Enhanced SQL-RAG: A Comprehensive Framework for SQL Prompt Generation and Evaluation

Melika Nobakhtian
melika.nobakhtian2000@gmail.com

Ali Zahedzadeh
alizahedzadeh7@gmail.com

Mohammad Ali Yousefzadeh
mohammadali.yousefzadeh@gmail.com

1 Problem Statement

The goal of our project, "Enhanced SQL-RAG: A Comprehensive Framework for SQL Prompt Generation and Evaluation," is to develop an innovative system that leverages the capabilities of Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG) to generate SQL queries from natural language inputs. Our motivation stems from the significant challenge users face when accessing and querying relational databases due to the need for proficiency in Structured Query Language (SQL). By addressing this barrier, we aim to democratize data access and enable a broader range of users to extract valuable insights from databases efficiently and accurately.

The motivations behind our project are multifaceted:

- **Enhanced Accessibility:** By eliminating the need for SQL knowledge, we empower non-expert users to interact with databases using natural language queries. This democratizes data access, allowing more individuals and organizations to leverage their data for informed decision-making.
- **Improved Efficiency:** Traditional methods of data extraction via SQL can be time-consuming and error-prone, especially for users unfamiliar with SQL syntax. Our system streamlines this process by translating natural language queries into precise SQL statements, thereby saving time and reducing the effort required to obtain data.
- **Increased Transparency:** Our approach enhances transparency in data retrieval processes by generating clear and concise SQL queries from user questions. This not only promotes a better understanding of how data is retrieved but also helps users trust the results, knowing the exact queries executed against their databases.

1.1 Performance Evaluation

To monitor the performance of our approach, we use one of the metrics introduced in (Zhong et al., 2020). Following the official test-suite [1], we consider these metrics:

- **Exact Match (EM):** This metric focuses on whether the predicted SQL query exactly matches the ground truth (correct) SQL query in terms of its structure and components. It considers each clause within the SQL query (e.g., SELECT, FROM, WHERE) and checks for a perfect match between the predicted and ground truth versions.
- **Execution Accuracy (EX):** This metric goes beyond just syntactic structure and focuses on the functional correctness of the predicted SQL query. It involves actually executing the predicted SQL query on the target database and comparing the results with the expected outcome based on the ground truth SQL query. Unlike EM, which only checks syntactic correctness, EX considers the semantic correctness of the query.

2 What We Proposed vs. What We Accomplished

- **Build an enhanced SQL-RAG model and examine its performance:** We aimed to build an enhanced SQL-RAG model to improve SQL query generation by retrieving relevant information. For this project, we focused on developing a system that retrieves

[1] https://github.com/taoyds/test-suite-sql-eval.

related question-query pairs to assist our model in generating accurate SQL queries. But we had plans to retrieve more information from RAG.

- **Examine different prompt engineering techniques to discover the best prompt for Text-to-SQL task:** We examined different forms of prompts and saw their effectiveness through the available metrics.

- **Evaluation with EM metric:** Since most of the articles that we reviewed used the EX metric between the EX and EM metrics, we did not use the EM metric either. The usefulness and applicability of metric EM is somewhat doubtful.

- **Perform in-depth error analysis to identify common failure modes and areas for improvement:** The output of some of the models that we tested did not have a proper format, especially the relatively smaller models, in addition to the fact that it was emphasized that they only produce SQL output. The SQL output was given along with natural language, which was impossible to execute the code, so it was necessary to parse the output format before execution and only extract the SQL code.

3 Related Work

DAIL-SQL: (Gao et al., 2023) propose a benchmark evaluation to address the lack of systematic benchmarks for text-to-SQL systems powered by LLMs. Their work, DAIL-SQL, focuses on prompt engineering techniques, investigating methods to improve the effectiveness of LLMs in this task. The evaluations are conducted on Spider dataset (Yu et al., 2019). They explore various approaches, including:

- **Question Representation:** Different methods for phrasing the question within the prompt are evaluated for their impact on LLM performance (e.g., natural language questions vs. keyword-based formats vs. structured representations).
- **Example Selection and Organization:** The study investigates strategies for selecting and structuring relevant examples within the prompt, analyzing their influence in a few-shot learning scenario (e.g., providing successful query examples or highlighting potential pitfalls).

Additionally, They compared two approaches for leveraging LLMs for text-to-SQL:

- **In-Context Learning:** The prompt itself guides the LLM towards generating the correct SQL query without any specific training on text-to-SQL tasks.
- **Supervised Fine-Tuning:** This approach involves training the LLM on existing text-to-SQL datasets before using it for query generation.

A crucial aspect of their work is token efficiency. Recognizing the cost implications of LLM interactions, they emphasize the importance of designing prompts that minimize the number of tokens required while maintaining accuracy.

PET-SQL: (Li et al., 2024) propose PET-SQL, a two-stage prompt-based framework that leverages LLMs for Text-to-SQL tasks. They address the limitations of LLM-based approaches in handling complex database information and user intentions. PET-SQL introduces a novel prompt representation strategy that incorporates two key elements:

- **Reference-enhanced Representation:** This representation includes schema information (table and column names) along with randomly sampled cell values from the database tables. This provides richer context for the LLM to understand the data structure and relationships.
- **Two Stage Processing:**
  - **Stage 1:** The system retrieves question-SQL pairs as demonstrations and presents them to the LLM along with the reference-enhanced prompt. This prompts the LLM to generate a preliminary SQL query (PreSQL).
  - **Stage 2:** Schema linking is performed on the entities mentioned in the PreSQL. This process condenses the information from the PreSQL, focusing on the essential elements for query construction. A simplified prompt with the linked schema is then presented to the LLM, guiding it to generate the final SQL query.

As a final step, PET-SQL employs cross-consistency across different LLMs for refinement.

This approach leverages the strengths of various LLMs to improve the final predicted SQL query. The study demonstrates that PET-SQL achieves state-of-the-art (SOTA) results on the Spider benchmark, reaching an execution accuracy of 87.6%.

RAG-based Text-to-SQL with Sample-aware Prompting and Dynamic Revision Chain: (Guo et al., 2023) propose a retrieval-augmented prompting method to address challenges in LLM-based Text-to-SQL tasks. Specifically, they tackle the issue of a large semantic gap between retrieved question-SQL demonstration examples and the user's actual question. Their approach incorporates two key elements:

- **Sample-aware Prompting:** This involves using demonstrations that go beyond just question-SQL pairs. The demonstrations include information about the composition of SQL operators and details relevant to the specific question. This provides richer context for the LLM to understand the task.
- **Dynamic Revision Chain:** To bridge the semantic gap and achieve accurate SQL generation, the framework employs a dynamic revision chain. This process iteratively refines the LLM's output based on feedback from previously generated SQL queries.

To assist in retrieving relevant question-SQL pairs, They proposed two strategies:

- **LLM-based Question Simplification:** They leverage LLMs to simplify the user's original question, aiming to unify syntax and clarify the user's intent.
- **Strategic retrieval based on intent:** This strategy retrieves questions that share similar intent with the user's question, ensuring the retrieved demonstrations are more relevant to the specific task.

DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction: Few-shot models, despite improvements over zero-shot models, continue to face significant challenges with complex SQL queries, particularly in schema linking and handling queries involving multiple joins or nested structures. Inspired by techniques such as chain-of-thought prompting (Wei et al., 2023) and least-to-most prompting, a novel approach has been proposed (Pourreza and Rafiei, 2023) that decomposes the problem into smaller sub-problems, solving each individually before constructing a solution for the original problem. This method draws from successful multi-step task performance enhancements seen in math word problems and compositional generalization (Cobbe et al., 2021; Lake and Baroni, 2018).

The process of writing SQL queries can be broken down into distinct phases: **(1)** detecting relevant database tables and columns, **(2)** identifying the general query structure for complex queries, **(3)** formulating procedural sub-components, and **(4)** writing the final query based on sub-problem solutions. Accordingly, the proposed method consists of four modules: schema linking, query classification and decomposition, SQL generation, and self-correction, all implemented using prompting techniques to leverage LLM capabilities.

The Schema Linking module identifies references to database schema and condition values in natural language queries, a crucial step for generalizability across domains and complex query synthesis (Lei et al., 2020). To address schema linking failures, a prompt-based module was designed using examples from the Spider dataset, following a chain-of-thought template (Kojima et al., 2023). The Query Classification and Decomposition module categorizes queries into easy, non-nested complex, and nested complex classes, addressing issues with joins and procedural components by decomposing them into manageable subtasks.

The SQL Generation module tackles the mismatch problem between natural language and SQL (Guo et al., 2023) by employing different strategies based on query complexity. For easy queries, simple few-shot prompting suffices, while non-nested complex queries utilize an intermediate representation like NatSQL (Gan et al., 2021). The most complex nested queries are handled through multi-step decomposition, where sub-queries are solved sequentially to form the final SQL statement. This structured, modular approach ensures accurate identification and handling of complex query structures, optimizing the use of LLMs for text-to-SQL tasks.

4 Data

For our project, we will utilize the Spider dataset, which provides a comprehensive benchmark for building and evaluating cross-domain text-to-SQL systems. The Spider dataset is a large-scale complex and cross-domain semantic parsing and text-to-SQL dataset.

- **Description:** Annotated by 11 Yale students, Spider aims to develop natural language interfaces to cross-domain databases.
- **Contents:**
  - 10,181 natural language questions.
  - 5,693 unique complex SQL queries.
  - 200 databases with multiple tables.
  - Covers 138 different domains.
- **Challenges:** Different complex SQL queries and databases appear in the training and test sets, requiring systems to generalize to new SQL queries and database schemas.

An example query from the Spider dataset:

Question: Which countries in Europe have at least 3 car manufacturers? **SQL Query:**

```
SELECT T1.country_name
FROM countries AS T1
JOIN continents AS T2 ON
T1.continent = T2.cont_id
JOIN car_makers AS T3 ON
T1.country_id = T3.country
WHERE T2.continent = 'Europe'
GROUP BY T1.country_name
HAVING COUNT(*) >= 3
```

4.1 Annotation

The Spider dataset is pre-annotated with pairs of natural language questions and their corresponding SQL queries.

4.2 Suitability for Research Questions

The Spider dataset's complexity, diversity, and comprehensive nature make it highly suitable for our research. It will enable us to rigorously evaluate our Enhanced SQL-RAG framework across various scenarios, ensuring its ability to generalize to new SQL queries and database schemas. The Spider dataset will be used to benchmark our models and to ensure robust performance in cross-domain text-to-SQL tasks.

4.3 The Data We Use

Due to the high costs associated with using the API, we opted not to utilize the full test dataset. Instead, we selected a sample of 300 test data points. To ensure that our sample accurately represented the overall test dataset, we first analyzed the data distribution. The test data encompasses four levels of question difficulty: easy, medium, hard, and extra hard. We ensured that our selected sample of 300 test data points maintained the same proportional representation of these difficulty levels.

5 Baseline

In the following table (Table 1) you can see top models and papers results on the Spider dataset. The reported metrics are explained in Section 1.

6 Tools

For our project, we utilize a combination of libraries, toolkits, and platforms to facilitate data processing, model development, and evaluation.

6.1 Model Development

- **Hugging Face Transformers:** For leveraging pre-trained language models such as LLAMA and their variants to our text-to-SQL tasks. Specially, we utilized Unsolth AI library to access open-source models and their quantified versions in a convenient way.

6.2 Retrieval-Augmented Generation (RAG)

- **ChromaDB:** A vector database is a crucial part of every RAG system. For this purpose, We use ChromaDB, an open-source vector database with variety of options.

6.3 Development Environment

- **Google Colab:** For initial model development and experimentation with free access to GPU resources.

6.4 APIs and External Services

- **OpenAI API:** For accessing and integrating advanced language models such as GPT-3, GPT-4.

7 Our Approach

We use an approach for Text-to-SQL generation that leverages the capabilities of LLMs to generate SQL queries based on user-submitted questions in natural language, utilizing the RAG approach.

Rank	Model	Execution Accuracy (Test)	Exact Match Accuracy (Test)
1	PET-SQL	87.6	66.6
2	DAIL-SQL + GPT-4 + Self-Consistency	86.6	84.4
3	DIN-SQL + GPT-4	85.3	60.0
4	C3 + ChatGPT + Zero-Shot	82.3	81.8
5	RESDSQL-3B + NatSQL	79.9	72.0
6	T5-3B + NatSQL + Token Preprocessing	78.0	73.7
7	Graphix-3B + PICARD	77.6	81.0
8	T5-SR	75.2	72.4
9	T5-3B + PICARD	75.1	71.9

Table 1: Top models and their results on the Spider dataset.

Upon receiving a query, the system retrieves pertinent data, including relevant SQL commands and similar questions from our database. Additionally, relevant sections of the database table related to the query can also integrated into the LLM input as part of the prompt. Given the substantial volume of information that may be included in the model's input, it is crucial to apply meticulous prompt engineering to ensure clarity and effectiveness in the query generation process. Our approach focuses on two key components:

1. Retrieval Augmented Generation (RAG):

Recognizing the importance of contextual information for accurate SQL query generation, we employ RAG to retrieve relevant information from two primary sources:

- **Related Questions/Queries:** We leverage the dataset to retrieve question-SQL pairs similar to the user's query. These examples serve as valuable demonstrations for the LLM, guiding it towards generating more accurate and relevant SQL queries.
- **Implementation Details:** For the retrieval part, we used chroma-db as our vector database to retrieve useful information for answering the questions. We utilized the training data questions from the Spider dataset to create this database. Initially, we obtained embeddings of training questions using OpenAI-Embeddings and the text-embedding-ada-002 model. With these embeddings, we were able to create our vector database.

  Our retrieval process works as follows:

  For every question in the test set, we first find k similar questions (where $k = 5$). The metric used to measure similarity is Cosine Similarity. This retrieved information is then used to prompt the language model and generate the SQL query according to the question.

- **Exploration of Large Language Models:** To further enhance our approach, we plan to investigate the performance of various LLMs. This includes both API models like GPT-3.5 and GPT-4, as well as open-source options such as Llama and Mistral. We employed two LLMs for our work: Llama3-8B-4bit-instruct and GPT3.5-Turbo.

  Due to computational resource constraints and slow inference times, we used the quantized version of llama3-8B. We considered using GPT4, but its high cost deterred us from doing so.

  To prompt these models, we initially provided some useful instructions about the input and output format. We used two strategies for prompting the models:

  1. Using only the retrieved samples from the vector database.

  2. Adding information about the tables of the database to the prompt.

2. Prompt Engineering:

Our approach involves designing and evaluating different prompts to effectively guide the LLM in generating the desired SQL query. These prompts incorporate information, such as schema details and relevant question-SQL examples. To identify the most effective prompt, we provided informa-

tion deemed useful based on the nature of the data and the task at hand, testing several different input prompt scenarios. The results varied significantly, with execution accuracy ranging from approximately 10% to 80%.

Our method addresses the challenges faced by few-shot models when handling complex SQL queries, including schema linking and nested structures. Drawing inspiration from chain-of-thought and least-to-most prompting techniques, we decompose the text-to-SQL task into smaller sub-problems to enhance the query generation process. This comprehensive approach comprises four key modules:

- **Schema Linking Module:** The Schema Linking Module is designed to identify references to the database schema and condition values within natural language queries. This is achieved using a prompt-based method that incorporates ten randomly selected samples from the Spider dataset. The prompts follow a chain-of-thought template, beginning with the phrase "Let's think step by step" to guide the model through the schema linking process. This approach enhances the model's accuracy in linking schema elements to the corresponding parts of the query.

- **Classification and Decomposition Module:** The Classification and Decomposition Module plays a crucial role in categorizing queries into three distinct types: easy, non-nested complex, and nested complex. This module is responsible for detecting the tables that need to be joined and identifying any subqueries within nested queries. By utilizing different prompts tailored to each query class, the module ensures that the SQL generation process is accurate and contextually appropriate for the complexity of the query.

- **SQL Generation Module:** The SQL Generation Module focuses on bridging the gap between natural language questions and SQL statements, addressing the commonly encountered mismatch problem. For non-nested complex queries, this module uses NatSQL as an intermediate representation to simplify the translation process. For nested complex queries, the module employs a structured prompt format that first solves any sub-queries before generating the final SQL statement. This step-by-step approach ensures that complex queries are accurately represented in SQL.

- **Self-correction Module:** The Self-correction Module is designed to refine the generated SQL queries by addressing issues such as missing or redundant keywords. This module uses two types of prompts to optimize the correction process based on the behavior of different models. For CodeX models, a generic prompt is used, while a gentle prompt is employed for GPT models. This differentiation ensures that the correction process is tailored to the strengths and weaknesses of each model, improving the overall accuracy of the generated SQL queries.

Benchmarking and Evaluation:

- We evaluated our approach on the Spider benchmark dataset. This established benchmark allows for a comprehensive comparison with existing Text-to-SQL models. In the following table (Table 1) you can see top models and papers results on the Spider dataset.

By combining RAG and carefully crafted prompts, our work aims to empower LLMs for more accurate and robust Text-to-SQL generation.

In the following table (Table 2) you can see our models and results on the selected sample of Spider dataset according to the level of questions.

8 Results and Comparisons

The results demonstrate the effectiveness of including information about the tables of the database, which surprisingly improved performance. However, the performance difference between the two models is not significant. The Llama model achieved competitive results compared to GPT3.5.

- The GPT-3.5 Turbo model combined with the DIN-SQL approach achieves the highest overall execution accuracy at 78.0%. which confirms the great effect of the DIN-SQL approach.

- The GPT-3.5 Turbo combined with RAG and few-shot prompting, shows a significant drop in performance compared to the top-ranked model, with an overall execution accuracy of

Rank	Model	EX (Easy)	EX (Medium)	EX (Hard)	EX (Extra Hard)	Execution Accuracy (Overall)
1	GPT-3.5 Turbo + DIN-SQL	91.2	76.0	77.8	58.5	78.0
2	GPT-3.5 Turbo + RAG + Few-shot + Table Info	81.2	64.0	63.0	41.5	65.3
3	GPT-3.5 Turbo + Table Info	77.5	64.0	63.0	46.3	65.0
4	Llama3-8b (Quantized) + RAG + Few-shot + Table Info	77.5	64.0	55.6	36.6	62.3
5	Llama3-8b (Quantized) + Table Info	58.8	52.0	44.4	29.3	49.3
6	Llama3-8b (Quantized) + Few-shot	17.5	11.2	16.7	12.2	14.0

Table 2: Performance (Execution Accuracy) of our models on the selected sample of the Spider dataset.

65.3%. In this model, the length of the input prompt is much shorter, and this decrease in score is justified.

- The GPT-3.5 Turbo with table-only info model shows very little difference with the second model. It seems as if RAG has no special effect on GPT-3.5 model. The question that may come to mind is that RAG does not have the same effect on large models as it does on small models, which requires more tests and evaluation to investigate this issue.

- Llama3-8b (Quantized) combined with RAG and few-shot prompting, ranks fourth with an overall execution accuracy of 62.3%. While it shows comparable performance to the GPT-3.5 model. It can be said that with RAG you can score as much from small models as from large models. Of course, it can be argued that RAG has a greater effect on small models than on large models.

- The fifth-ranked model, Llama3-8b (Quantized) with only table information, shows a further decline in performance, with an overall execution accuracy of 49.3%. A doubt that could be raised was that RAG is not very useful and the good performance of the previous models was due to the table information that was given to it. So we only gave the table information to the model without using RAG. We saw that the performance of the model decreased by 21% and this obvious difference shows the effect of RAG.

- The model that performed the worst was the Llama3-8b (Quantized) using few-shot prompting alone, with an overall execution accuracy of 14.0%. This model shows particularly poor performance across all difficulty levels, highlighting the limitations of using few-shot prompting without additional contextual information from the table schema.

9 Conclusion

Overall, the study underscores the importance of schema context and retrieval-augmented generation in improving SQL query generation accuracy. The GPT-3.5 Turbo model, when combined with the DIN-SQL approach, achieved the highest accuracy at 78.0%, highlighting the approach's efficacy. The substantial performance drop observed when using only table information with Llama3-8B (49.3% accuracy) compared to the RAG-enhanced version (62.3% accuracy) reaffirms RAG's utility. Additionally, the poorest performance with few-shot prompting alone (14.0% accuracy) emphasizes the necessity of integrating contextual information. These findings suggest that while both GPT-3.5 Turbo and Llama3-8B models can achieve competitive results, the integration of RAG and detailed table information is crucial for maximizing their performance. Future work should further explore the differential impact of RAG on models of varying sizes and investigate optimal prompt lengths for enhancing performance.

References

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. (2021). Training verifiers to solve math word problems.

Gan, Y., Chen, X., Xie, J., Purver, M., Woodward, J. R., Drake, J., and Zhang, Q. (2021). Natural sql: Making sql easier to infer from natural language specifications.

Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., and Zhou, J. (2023). Text-to-sql empowered by large language models: A benchmark evaluation.

Guo, C., Tian, Z., Tang, J., Li, S., Wen, Z., Wang, K., and Wang, T. (2023). Retrieval-augmented gpt-3.5-based text-to-sql framework with sample-aware prompting and dynamic revision chain.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. (2023). Large language models are zero-shot reasoners.

Lake, B. M. and Baroni, M. (2018). Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks.

Lei, W., Wang, W., Ma, Z., Gan, T., Lu, W., Kan, M.-Y., and Chua, T.-S. (2020). Re-examining the role of schema linking in text-to-SQL. In Webber, B., Cohn, T., He, Y., and Liu, Y., editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6943–6954, Online. Association for Computational Linguistics.

Li, Z., Wang, X., Zhao, J., Yang, S., Du, G., Hu, X., Zhang, B., Ye, Y., Li, Z., Zhao, R., and Mao, H. (2024). Pet-sql: A prompt-enhanced two-stage text-to-sql framework with cross-consistency.

Pourreza, M. and Rafiei, D. (2023). Din-sql: Decomposed in-context learning of text-to-sql with self-correction.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. (2023). Chain-of-thought prompting elicits reasoning in large language models.

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. (2019). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task.

Zhong, R., Yu, T., and Klein, D. (2020). Semantic evaluation for text-to-sql with distilled test suite. In *The 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.