Self-Contrastive Loop of Thought Method for Text-to-SQL Based on Large Language Model

Fengrui Kang[1,2], Mingxi Tan[1], Xianying Huang[2,*], Shiju Yang[1]
[1]Xffuture, [2]Chongqing University of Technology
kenfree000@gmail.com, tanmingxi@xffuture.com,
hxy@cqut.edu.cn, yangshiju@xffuture.com
Correspondence: hxy@cqut.edu.cn

Abstract

Text-to-SQL is a task with excellent prospects and challenges, and it aims to convert natural language queries (NL) into corresponding structured query language (SQL) statements. The main challenge of this task is how to efficiently transform unstructured data and structured data. In recent years, the emergence of large language models (LLMs) has further promoted the development of this field. However, current LLM-based text-to-SQL methods rely on specific few-shot example construction, resulting in poor performance across domains. To solve this problem, we propose a text-to-SQL method of self-contrastive loop of thought structure. This method designs the LLM inference process as a loop structure based on the comparison of positive and negative examples. The model optimizes the generated results through continuous verification and error correction, greatly improving accuracy and reducing dependence on few-shot example construction. The experimental results on SPIDER and BIRD datasets show that this method can generate SQL with higher precision without relying on few-shot example construction.

1 Introduction

The goal of text-to-SQL is to generate SQL based on natural language. This technique can generate the corresponding SQL statements by verbal description. Due to its broad application prospect and challenge, this task has attracted wide attention(Bogin et al., 2019; Elgohary et al., 2020; Chen et al., 2021; Lin et al., 2020).

In the early stages, such research typically achieved text-to-SQL through rule-based designs and pre-trained models. Design methods improved the quality of SQL generation by reinforcing the alignment between text and database schemas, such as RAT-SQL (Wang et al., 2020), SDSQL(Hui et al., 2021), LGESQL(Cao et al., 2021), RESDSQL(Li et al., 2023a). Additionally, some studies employed various pre-training strategies, such as TaBERT(Yin et al., 2020), GRAPPA(Yu et al., 2021), GAP(Zhao et al., 2022), and MIGA(Fu et al., 2023), these models enhanced ability to capture the complex relationship between natural language and SQL structures. Together, these efforts improve the performance of text-to-SQL.

With the development of LLM, recent studies have demonstrated the superior ability of LLM in complex tasks(Gao et al., 2023; Nan et al., 2023b; Imani et al., 2023). In the text-to-SQL field, the LLM-based approach exceeds previous work on multiple datasets without any fine-tuning or training(Gu et al., 2023; Pourreza et al., 2024; Nan et al., 2023a).

Based on the advanced performance of LLM, prompt-based methods have further promoted the progress of text-to-SQL. Typical work such as DINSQL(Pourreza and Rafiei, 2023), DAIL-SQL(Gao et al., 2024), TA-SQL(Qu et al., 2024)and MACSQL(Wang et al., 2023). These methods are based on the few-shot prompt method, combined with chain-of-thought technology to decompose the task to improve LLM performance.

Although the prompt-based method has made significant progress, it still has some problems. First, the performance depends on the quality of the few-shot examples, which can lead to LLM not achieving the potential performance. Second, most of this work uses a linear thinking workflow to guide LLM generation, which comprises the search space. Third, this work improves the ability to deal with complex problems through detailed decomposition tasks. However, it may introduce an additional risk of hallucination, which can adversely affect the results.

To solve the above problems, we propose a text-to-SQL method based on a self-contrastive loop

* Corresponding author

Proceedings of the 1st Joint Workshop on Large Language Models and Structure Modeling (XLLM 2025), pages 71–85

of thought structure (SCL). The LLM generation process is constructed as a discriminant loop structure based on the contrast of positive and negative examples so that the model can improve the accuracy of the results in the continuous self-correction. This reduces the dependence on the construction of few-shot examples, and the generation relies only on the self-contrast between positive and negative examples. With this approach, the LLM can better focus on outputs similar to positive examples and exclude erroneous results similar to negative examples, reducing the inference burden of the LLM. On this basis, we summarize the contributions of this paper as follows.

- Designed a question-splitting strategy that enhances the utilization of question information via skeleton-based positive example retrieval and entity-schema linking. In addition, positive example retrieval replaces the few-shot construction, simplifying prompt design.
- Enhance the LLM's verification and selection abilities through result-guided execution and logical verification, integrated with a selection mechanism. This expands the search space for generation.
- A loop-of-thought structure is designed to guide the LLM through self-contrast between positive and negative examples, enhancing its ability to handle text-to-SQL tasks without relying on few-shot example construction.

2 Methodology

2.1 Overview

SCL-SQL is a self-contrastive framework for text-to-SQL. As shown in Figure 7. We first decouple the query into entity and skeleton parts to improve schema linking and example retrieval. Then, a loop-structured reasoning process guides the LLM to generate, execute, and verify SQL with both positive and negative examples. Finally, a voting-based mechanism selects the most reliable result.

2.2 Query Split

To fully utilize the information in the question, we propose a novel entity-skeleton splitting strategy that divides the question into entity and skeleton parts, which respectively guide schema linking and few-shot example construction.

2.2.1 Entity-guided Schema Linking

Schema linking identifies relevant schema elements (e.g., table names and columns), effectively reducing mismatches and minimizing hallucinations, as validated in prior studies (Gan et al., 2023; Yang et al., 2024). This work adopts an entity-guided approach by extracting keywords using YAKE (Campos et al., 2020), which divides the question into keyword (entity) and non-keyword (skeleton) parts, as shown in Equation 1.

$$\begin{aligned} T &= \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}\} \\ e &= \{x_2, x_4, x_7, x_8, x_9\}, \quad e \in T \\ s &= \{x_1, x_3, x_5, x_6, x_{10}\}, \quad s \in T \end{aligned} \tag{1}$$

Where T represents the original question, e represents the extracted entity, and s represents the question skeleton, where $e \cup s = T$.

The prompt for schema linking has three parts (entity e, full schema, question). The LLM matches all similar tables and corresponding columns during the linking process.

2.2.2 Skeleton-Guided Example Retrieval

An efficient dynamic retrieval method is introduced to replace manual few-shot example construction. Questions with similar intents (e.g., searching, counting, percentage calculation) exhibit similar structural patterns. By matching skeleton structures, representative positive examples can be retrieved to support model reasoning. The Jaccard coefficient is used to measure structural similarity between the current question and existing samples, as shown in Equation2.

$$Sim(s, D) = \frac{|s \cap D|}{|s \cup D|} \tag{2}$$

Where D represents existing data, s represents the question skeleton, the score calculated by each row of data is finally sorted from high to low, and the corresponding number of pieces is returned as positive examples by setting the initial positive number P_0. The advantage of this method is that it can ensure the efficiency of retrieval while performing accurate retrieval.

Retrieved positive examples are formatted as Question-SQL (QS) pairs, with the prefix `[valid]` added to each pair to indicate 'positive' identification in the prompt.

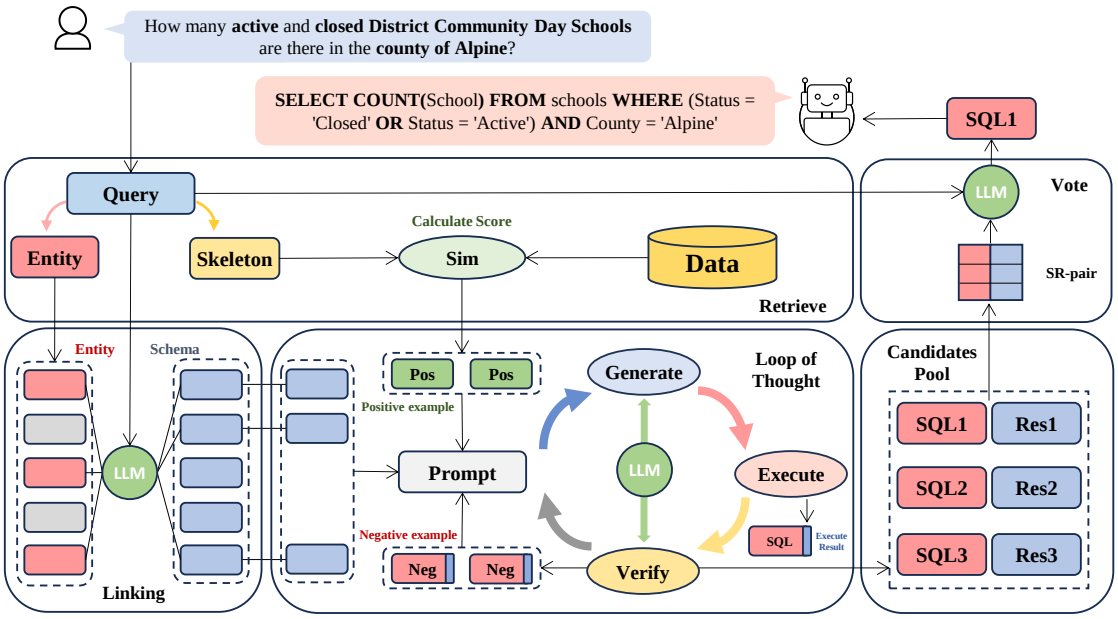

Figure 1: SCL-SQL structure illustration. The 'Pos' mark represents the positive sample, the 'Neg' mark represents the negative sample, and SQL and Res represent the generated SQL and the execution result. The numbers behind them indicate the number of loops in which they are generated.

2.3 Loop of Thought

We propose a loop-structured LLM reasoning process to address complex text-to-SQL problems. The loop consists of four stages: (1) Prompting, (2) Generation, (3) Execution, (4) Verification. The goal of these four stages is to enable the LLM to engage in iterative reasoning by comparing positive and negative examples. The specific process is illustrated as follows.

(1) Prompting: A prompt is constructed by integrating the original question, the linked database schema, positive examples retrieved via skeleton-guided example retrieval, and negative examples from the verification stage, aiming to guide the LLMâĂŹs SQL generation.

(2) Generation: The constructed prompt is fed into the LLM to generate candidate SQL statements.

(3) Execution: The generated SQL statements are executed, and both the SQL and their execution results are passed to the verification stage.

(4) Verification: A self-contrastive mechanism is applied, where the LLM evaluates whether the SQL and its result are logically correct. The verification outcome is represented as a binary pseudo-probability, with 0 indicating failure and 1 indicating success.

The most critical step in the loop of thought is result verification. However, (Huang et al., 2024) showed that directly relying on LLM for result verification and correction may lead to performance degradation. In the text-to-SQL task, the key evaluation criterion is whether the SQL retrieves the expected results. To address this, we design a result-guided verification mechanism, focusing on two aspects:

(1) Execution verification: Whether the SQL executes successfully and whether the result is empty.

(2) Logical verification: Whether the result logically matches the query intent. For instance, when querying the most sold car, the expected result should include the car's name. If irrelevant or multiple results are returned, the SQL is considered logically invalid.

If the verification passes, the SQL-Result (SR) pair is added to a candidate pool. If the pool contains only one entry, it is selected as the final output. If the verification fails, the SR pair is treated as a negative example and used in the next round of

prompt construction. It is structured as a triplet SQL, result, cause, with a prefix label [invalid]. As shown in Figure 2.

In the whole process of the loop of thought, negative examples are not only used to correct errors but also an important tool to guide the LLM to optimize the thought of generation in the loop. By emphasizing negative examples, avoid repeating errors. The process is repeated until the verification passes or a maximum loop times L is reached. In this process, to reduce the influence of positive examples on modification, a positive decay mechanism is designed here. The number of positive examples will decrease with the increase in the number of negative examples, and their relationship is shown in Equation 3.

$$P = P_0 - \alpha \cdot N, \alpha \in \mathbb{Z} \tag{3}$$

Where P_0 represents the initial number of positive examples, P represents the current number of positive examples, N represents the number of negative examples, α is the correlation coefficient of positive and negative examples, and must be an integer. This mechanism aims to prevent the adverse effects of positive examples on modifications at a later stage.

Through this design, it is not necessary to make specific few-shot examples. The generation process can be manipulated by positive and negative examples. In loop execution, the LLM can quickly provide results for simple questions, while the LLM can call the loop to think repeatedly and provide more diverse responses for complex questions.

2.4 Candidates Selection

Despite loop-based refinement, LLM may still misjudge results. To mitigate this, each SR pair is stored in a candidate pool. When multiple candidates exist, a voting mechanism is employed to determine the final selection. The LLM evaluates each SR pair based on the user query and provides votes accordingly. Compared to directly correcting erroneous SQLs, selecting the best one through voting among existing candidates is more efficient and reliable. The effectiveness of this strategy is further analyzed in Section 4.3.

[valid]
Question: Please list the phone numbers of the schools with the top 3 SAT excellence rate.
SQL: SELECT phone_number FROM schools ORDER BY sat_excellence_rate DESC LIMIT 3;
[valid]
Question: List the names of the top 10 products with the highest customer ratings, but only
if they have more than 100 reviews.
SQL: SELECT product_name FROM products WHERE review_count > 100 ORDER BY
customer_rating DESC LIMIT 10;

[invalid]
SQL:
SELECT employee_email FROM employees ORDER BY annual_salary DESC LIMIT 5;
Result:
OperationalError: no such column: employee_email
Reason:
employee_email does not exist in the table, which is obviously the incorrect column
[invalid]
SQL:
SELECT email, annual_salary FROM employees ORDER BY annual_salary DESC LIMIT 5;
Result:
[('john.doe@example.com', 150000), ('jane.smith@example.com', 145000)…..
Reason:
The question only asks to return the mailbox, and the salary is returned inside, which obviously does not meet the requirements of the question

Figure 2: Positive and negative examples. Green [valid] indicates a positive example label, and red [invalid] indicates a negative example label.

3 Experiments

3.1 Setup

For the positive instance retrieval number P_0, we set it to 3, the maximum loop number L of the thinking loop to 3, and the correlation coefficient of positive and negative cases to 1. The temperature of LLM is set to 0.1.

3.2 Dataset

[1] SPIDER(Yu et al., 2018)[1]: A large-scale, complex cross-domain text-to-SQL dataset containing over 10,000 questions and nearly 6,000 unique SQL queries covering 200 databases and 138 different domains.

[2] BIRD(Li et al., 2023c)[2]: The most challenging large-scale cross-domain text-to-SQL benchmark. Contains 12,751 pairs of data and 95 databases covering 37 fields.

[1] https://yale-lily.github.io/spider
[2] https://bird-bench.github.io/

3.3 Baseline

To ensure a fair comparison, few-shot learning methods with validated results are selected as baseline models, such as DIN-SQL(Pourreza and Rafiei, 2023), DAIL-SQL(Gao et al., 2024), TA-SQL(Qu et al., 2024), MAC-SQL(Wang et al., 2023). The second category evaluates the improvement of LLMs using the proposed SCL method, with GPT-3.5, GPT-4, and GPT-4o as base models.

3.4 Evaluation Metrics

To facilitate comparison with other similar types of work, the evaluation process mainly includes the following two indicators: (1)EX: Represents the execution accuracy of the generated SQL by calculating how close the SQL is to the real SQL execution result. (2)VES: Used to calculate the efficiency of generating valid SQL. SQL efficiency considerations are added to accuracy.

3.5 Result

As shown in Table 1, SCL-SQL outperforms existing methods on SPIDER and BIRD, especially on the complex BIRD dataset. It achieves higher EX and VES scores, improving robustness and semantic accuracy. The gap between GPT-4 and GPT-4o further demonstrates the method's scalability. Our method generates executable and accurate SQL, highlighting its effectiveness and generalizability.

As shown in Table 2, SCL-SQL consistently enhances the performance of all evaluated LLMs. Notably, the improvements extend beyond simple queries, with significant gains on moderate and challenging samples. This demonstrates that SCL-SQL not only improves overall accuracy but also strengthens complex reasoning and schema understanding. Moreover, while stronger models already perform well, SCL-SQL further boosts their effectiveness, showing strong compatibility and generalizability across different model capacities.

3.6 Ablation Study

We conducted ablation experiments for the method proposed in the paper to verify the effectiveness of the proposed method. The ablation experimental results and analysis of different methods are shown in Table 3.

It can be seen that each module brings incremental improvements, but together they produce a synergistic effect. Positive examples enhance guidance, negative examples improve discrimination, and the thought loop significantly boosts iterative correction. Each module contributes from a different perspective, jointly achieving the best performance.

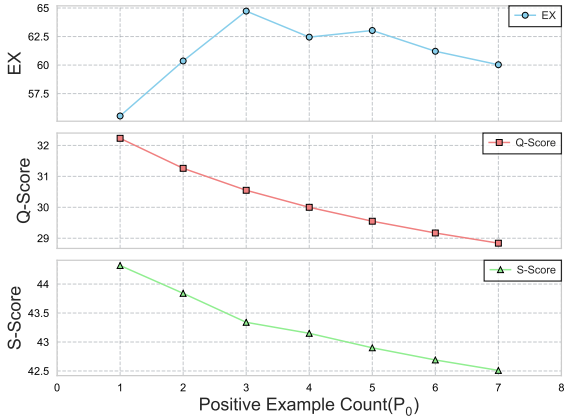

Figure 3: The effect of the number of positive examples, where EX represents the execution accuracy, and Q-Score represents the Jaccard score of the question versus the question in the positive example. S-Score indicates the Jaccard score of the ground truth SQL and the SQL in the positive example.

4 Analysis

To show the details of SCL-SQL, we conducted a detailed analysis of the parts of SCL-SQL, including hyper-parameter experiments and result analysis, where all analysis is based on the BIRD development set, and defaults are used for parameters not mentioned.

4.1 Effect of Number of Positive Examples

To verify the influence of positive examples on the result, we set a different number of initial positive examples (positive decay is off), and the result is shown in Figure 3.

It can be seen that the best results can be achieved when the initial positive example P_0 is set to 3. Beyond 3, there is a decrease in the generation accuracy due to the smaller Jaccard score of the retrieved sample. The performance drop beyond 3 positive examples is mainly due to the reduced relevance of additional positive examples. As more positive examples are added, their similarity to the current query decreases, introducing noise that misleads the model and lowers generation accuracy.

Method	SPIDER		BIRD			
	Dev	TEST	Dev		Test	
	EX(%)	EX(%)	EX(%)	VES(%)	EX(%)	VES(%)
DIN-SQL + GPT-4	82.80	85.30	50.72	58.79	55.90	59.44
DAIL-SQL + GPT-4	84.40	86.60	54.76	56.08	57.41	61.95
TA-SQL + GPT-4	85.00	–	56.19	–	59.14	–
MAC-SQL + GPT4	86.75	82.80	57.56	58.76	59.59	67.68
Ours(GPT-4)	87.04	86.35	62.25	65.62	–	–
Ours(GPT-4o)	**87.13**	**87.37**	**64.73**	**66.48**	**65.23**	**70.75**

Table 1: Result on SPIDER and BIRD, "–" indicates that the result is not provided

Method	Sim(%)	Mod(%)	Chall(%)	Total(%)
GPT-3.5	47.56	22.36	18.05	37.15
GPT-3.5 + SCL	57.08 (+9.52)	31.82 (+9.46)	20.83 (+2.78)	46.02 (+8.87)
GPT-4	54.27	34.62	31.94	46.21
GPT-4 + SCL	64.00 (+9.73)	63.44 (+28.82)	47.22 (+15.28)	62.25 (+16.04)
GPT-4o	58.59	43.53	40.68	52.99
GPT-4o + SCL	70.05 (+11.46)	59.05 (+15.52)	48.97 (+8.29)	64.73 (+11.74)

Table 2: Execution accuracy across different LLMs. 'Sim', 'Mod', and 'Chall' denote simple, moderate, and challenging subset samples. Numbers in parentheses indicate the relative improvement brought by SCL.

BIRD	EX(%)
SCL-SQL	**64.73**
w/o schema linking	62.58
w/o question split	63.23
w/o thought loop	58.34
w/o positive	60.16
w/o negative	59.32
w/o positive decay	62.23
w/o vote	61.47

Table 3: Result of ablation study. Where, w/o question split means that the question is no longer split, and the complete question is directly sent to the LLM. w/o thought loop indicates that the loop is closed and the method is executed linearly. w/o positive and w/o negative indicates that the positive example and negative example are not set respectively. w/o positive decay disables positive decay. w/o vote disables the voting mechanism and takes the last loop result as output.

4.2 Effect of Times of Loop

We set different values of L to examine the impact of loop execution times on overall performance and subset results (Sim, Mod, Chall), as shown in Figure 4.

Accuracy increases from $L = 1$ to $L = 3$, indicating that moderate iterations enhance the model's self-correction ability and decision quality. However, performance declines when $L > 3$, likely due to increased candidate pool complexity and noise interference.

Subset analysis reveals that performance on the simple subset remains relatively stable, while the moderate subset shows moderate improvements. The most significant gain is observed in the challenging subset, where accuracy peaks at $L = 3$. Beyond this point, performance decreases, suggesting that excessive iterations may introduce misleading information and hinder final judgment.

4.3 Selection Accuracy

Since the selected result from the candidate pool is not always correct, we further evaluated the accuracy of the selection mechanism. Specifically, we analyzed the loop execution behavior at $L = 3$, as shown in Figure 5, where "n-Loop" denotes samples that exit the loop at the n-th iteration. Figure 6 presents the statistics of selection errors (C-Error) at different loop stages.

It can be seen that selection errors (C-Error) account for only 3.1% of the total errors, indicating that the selection mechanism introduces relatively minor inaccuracies. Nevertheless, the loop mechanism still leads to a net performance gain, as the overall accuracy improves significantly from 58.34% to 64.73%. This demonstrates that although some selection errors occur, the overall effect of loop-based refinement remains positive.

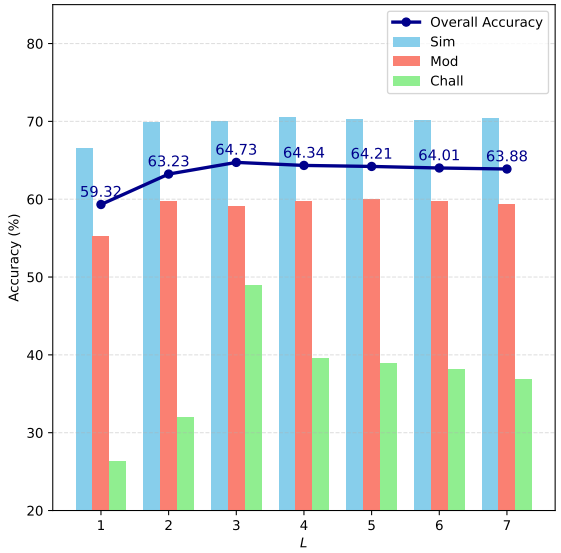

Figure 4: Result on the different number of loop cycles. The line represents the overall accuracy, and the blue, red, and green bars indicate the accuracy on the 'Sim', 'Mod', and 'Chall' subsets at different difficulty levels.

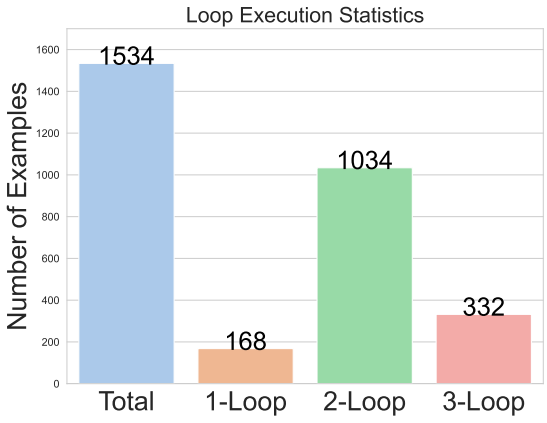

Figure 5: Loop execution statistics.

5 Related Work

5.1 Text-to-SQL

5.1.1 Design Method

The method based on the design method focuses on enhancing the relationship between text and database schema to improve the quality of SQL generation.

RAT-SQL(Wang et al., 2020) improved the schema encoding and feature representation in the encoder through the relational awareness self-attention mechanism and improved the generation accuracy. In addition, (Hui et al., 2021) proposes a multi-task text-to-SQL model (SDSQL) guided by schema dependency to capture the question interaction with the database schema without having to perform booting, significantly reducing inference time. LGESQL(Cao et al., 2021) enhances line graph coding and improves the parsing performance of heterogeneous graphs. S^2SQL(Hui et al., 2022) uses syntax-dependent information to strengthen the connection between the question and the database. Proton(Wang et al., 2022) extracted the relational structure from PLMs through the Poincare distance detection process to optimize the schema linking. RESDSQL(Li et al., 2023a) relieves the burden of schema linking in SQL parsing by decoupling schema linking from skeleton parsing.

5.1.2 Pre-training Method

Most existing Text-to-SQL pre-training methods use a single Transformer or Transformer-based encoder-decoder framework to capture task characteristics through different pre-training targets.

For example, TaBERT(Yin et al., 2020) performs well by combining natural language and tabular data representation. Similarly, GRAPPA(Yu et al., 2021) constructs and synthesizes data with SCFG and combines mask language model (MLM) pre-training to improve the table semantic parsing ability. Further, GAP(Zhao et al., 2022) enhanced the parsing ability through the joint learning of natural language and schematic representation. MIGA(Fu et al., 2023) designed four pre-training tasks based on T5(Raffel et al., 2020) to implement text-to-SQL. Similarly, GRAPHIX-T5(Li et al., 2023b) is enhanced by a graphic awareness layer. Codes(Li et al., 2024) employs incremental pre-training and data enhancement techniques to deal with schema linking and domain adaptation. Each method improves the performance of text-to-SQL parsing with different strategies.

5.1.3 Prompt-based Method

(Rajkumar et al., 2022; Liu et al., 2023) prove the advanced performance of various LLM on text-to-SQL. Then, around how to improve the performance of LLM in text-to-SQL, DIN-SQL(Pourreza and Rafiei, 2023) attempts to decompose complex text-to-SQL tasks into smaller subtasks to improve the performance of LLM in inference. DAIL-SQL(Gao et al., 2024) is a text-to-SQL solution that optimizes large language model prompts engineering, encodes SQL statements with structured knowledge, and reduces the impact of cross-domain knowledge to improve token efficiency. To eliminate the hallucination problem in LLM

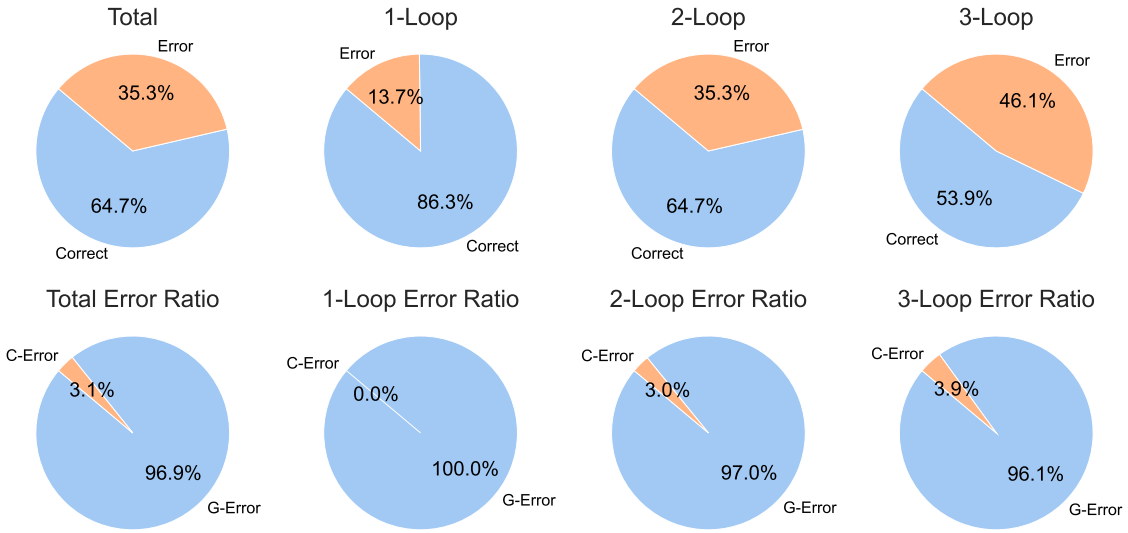

Figure 6: Error statistics. C-Error indicates an error caused by selection, and G-Error indicates an error caused by generation.

generation, TA-SQL(Qu et al., 2024) adjusted the model's processing methods for unfamiliar tasks by comparing them with previously trained tasks, reducing the model's dependence on the generalization ability to generate responses from scratch, thus significantly reducing the incidence of hallucination. MAC-SQL(Wang et al., 2023) further exploits the inference ability of LLM through serial collaboration of multiple agents.

5.2 Prompt Engineering

Although LLM have a strong understanding and inference ability, they still have difficulties in complex logical tasks such as mathematical operations and structured output. To this end, the research focus has shifted to solving problems step by step by prompt engineering guide LLM.

In this type of work, the chain of thought (CoT)(Wei et al., 2022) enhances complex reasoning ability through intermediate reasoning steps. The tree of thought (ToT)(Yao et al., 2023a) forms a tree-like structure through multiple reasoning paths to expand the search space. A mind map (GoT)(Besta et al., 2024) forms a network of crossing paths to generate more flexible solutions. ReAct(Yao et al., 2023b) proposes a general paradigm that combines reasoning and action with LLM. Prompt-based text-to-SQL methods often combine such prompt engineering to harness the potential of LLM.

6 Conclusion

We proposed a text-to-SQL method based on a self-contrastive loop of thought to solve the problem of LLM' dependence on few-shot example construction in text-to-SQL tasks. By designing the generation process as a discriminant loop structure based on a comparison of positive and negative examples, we significantly improved the accuracy and generalization of LLM and reduced the dependence on small samples. Experimental results show that the proposed method performs well on multiple data sets, and the generated SQL queries have higher accuracy. The contribution of this paper also includes the design of a question split strategy, execution, logical verification mechanism, and introduction of the "loop" structure to optimize the self-verification and error-correcting ability of LLM in text-to-SQL tasks.

7 Limitation

There are still limitations in the SCL-SQL method. The first is that it solves the problem of constructing few-shot examples in the generation process, but prompts of some complexity are still necessary. The second point is that the effect depends on the number and quality of the dataset. If the data set is too small or there are not enough similar examples, then the effect of this method will be greatly limited.

References

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 17682–17690. AAAI Press.

Ben Bogin, Jonathan Berant, and Matt Gardner. 2019. Representing schema structure with graph neural networks for text-to-sql parsing. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4560–4565. Association for Computational Linguistics.

Ricardo Campos, VÃ■tor Mangaravite, Arian Pasquali, AlÃ■pio Jorge, CÃl'lia Nunes, and Adam Jatowt. 2020. Yake! keyword extraction from single documents using multiple local features. *Information Sciences*, 509:257–289.

Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: line graph enhanced text-to-sql model with mixed local and non-local relations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 2541–2555. Association for Computational Linguistics.

Zhi Chen, Lu Chen, Hanqi Li, Ruisheng Cao, Da Ma, Mengyue Wu, and Kai Yu. 2021. Decoupled dialogue modeling and semantic parsing for multi-turn text-to-sql. *CoRR*, abs/2106.02282.

Ahmed Elgohary, Saghar Hosseini, and Ahmed Hassan Awadallah. 2020. Speak to your parser: Interactive text-to-sql with natural language feedback. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 2065–2077. Association for Computational Linguistics.

Yingwen Fu, Wenjie Ou, Zhou Yu, and Yue Lin. 2023. MIGA: A unified multi-task generation framework for conversational text-to-sql. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 12790–12798. AAAI Press.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2023. Re-appraising the schema linking for text-to-SQL. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 835–852, Toronto, Canada. Association for Computational Linguistics.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: program-aided language models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 10764–10799. PMLR.

Zihui Gu, Ju Fan, Nan Tang, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Sam Madden, and Xiaoyong Du. 2023. Interleaving pre-trained language models and large language models for zero-shot NL2SQL generation. *CoRR*, abs/2306.08891.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Bowen Li, Jian Sun, and Yongbin Li. 2022. S^2sql: Injecting syntax to question-schema interaction graph encoder for text-to-sql parsers. *CoRR*, abs/2203.06958.

Binyuan Hui, Xiang Shi, Ruiying Geng, Binhua Li, Yongbin Li, Jian Sun, and Xiaodan Zhu. 2021. Improving text-to-sql with schema dependency learning. *CoRR*, abs/2103.04399.

Shima Imani, Liang Du, and Harsh Shrivastava. 2023. Mathprompter: Mathematical reasoning using large language models. In *Proceedings of the The 61st Annual Meeting of the Association for Computational Linguistics: Industry Track, ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 37–42. Association for Computational Linguistics.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. RESDSQL: decoupling schema linking and skeleton parsing for text-to-sql. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 13067–13075. AAAI Press.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan,

Cuiping Li, and Hong Chen. 2024. Codes: Towards building open-source language models for text-to-sql. *Proc. ACM Manag. Data*, 2(3):127.

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023b. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 13076–13084. AAAI Press.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023c. Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 4870–4888. Association for Computational Linguistics.

Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S. Yu. 2023. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *CoRR*, abs/2303.13547.

Linyong Nan, Yilun Zhao, Weijin Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, and Dragomir Radev. 2023a. Enhancing few-shot text-to-sql capabilities of large language models: A study on prompt design strategies. *CoRR*, abs/2305.12586.

Linyong Nan, Yilun Zhao, Weijin Zou, Narutatsu Ri, Jaesung Tae, Ellen Zhang, Arman Cohan, and Dragomir Radev. 2023b. Enhancing text-to-sql capabilities of large language models: A study on prompt design strategies. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 14935–14956. Association for Computational Linguistics.

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: decomposed in-context learning of text-to-sql with self-correction. *CoRR*, abs/2304.11015.

Mohammadreza Pourreza, Davood Rafiei, Yuxi Feng, Raymond Li, Zhenan Fan, and Weiwei Zhang. 2024. Sql-encoder: Improving NL2SQL in-context learning through a context-aware encoder. *CoRR*, abs/2403.16204.

Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. Before generation, align it! A novel and effective strategy for mitigating hallucinations in text-to-sql generation. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 5456–5471. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *CoRR*, abs/2204.00498.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: relation-aware schema encoding and linking for text-to-sql parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7567–7578. Association for Computational Linguistics.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023. MAC-SQL: A multi-agent collaborative framework for text-to-sql. *CoRR*, abs/2312.11242.

Lihan Wang, Bowen Qin, Binyuan Hui, Bowen Li, Min Yang, Bailin Wang, Binhua Li, Jian Sun, Fei Huang, Luo Si, and Yongbin Li. 2022. Proton: Probing schema linking information from pre-trained language models for text-to-sql parsing. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, pages 1889–1898. ACM.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Sun Yang, Qiong Su, Zhishuai Li, Ziyue Li, Hangyu Mao, Chenxi Liu, and Rui Zhao. 2024. Sql-to-schema enhances schema linking in text-to-sql. *CoRR*, abs/2405.09593.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 8413–8426. Association for Computational Linguistics.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir R. Radev, Richard Socher, and Caiming Xiong. 2021. Grappa: Grammar-augmented pre-training for table semantic parsing. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 3911–3921. Association for Computational Linguistics.

Chen Zhao, Yu Su, Adam Pauls, and Emmanouil Antonios Platanios. 2022. Bridging the generalization gap in text-to-SQL parsing with schema expansion. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5568–5578, Dublin, Ireland. Association for Computational Linguistics.

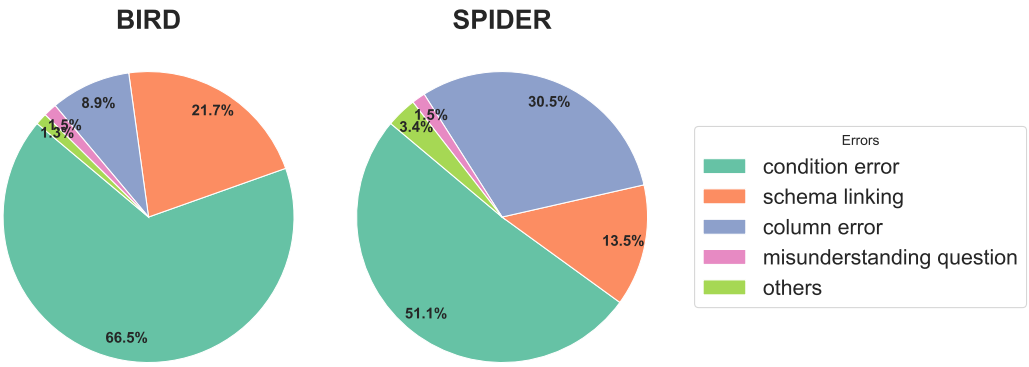

Figure 7: Error distribution statistic on BIRD and SPIDER dev set. Where "condition error" indicates wrong condition construction, "schema linking" indicates incorrect linking with tables, "column error" means the wrong column was selected, and "misunderstanding question" indicates the SQL was not compatible with question.

A Error Analysis

To fully explore our proposed approach, we also counted error types on the BIRD and SPIDER dev datasets in Figure 7. Among them, we mainly examine schema linking and semantic error. As the figure shows, the most common error types are conditional errors. Conditional errors account for more than 50% of total errors in BIRD and SPIDER. Compared to past methods, we have reduced error rates in many aspects, but we still lack advantages in constructing SQL for complex conditions. We give examples of different error types in Tables 4, 5, 6, 7, and 8.

Schema Linking Error	
BIRD	
Question:	How many schools in Fresno (directly funded) have number of test takers not more than 250?
PRED:	`SELECT COUNT(*) FROM schools AS s INNER JOIN satscores AS ss ON s.CDSCode = ss.cds WHERE s.City = 'Fresno' AND s.FundingType = 'Directly funded' AND ss.NumTstTakr <= 250`
GOLD:	`SELECT COUNT(T1.CDSCode) FROM frpm AS T1 INNER JOIN satscores AS T2 ON T1.CDSCode = T2.cds WHERE T1.‘Charter Funding Type‘ = 'Directly funded' AND T1.‘County Name‘ = 'Fresno' AND T2.NumTstTakr <= 250`
SPIDER	
Question:	What are the locations and names of all stations with capacity between 5000 and 10000?
PRED:	`SELECT location, station_name FROM stations WHERE capacity BETWEEN 5000 AND 10000`
GOLD:	`SELECT LOCATION , name FROM stadium WHERE capacity BETWEEN 5000 AND 10000`

Table 4: Examples of schema linking errors.

Condition Error	
BIRD	
Question:	Please list the phone numbers of the direct charter-funded schools that are opened after 2000/1/1.
PRED:	`SELECT schools.Phone FROM frpm INNER JOIN schools ON frpm.CDSCode = schools.CDSCode WHERE frpm.‘Charter School (Y/N)‘ = 1 AND frpm.‘Charter Funding Type‘ = 'Directly funded' AND schools.OpenDate > '2000-01-01' AND schools.Phone IS NOT NULL AND schools.OpenDate IS NOT NULL AND schools.Charter IS NOT NULL;`
GOLD:	`SELECT T2.Phone FROM frpm AS T1 INNER JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode WHERE T1.‘Charter Funding Type‘ = 'Directly funded' AND T1.‘Charter School (Y/N)‘ = 1 AND T2.OpenDate > '2000-01-01'`
SPIDER	
Question:	What is the maximum capacity and the average of all stadiums?
PRED:	`SELECT MAX(Capacity) AS Max_Capacity, AVG(Average) AS Avg_Attendance FROM stadium`
GOLD:	`SELECT max(capacity), average FROM stadium`

Table 5: Examples of condition errors.

Column Error	
BIRD	
Question:	If there are any, what are the websites address of the schools with a free meal count of 1,900-2,000 to students aged 5-17? Include the name of the school.
PRED:	`SELECT frpm.‘School Name‘, schools.‘Web‘ FROM frpm INNER JOIN schools ON frpm.‘CDSCode‘ = schools.‘CDSCode‘ WHERE frpm.‘Free Meal Count (Ages 5-17)‘ BETWEEN 1900 AND 2000 AND schools.‘Website‘ IS NOT NULL;`
GOLD:	`SELECT T2.Websit, T1.‘School Name‘ FROM frpm AS T1 INNER JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode WHERE T1.‘Free Meal Count (Ages 5-17)‘ BETWEEN 1900 AND 2000 AND T2.Website IS NOT NULL`
SPIDER	
Question:	Find the average and maximum age for each type of pet.
PRED:	`SELECT Pets.PetType, AVG(Pets.pet_age) AS average_age, MAX(Pets.pet_age) AS maximum_age FROM Pets JOIN Has_Pet ON Pets.PetID = Has_Pet.PetID GROUP BY Pets.PetType;`
GOLD:	`SELECT avg(pet_age) , max(pet_age) , pettype FROM pets GROUP BY pettype`

Table 6: Examples of column errors.

Question misunderstanding	
BIRD	
Question:	Of the schools that offers a magnet program serving a grade span of Kindergarten to 8th grade, how many offers Multiple Provision Types? List the number of cities that offers a Kindergarten to 8th grade span and indicate how many schools are there serving such grade span for each city.
PRED:	`SELECT COUNT(*) AS NumberOfSchoolsWithMultipleProvisionTypes FROM schools JOIN frpm ON schools.CDSCode = frpm.CDSCode WHERE schools.Magnet = 1 AND schools.GSserved = ’K-8’ AND frpm.‘NSLP Provision Status‘ = ’Multiple Provision Types’; - Second part: List the number of cities and the number of schools serving a K-8 grade span SELECT schools.City, COUNT(*) AS NumberOfSchools FROM schools WHERE schools.GSserved = ’K-8’ GROUP BY schools.City;`
GOLD:	`SELECT T2.City, COUNT(T2.CDSCode) FROM frpm AS T1 INNER JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode WHERE T2.Magnet = 1 AND T2.GSoffered = ’K-8’ AND T1.‘NSLP Provision Status‘ = ’Multiple Provision Types’ GROUP BY T2.City`
BIRD	
Question:	Which model has the most version(make) of cars?
PRED:	`SELECT ml.Model, COUNT(DISTINCT cn.Make) AS make_count FROM model_list ml JOIN car_names cn ON ml.Model = cn.Model GROUP BY ml.Model ORDER BY make_count DESC LIMIT 1;`
GOLD:	`SELECT Model FROM CAR_NAMES GROUP BY Model ORDER BY count(*) DESC LIMIT 1;`

Table 7: Example of question misunderstanding errors.

Other Error	
BIRD	
Question:	Among the customers who paid in euro, how many of them have a monthly consumption of over 1000?
PRED:	`SELECT COUNT(DISTINCT yearmonth.CustomerID) FROM customers INNER JOIN yearmonth ON customers.CustomerID = yearmonth.CustomerID WHERE customers.Currency = 'EUR' AND yearmonth.Consumption > 1000`
GOLD:	`SELECT COUNT(*) FROM yearmonth AS T1 INNER JOIN customers AS T2 ON T1.CustomerID = T2.CustomerID WHERE T2.Currency = 'EUR' AND T1.Consumption > 1000.00`
BIRD	
Question:	List the first and last name of all players in the order of birth date.
PRED:	`SELECT first_name, last_name FROM players ORDER BY birth_date ASC`
GOLD:	`SELECT first_name , last_name FROM players ORDER BY birth_date`

Table 8: Examples of other errors.