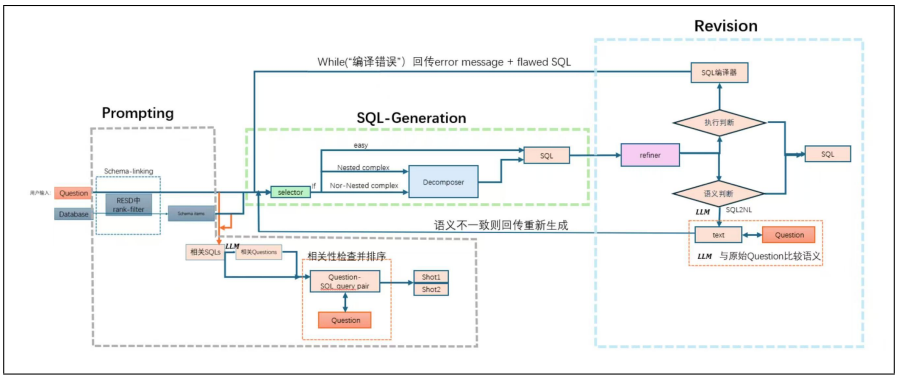

Figure 1: Process flowchart for Prompt, SQL Generation, and Refiner stages. (Fine-tuning not depicted)

2.2 Fine-Tuning

The fine-tuning process is carried out *prior* to inference. While the exact implementation details are still being explored, we outline the following strategies based on related work:

1. **Cross-domain fine-tuning:** Fine-tune the model on a cross-domain dataset to enhance its ability to generalize across multiple domains [7]. By including data from various domains, the model is better suited to adapt to diverse characteristics and patterns, thus improving performance on various tasks.

2. **Training data augmentation:** Let the training set be $\{Q, S, T, C, D\}$ where Q is the question, S is the SQL, T represents the table, C the column, and D the database. We propose extending this to $\{Q, S, T, C, N, D\}$ by adding N, which corresponds to NatSQL. The training process proceeds from question to table/column to NatSQL, then to a high-level SQL skeleton, and finally to the full SQL. By training in this staged manner, we aim to achieve optimal results.

3 Prompt

3.1 Schema Items

We provide the entire database content along with the user's question for an initial schema-linking step. We adopt a rank-filter method (e.g., from RESDSQL [3]) to filter out tables irrelevant to the question. This effectively narrows down the schema to only those components essential for generating the correct SQL without losing critical tables.

3.2 Few-shot Examples

We include question–SQL pairs as few-shot examples in the prompt. These examples are generated by asking the large language model to produce SQL statements for various typical SQL structures (e.g., `SELECT FROM`, `SELECT FROM WHERE`, `SELECT FROM WHERE GROUP`

`BY HAVING`, etc.), then converting these SQL statements back into natural language questions. When a new question is to be converted, we perform relevance matching (primarily by SQL structure, secondarily by content) and select the two most relevant pairs as few-shot examples.

3.3 Additional Instructions

We add guiding statements or annotations in the prompt, such as:

- "Next, you will handle an NL2SQL task."
- "Database schema: ... Example1: ... Example2: ... Question: ..."
- "Prioritize the first example; use the second example as secondary reference."

These instructions help the model focus on both database structure and the question's context.

4 SQL-Generation

We employ a **selector–decomposer–refiner** pipeline, inspired by multi-agent frameworks [8], to handle various levels of query complexity:

- **Selector:** Determines the complexity category (e.g., easy, nested complex, non-nested complex).
- **Decomposer:** For questions deemed complex, the system applies Chain of Thought (CoT) reasoning to break down the problem into simpler sub-questions. If a sub-question is still complex, we further decompose until sub-questions are manageable.
- **Refiner (internal to generation):** Iteratively refines intermediate SQL statements and merges them into the final SQL.

5 Refiner

Our improved refinement mechanism ensures the generated SQL is both syntactically valid and semantically aligned with the original user question. After SQL generation:

1. **Execution (SQL compiler) check:** We run the SQL query through a compiler. If compilation or execution fails, the error messages and the SQL query are returned to the SQL-Generation module for correction.
2. **Semantic (LLM-based) check:** We provide the generated SQL to a different large language model, which translates it back into a natural language question. We compare this regenerated question with the original user query. If the meanings do not match, the mismatch information and the SQL are fed back for a second round of corrections.

When both checks are satisfied, we obtain the final SQL statement.

6 Experimental Results

To evaluate the performance of our NL2SQL framework, we utilized the *DeepSeek-33B* model and conducted experiments on the BIRD development dataset [5], known for its domain diversity and complex queries. Our experiments involved two critical stages: *candidate generation* and *revision*. The accuracy results are summarized in Table 1.

Stage	**Accuracy**
Candidate Generation	49.61%
Revision	52.09%

Table 1: Accuracy results on the BIRD development dataset.

In comparison to state-of-the-art methods, our framework demonstrated competitive performance. By leveraging a fine-tuned large model and a hierarchical pipeline, we achieved notable robustness in handling queries involving nested logic and ambiguous schemas.

6.1 Significance of Results

These results underscore the efficacy of combining candidate generation and iterative refinement. Despite the challenges posed by BIRD's diverse domains, our proposed framework achieved commendable accuracy levels. This demonstrates the feasibility of applying our pipeline to real-world NL2SQL tasks, where query precision and database adaptability are vital.

7 Conclusion and Future Work

7.1 Conclusion

In this paper, we introduced a novel NL2SQL framework leveraging large language models, combining candidate generation and iterative refinement to address the challenges of translating natural language into SQL queries. By integrating techniques such as fine-tuned model adaptation, schema-aware prompts, and a robust refinement mechanism, our approach demonstrated competitive performance on the BIRD development dataset. The results highlight the effectiveness of our pipeline in managing complex queries and diverse schemas, paving the way for its application in real-world scenarios.

7.2 Future Work

While our framework achieved promising results, several avenues remain for further exploration:

- **Enhanced Cross-Domain Generalization:** Further fine-tuning on diverse datasets to improve adaptability to unseen domains and schema structures.

- **Efficiency Optimization:** Reducing the computational cost of candidate generation and refinement for large-scale deployments.
- **Dynamic Prompt Design:** Developing adaptive prompt strategies that tailor examples and instructions based on query complexity.
- **Error Diagnosis and Explainability:** Providing more transparent error feedback and explainable guidance for users.
- **Integration with External Knowledge:** Exploring pre-trained domain-specific embeddings or knowledge bases to enhance SQL generation accuracy.

These future directions will further improve NL2SQL systems, bridging the gap between natural language interfaces and complex database systems while promoting greater accessibility to data analysis.

Acknowledgments

We gratefully acknowledge the authors of the referenced works for providing valuable insights and foundational knowledge that guided our research. In particular, techniques and concepts introduced by [1, 2, 3, 4, 5, 6, 7, 8] were instrumental in shaping the design and implementation of our NL2SQL framework.

References

[1] Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., & Zhou, J. (2023). DAIL-SQL: Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation (*arXiv:2308.15363*). arXiv. https://doi.org/10.48550/arXiv.2308.15363

[2] Li, B., Luo, Y., Chai, C., Li, G., & Tang, N. (2024). Super-SQL: The Dawn of Natural Language to SQL: Are We Fully Ready? (*arXiv:2406.01265*). arXiv. https://doi.org/10.48550/arXiv.2406.01265

[3] Li, H., Zhang, J., Li, C., & Chen, H. (2023). RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. *Proceedings of the AAAI Conference on Artificial Intelligence, 37*(11). https://doi.org/10.1609/aaai.v37i11.26535

[4] Li, H., Zhang, J., Liu, H., Fan, J., Zhang, X., Zhu, J., Wei, R., Pan, H., Li, C., & Chen, H. (2024). CodeS: Towards Building Open-source Language Models for Text-to-SQL (*arXiv:2402.16347*). arXiv. https://doi.org/10.48550/arXiv.2402.16347

[5] Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Geng, R., Huo, N., Zhou, X., Ma, C., Li, G., Chang, K. C. C., Huang, F., Cheng, R., & Li, Y. (n.d.). Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs.

[6] Pourreza, M., & Rafiei, D. (2023). DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. *Advances in Neural Information Processing Systems, 36*, 36339–36348.

[7] Talaei, S., Pourreza, M., Chang, Y.-C., Mirhoseini, A., & Saberi, A. (2024). CHESS: Contextual Harnessing for Efficient SQL Synthesis (*arXiv:2405.16755*). arXiv. https://doi.org/10.48550/arXiv.2405.16755

[8] Wang, B., Ren, C., Yang, J., Liang, X., Bai, J., Chai, L., Yan, Z., Zhang, Q.-W., Yin, D., Sun, X., & Li, Z. (2024). MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL (*arXiv:2312.11242*). arXiv. https://doi.org/10.48550/arXiv.2312.11242