MCTS-SQL: A Practical Framework for Text-to-SQL with Monte Carlo Tree Search

Anonymous ACL submission

Abstract

Text-to-SQL is a fundamental yet challenging task in the NLP area, aiming at translating natural language questions into SQL queries. While recent advances in large language models have greatly improved performance, most existing approaches depend on models with tens of billions of parameters or costly APIs (e.g. ChatGPT or Gemini), limiting their applicability in resource-constrained real-world environments. Therefore, enabling the light-weight models for Text-to-SQL is of great practical significance. However, smaller LLMs often struggle with complex user intent understanding, schema linking and syntax correctness. To address these challenges, we propose **MCTS-SQL**, a novel framework that uses Monte Carlo Tree Search (MCTS) to guide SQL generation through multi-step refinement. Since the light-weight models' weak performance of single-shot prediction, we generate better results through several trials with feedback. From another perspective, this mechanism also improve the model's reasoning ability to solve harder examples. Moreover, to further filter irrelevant information of databases, we designed an additional schema selector. Experiments results on the SPIDER and BIRD benchmarks demonstrate the effectiveness of our approach. Using a small open-source Qwen2.5-Coder-Instruct-1.5B, our method outperforms ChatGPT-3.5. And when we use GPT-4o as the base model, our method achieves a new **SOTA** execution accuracy **69.40%** on BIRD. Notably, our method achieves a significant performance improvement(51.48%) on the more challenging subset, outperforming the previous SOTA by **3.41%**.

1 Introduction

Text-to-SQL is a task aimed at converting natural queries into SQL, which plays a critical role in data analytics and supports a wide range of real-world applications(Shi et al., 2024; Liu et al., 2024). Recent advances in LLM(Team et al., 2023; Achiam et al., 2023) have significantly improved the performance of Text-to-SQL systems. However, most of these powerful methods(Wang et al., 2024; Lee et al., 2024; Alp Caferoğlu and Ulusoy, 2024) rely on extremely huge models or costly APIs, making them expensive and can not be used in real resource-constrained environments. This raises an important challenge: **How can we enable lightweight, poorly performing models to effectively handle Text-to-SQL tasks in real-world applications?**

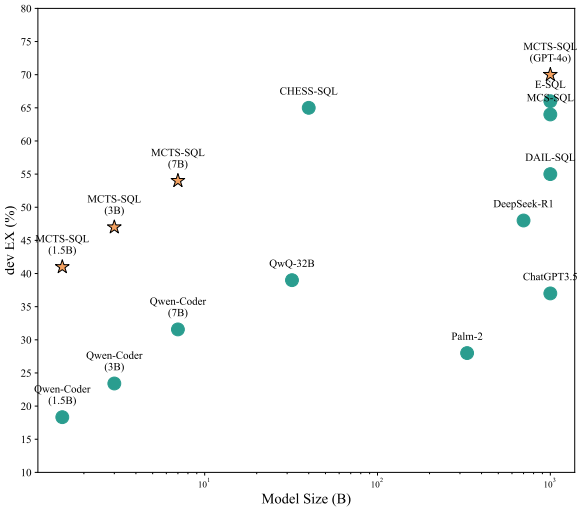

Figure 1: Execution accuracy comparison of MCTS-SQL across some existing methods. The results show that MCTS-SQL significantly enhances the performance of light-weight models, achieving performance comparable to some larger models. And when using GPT-4o as the base model, we achieve SOTA.

Most common errors made by these lower-performing LLMs is the mistake understanding of users' intent, wrong schema selection and syntax errors. Due to the poor performance of single-shot prediction, an intuitive way to address these challenges is to conduct a trial-and-feedback mechanism to iteratively optimize the generated SQL.

However, exploration without any constraints or guidance is inefficient. Therefore, a more powerful optimization strategy is needed to direct better solutions.Monte Carlo Tree Search (MCTS) has proven to be an efficient tool in decision-making and optimization tasks.Recent studies (Pitanov et al., 2023; Li et al., 2023a; Chen et al., 2024)have demonstrated that MCTS can be effectively applied to problems requiring iterative improvements. Given its strengths, MCTS presents a practical tool for optimizing SQL generation in Text-to-SQL tasks.

In this paper, we introduce **MCTS-SQL**, a novel framework that integrates MCTS with Text-to-SQL tasks. Our approach consists of three simple components:**Selector**,**Direct Generator**,and **MCTS-Refiner**. The Selector is used to filter the tables and fields that are most relevant to the users' intent, reduce the redundant interference. The Direct Generator produces the initial SQL, which is directly output if it meets the requirements. MCTS-refiner iteratively enhances the SQL through multiple trial-and-feedback steps.

We evaluate the performance of MCTS-SQL on two widely-used benchmarks: The Spider (Yu et al., 2018) and BIRD(Li et al., 2023b). The results show that MCTS-SQL based on the Qwen-2.5-Coder-Instruct-1.5B (Hui et al., 2024) outperforms ChatGPT-3.5, and 3B version achieves even better performance than some earlier GPT-4o based methods. Moreover, to explore the boundaries of our algorithm, we evaluate results based on GPT-4o. Unsurprisingly, we achieve SOTA, with execution accuracy of 69.40% on BIRD, particularly on challenging samples, surpassing existing SOTA by **3.41%**. The comparison across some existing methods can be seen in Figure 1.

The limitations of our approach are also quite evident. MCTS-SQL approximates better results through multiple attempts. Comparing to single-shot predictions, the method leads to more token and time consumption inevitably. Therefore, we believe that the best practice of MCTS-SQL is to use light-weight models to address Text-to-SQL tasks, which is also our motivation. And it is undeniable that even for more powerful models, MCTS-SQL still enhances their performance.

The main contributions of our proposed MCTS-SQL can be summarized as follows:

- To our best knowledge, we are the first to apply Monte Carlo Tree Search(MCTS) to the Text-to-SQL task, aiming to achieve usable performance with models that have a small number of parameters.

- We propose a novel Text-to-SQL framework, MCTS-SQL, which consists of three key modules: Selector (to filter relevant tables and fields), Direct Generator (to generate initial SQL), and MCTS-Refiner (to iteratively enhance SQL).

- Our experiments on the Spider and BIRD datasets demonstrate that MCTS-SQL outperforms existing methods, particularly on difficult queries. We achieve SOTA performance with GPT-4o.

2 Related Work

In this section, we provide an overview of related work on Text-to-SQL and Monte Carlo Tree Search, highlighting their relevance and main differences to our proposed research.

2.1 Text-to-SQL

Text-to-SQL aims to bridge natural language queries and structured database queries, and numerous approaches are proposed to address its challenges. Early systems, such as LUNAR(Kang et al., 2012) and NaLIX(Hammami et al., 2021), employed rule-based methods that manually crafted grammar rules and heuristics. However, the generalization performance of these methods across different tasks or databases is difficult to guarantee.

The deep learning marked a turning point for Text-to-SQL. End-to-end models like Seq2SQL(Zhong et al., 2017) and SQL-Net(Katsogiannis-Meimarakis and Koutrika, 2021) directly mapped natural language to SQL but struggled with complex queries, especially those involving nested structures or intricate reasoning. Pre-trained Language Models (PLMs), such as TaBERT(Katsogiannis-Meimarakis and Koutrika, 2023) and BERT-SQL(Guo and Gao, 2019), enhance cross-domain generalization and improve the accuracy of SQL generation. However, these methods require a certain amount of domain-specific SQL training data, which makes them difficult to land in practical applications

Recently, Large Language Models (LLMs) such as GPT-4(Achiam et al., 2023), Palm-2(Anil et al., 2023), and LLaMA(Touvron et al., 2023) have

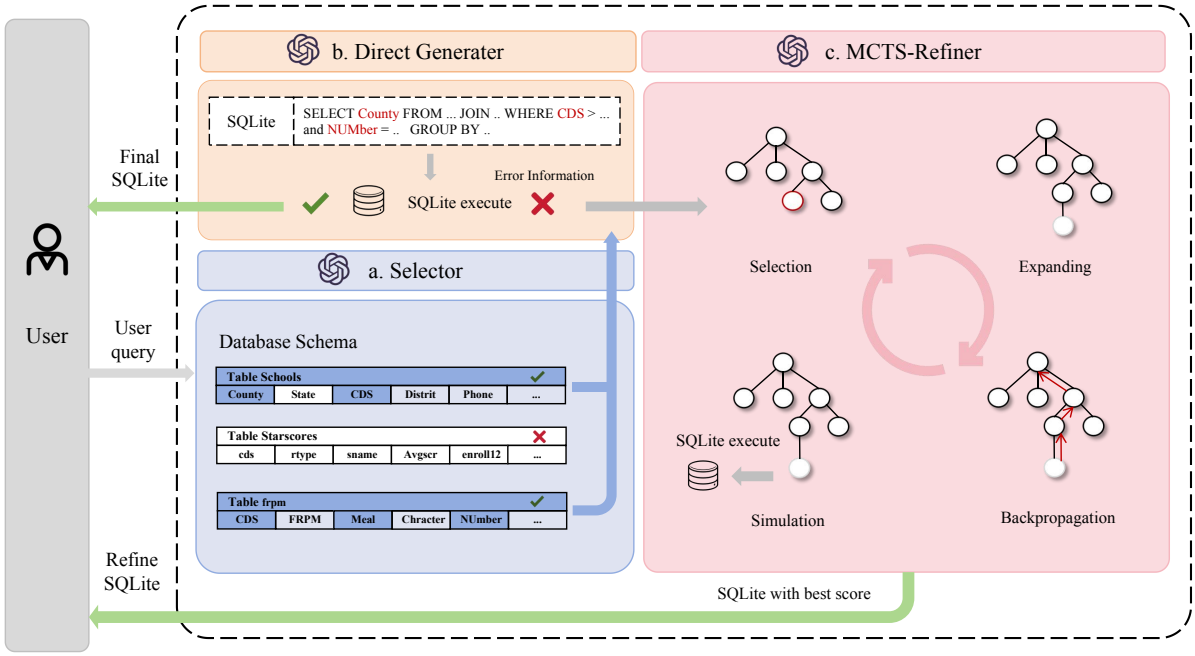

Figure 2: The MCTS-SQL framework consists of three core components: the **Selector**, the **Direct Generator** and the **MCTS-Refiner**. The Selector is used to filter the most relevant tables and columns based on the user's intent. The Direct Generator aims to produce an initial SQL query. And the MCTS-Refiner is activated when the initial SQL fails to meet the requirement, which adopts iterative trial-and-feedback optimization to refine the query progressively.

revolutionized Text-to-SQL tasks. These methods excel in zero-shot and few-shot settings without any extensive training data.(Lee et al., 2024; Talaei et al., 2024; Alp Caferoğlu and Ulusoy, 2024). DAIL-SQL(Gao et al., 2023) optimized prompt engineering, focusing on question representation, prompt structure, and example selection to enhance SQL accuracy with minimal supervision. Frameworks like C3-SQL(Dong et al., 2023), DIN-SQL(Pourreza and Rafiei, 2024), and StructGPT(Jiang et al., 2023) further advanced the field by addressing complex queries through database simplification, query decomposition, and structured data access. Additionally, MAC-SQL(Wang et al., 2024) introduced a collaborative framework integrating decomposer, auxiliary selector, and refiner modules for iterative SQL refinement.

From the analysis of existing methods, it becomes clear that advancing Text-to-SQL performance relies on the models' understanding and reasoning capabilities. However, these models are often of large scale and costly, making them impractical for real-world resource-constrained application. Our core motivation is to enable lightweight models to achieve practical performance in Text-to-SQL tasks. To this end, we incorporate Monte Carlo Tree Search (MCTS) to guide the generation.

2.2 Monte Carlo Tree Search

MCTS is widely used for planning complex problems, and a large number of downstream experiments have demonstrated its effectiveness. For example, (Pitanov et al., 2023) demonstrates its benefits in multi-agent path search, highlighting the advantages over traditional heuristic search methods. Similarly, (Li et al., 2023a) use MCTS to effectively address various types of SAT problems. Recently, combining MCTS with large-scale language models has been a great trend. (Chen et al., 2024) proposed IMCTS, an approach designed to enhance the mathematical reasoning capabilities of fine-tuned LLMs. (Xu, 2023) integrated MCTS with a lightweight energy function, demonstrating notable performance improvements. In addition, MCTSr(Di Zhang et al., 2024) introduced systematic exploration and heuristic self-refinement mechanisms, further advancing its applications in complex decision-making tasks.

Building on these successes, our work is the first to introduce Monte Carlo Tree Search (MCTS) into the Text-to-SQL domain.The core idea is very simple: reduce errors through iterative trial and

error. However, naive exhaustive attempts are inefficient and impractical. To address this, we leverage MCTS to find a more efficiently and reliable exploration path.

3 MCTS-SQL Framework

As shown in Figure 2, the MCTS-SQL framework consists of three key components: the Selector, Direct Generator, and MCTS-Refiner. The Selector filter relevant tables and schema elements based on the user query, while the Direct Generator produces an initial SQL query. Queries that fail or yield errors are refined by the MCTS-Refiner through iterative tree search. A detailed explanation of each component is provided in the subsequent section. Moreover, all three components use the training-free fresh shot agent method.

The collaboration process of our MCTS-SQL is presented in Algorithm 1:

Algorithm 1 The algorithm of MCTS-SQL

Input: query q, database schema db, knowledge kg

Output: SQL statement

db' = $LLM_{Selector}$(q, db, kg)
sql,err = $LLM_{DirectGenerator}$(q, db, kg)
ver = $LLM_{Verifier}$(sql,q,db,kg,)
if err is NULL and ver is ok **then**
  return sql
else
  count = 0
  while count < maxRollout **do**
    select a node
    cri = $LLM_{Critiquer}$(sql, err, q, db, kg)
    ref = $LLM_{Refiner}$(sql, err, q, db, kg, cri)
    score = $LLM_{Evaluater}$(ref, err, q, db, kg)
    back-propagation
    update the UCT value
  end while
  sql = ref with best score
  return sql
end if

3.1 Schema

Before introducing the specific components, we would like to describe the special design of effectively translating database structures. Combining the database schema information in the prompt is essential for enabling the LLM to comprehend the database structure accurately and generate precise queries. We present a novel method that illustrates the hierarchical relationships between databases, tables, and columns using a semi-structured format.

To be specific, we provide the table name and corresponding description for each table(which can be omitted if not necessary). The table information is converted into a list where each entry is a tuple containing a column of details. Each column includes the name, data type, description, and example values, thus providing a comprehensive view of its contents. In addition, foreign keys must be included to represent the relationships between tables accurately. Understanding hierarchical relationships is critical for query generation. An example of proposed schema format is shown in Figure 3. All the agents in this paper introduce database information through this schema.

Schema

【Database schema】
Table: frpm
[
 (CDSCode, CDSCode, TEXT, Value examples: ['01100170109835', '01100170112607'].),
 (Charter School (Y/N), Charter School (Y/N), TINYINT, Value examples: [1, 0, None]. And 0: N;. 1: Y)]
Table: satscores
[
 (cds, California Department Schools, TEXT, Value examples: ['10101080000000', '10101080109991'].),
 (sname, school name. TEXT, Value examples: ['None', 'Middle College High', 'John F. Kennedy High', 'Independence High', 'Foothill High'].),
 (NumTstTakr, INTEGER,Number of Test Takers in this school. Value examples: [24305, 4942, 1, 0, 280]),
 ]
【Foreign keys】
frpm.`CDSCode` = satscores.`cds`

Figure 3: An example of proposed database schema format.The format consists of table names, descriptions and column level details(name, data type, description, and examples) to represent the hierarchical information of databases.

3.2 Selector

The role of the Selector can be formally described as follows. Given an input triplet $\mathcal{X} = (Q, S, \mathcal{K})$, where Q is the query, $S = T, C$ is the database schema consisting of tables (T) and columns (C), and $\mathcal{K}$ denotes the knowledge provided. The Se-

lector aims to identify a minimal subset of tables and columns, denoted as $S' = T', C'$, which are necessary to answer the query Q. The behavior of the Selector is formally defined as follows:

$$S' = f_{\text{Selector}}(Q, S, \mathcal{K} \mid \mathcal{M}) \quad (1)$$

Where $f_{\text{Selector}}(\cdot \mid \mathcal{M})$ represents the Selector's function, implemented via prompt engineering powered by a large language model $\mathcal{M}$.

The design of the Selector is motivated by two key considerations: (i) Including unnecessary schema elements in the prompt increases the risk of irrelevant or extraneous items being incorporated into the generated SQL, which can degrade output quality; (ii) Directly utilizing the entire database schema may result in excessively long prompts, which could lead to higher computational costs and potentially exceed the input length limitations of the language model.

3.3 Direct Generator

The purpose of the Direct Generator is to generate SQL queries directly through an end-to-end process. It can be described as follows, where R represents the generated SQL query.

$$R = f_{\text{Direct Generator}}(Q, S', \mathcal{K} \mid \mathcal{M}) \quad (2)$$

After the SQL is generated, it follows two steps of evaluation. First, an executor checks its syntactic correctness and successful execution. Then, an LLM verifies if the SQL meets the user's requirements. The LLM-based verifier can be formalized as:

$$V = f_{\text{Verifier}}(R, Q, S', K \mid \mathcal{M}) \quad (3)$$

Specifically, the Direct Generator employs chain-of-thought prompting. (Wei et al., 2022) We assemble the relevant table and field information obtained from the Selector mentioned above with the user input. The LLM processes this input to generate SQL queries, accompanied by a detailed rationale. Additionally, we employ a few-shot learning strategy, using several in-context examples to improve the LLM's understanding of task-specific instructions and enhance its generalization capabilities.

3.4 MCTS-Refiner

Typically, SQL queries generated by the Direct Generator fail to meet task requirements due to syntactic errors or mismatch with the user's intent.

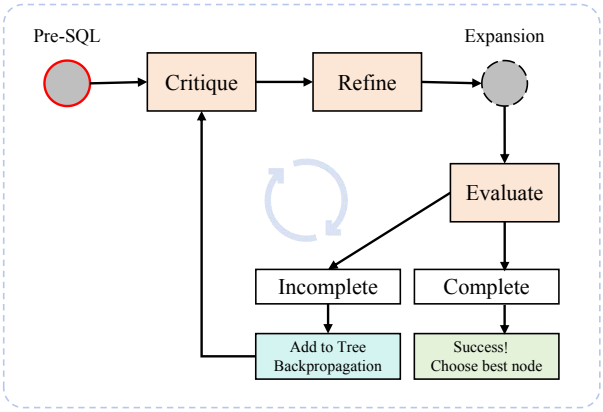

Figure 4: The main workflow of our proposed MCTS-refiner. The SQL generated in the last step is firstly get a critique. Then, based on the critique, a refinement is provided. The search tree is now expanded.If the iteration is complete, the node with best score is selected as the final output, otherwise, the node will be added to the search tree and backpropagation.

The MCTS-Refiner aims to refine SQL queries using the self-critique mechanism to optimize the query iteratively. The main workflow of the proposed method consists of several stages, detailed as follows:

Initialization: The root node is initialized with the suboptimal SQL generated by the Direct Generator as a reference for step-by-step optimization to reduce the complexity of the search process.

Selection: Following the existing practices, we define a function P to rank all generated SQL queries that are not fully expanded. The node with the highest value is selected for further refinement. The function P of a node a can be defined as follows, where r_a represents the set of results associated with node a.

$$P(a) = \frac{1}{2}\left(\min r_a + \frac{1}{|r_a|}\sum_{i=1}^{|r_a|} r_a^i\right) \quad (4)$$

Self-Refine: The SQL query a is initially executed by the executor to get the error information E_a, which is then used to refine the query through the self-refine framework. In this process, the LLM generates a critique c, serving as the guidance for refining the query and producing an improved SQL query a'. Specifically, E_a represents the error details related to the initial SQL query a, and I denotes the prompt used in the Direct Generator, which includes the input query Q, the database

schema S', and relevant knowledge K. The details can be formally described as follows:

$$c = f_{\text{Critiquer}}(a, E_a, I \mid \mathcal{M}) \quad (5)$$

$$a' = f_{\text{Refiner}}(a, c, E_a, I \mid \mathcal{M}) \quad (6)$$

The Self-refine module designed a refinement mechanism using error feedback and critique generation to enhance the accuracy and robustness of SQL queries.

Self-Evaluation: The refined SQL query is evaluated to obtain a reward value, denoted as r, and its corresponding P-value is computed. To be specific, we proposed a model-based self-reward feedback mechanism, with the reward value constrained within the range of -95 to 95. To ensure the reliability and fairness, the highest scores are deliberately suppressed. The reward r is formally defined as:

$$r_a = f_{\text{Evaluater}}(a', E_{a'}, I \mid \mathcal{M}) \quad (7)$$

Backpropagation: The value r of the refined SQL query is back-propagated through the search tree, updating the value information of the parent node and other relevant nodes. If the P-value of any child node is changed, the corresponding P-value of its parent node is recalculated accordingly. The process can be described as follows:

$$P'(a) = \frac{1}{2}\left(P(a) + \max_{i \in \text{a.}children} P(i)\right) \quad (8)$$

UCT update: Following the existing practice (Di Zhang et al., 2024), after updating the P values for all nodes, we choose the *UCT* function to measure the combined value of each node, which is used as an important basis for expansion in the next selection stage. The *UCT* value of a node a is formally defined as:

$$UCT_a = P(a) + c\sqrt{\frac{\ln N(\text{Father}(a)) + 1}{N(a) + \epsilon}} \quad (9)$$

In this formulation, $N(.)$ denotes the total number of visits to a given node, and c is a constant that balances the trade-off between P-value and visit times. The term ϵ is a small constant to prevent division by zero.

The algorithm proceeds through all these steps iteratively until the maximum rollout numbers are reached. And the SQL queries with the highest score r is chosen as the final output.

4 Experiments

To evaluate the performance of our MCTS-SQL, we present the implementation details, explain the experiments performed, and offer a thorough analysis of the results.

4.1 Datasets

We evaluate our MCTS-SQL framework using two Text-to-SQL benchmarks: Spider and BIRD. The Spider contains 7000 training question-query pairs and 1038 development pairs, which are from 200 different databases across 138 domains. In this study, we only concentrate on the development set. In contrast, BIRD includes 95 large-scale databases with high-quality Text-to-SQL pairs, covering 37 specialized domains. Unlike Spider, BIRD focuses on real-world database content and provides external knowledge reasoning to bridge natural language queries and database contextual information.

4.2 Evaluation Metrics

To evaluate our proposed method's performance, we use two metrics: Execution Accuracy(EX) and Valid Efficiency Score(VES).(Li et al., 2023b; Zhong et al., 2020) The Execution Accuracy(EX) calculates the percentage of queries where the predicted SQL queries match the correct SQL queries when executed. Valid Efficiency Score(VES) measures the percentage of predicted SQL queries that output sets consisting of the results from the ground-truth SQL queries.

4.3 Base Models

In this paper, we adopt Qwen2.5-Coder-Instruct series as our base models, given their leading performance in the filed of code generation. We evaluate the effectiveness of our framework across multiple model sizes, including 1.5B, 3B and 14B, to demonstrate its ability to enhance lightweight models. To further explore the boundary of our method, we also conduct experiment using more powerful closed APIs, including GPT-4o-mini and GPT-4o. In future work, we aim to distill the reasoning precess into a lightweight model, which currently implemented through MCTS refinement.

4.4 Hyper-parameters

In order to ensure the stability of our experiment results, we standardized the hyper-parameters as follows. The temperature is fixed at 0.1, the top-p parameter is set to 1.0, and the max-token length is

32168. As for the hyper-parameters in the MCTS-Refiners, the child nodes of a node are set to 2, and the max-rollout numbers are 5.

4.5 Experimental Results

4.5.1 A. BIRD Results

Table 1 presents a performance comparison of our method in the BIRD dataset against existing approaches. When using lightweight models(1.5B and 3B), MCTS-SQL outperforms ChatGPT-3.5 and even rivals some earlier methods based on GPT-4. This demonstrates that our method can be deployed on resource-constrained edge devices, without relying on large-scale models or costly APIs. Furthermore, when combined with the most powerful GPT-4o seties, MCTS-SQL achieves state-of-the-art performance, reaching **69.4%** execution accuracy (EX) and **66.24%** value execution score (VES) on the development set, confirming its superiority over existing methods and its practical utility.

Table 2 shows the detailed performance across different complexity levels. Compared to the baseline, our method achieve significant improvements. Analyzing the results, we observe that the model improves more on simpler examples. This may because mistakes in these cases are mostly about syntax, and the MCTS-Refiner can fix them easily using its feedback-based editing process. However, with a stronger base model, MCTS can still bring clear improvements on harder examples by exploring different ways to fix the errors.

Method	dev EX	test EX	dev VES
Palm-2	27.38	33.04	-
ChatGPT-3.5	36.64	40.08	42.30
DIN-SQL+GPT-4	50.72	55.09	58.79
DAIL-SQL+GPT-4	54.76	57.41	56.08
MAC-SQL+GPT-4	59.39	59.59	66.39
PB-SQL,v1	60.50	64.84	60.36
MCS-SQL+GPT-4	63.36	65.45	61.23
CHESS	65.00	66.69	62.77
ByteBrain	65.45	68.87	-
ASKData+GPT-4o	65.19	65.62	60.25
E-SQL+GPT-4o	65.58	66.29	62.43
Ours+Qwen-1.5B	40.69	43.72	44.87
Ours+Qwen-3B	46.71	48.37	48.19
Ours+Qwen-7B	53.61	51.79	52.21
Ours+GPT-4o-mini	63.15	61.39	60.78
Ours+GPT-4o	**69.40**	**68.91**	**66.24**

Table 1: Comparison with the results of existing methods on BIRD of the Execution accuracy and Valid Efficiency Score.The Qwen models in this table are Qwen-Coder-Instruct.

Method	Simp.	Mod.	Chall.	All
Qwen-1.5B	15.36	14.96	9.78	14.71
Qwen-3B	19.24	16.18	12.49	17.68
Ours + Qwen-1.5B	46.36	34.96	22.78	40.69
Ours + Qwen-3B	53.74	39.62	24.55	46.71
Ours + Qwen-7B	62.98	42.21	30.28	53.61
Ours+GPT-4o-mini	68.56	57.76	45.83	63.15
Ours+GPT-4o	**74.32**	**65.17**	**51.48**	**69.40**

Table 2: Execution accuracy in BIRD development set. The Qwen models in this table are Qwen-Coder-Instruct.

4.5.2 B. Spider Results

Table 3 presents the performance comparison on the Spider dataset. When using lightweight models, our method achieves results that are already practically usable. Furthermore, when equipped with GPT-4o as the base model, MCTS-SQL achieves outstanding performance, reaching **88.71%** on the development set and **86.63%** on the test set. While existing approaches have already demonstrated strong results on this benchmark, our method continues to deliver highly competitive performance.

Method	EX(Dev)	EX(Test)
C3+ChatGPT	81.80	82.30
DIN-SQL+GPT-4	82.80	85.30
DAIL-SQL+GPT-4	84.40	86.60
MAC-SQL+GPT-4	86.75	82.80
CHESS	87.2	-
MCS-SQL+GPT-4	89.5	89.6
Ours + Qwen-1.5B	67.45	71.68
Ours + Qwen-3B	74.03	73.98
Ours+GPT-4o-mini	86.16	83.74
Ours+GPT-4o	**88.71**	**86.63**

Table 3: Execution accuracy on both dev and test set of spider. The Qwen models in this table are Qwen-Coder-Instruct.

4.6 Ablation study

We conduct the ablation study to evaluate the contributions of the key components in our proposed MCTS-SQL. As shown in Table 4, each component plays a critical role in the overall performance. To better show the effects of each module, we use GPT-4o-mini as the base model, as lightweight models are often too weak to produce stable and meaningful performance differences. Specifically, replacing our structured Schema representation with a con-

Pipeline	Simp.	Mod.	Chall.	All
Ours + GPT-4o-mini	**68.56**	**57.76**	**45.83**	**63.15**
w/o schema	66.98	56.27	43.82	61.56
w/o Selector	66.71	55.40	40.18	60.79
w/o Direct Generator	67.15	56.63	44.82	61.86
w/o MCTS-Refiner	64.46	52.00	35.44	56.75

Table 4: Ablation study using GPT-4o-mini with EX on the development set.

Pipeline	Simp.	Mod.	Chall.	All
Ours + GPT-4o-mini	**68.56**	**57.76**	**45.83**	**63.15**
max-rollouts-5	68.56	57.76	45.83	63.15
max-rollouts-6	68.14	57.81	44.91	62.83
max-rollouts-7	68.42	57.12	45.93	62.88
child nodes-2	68.56	57.76	45.83	63.15
chile nodes-3	68.62	57.57	45.83	63.13

Table 5: Ablation study of hyper-parameters in MCTS with EX on the development set.

ventional DDL format leads to a slight performance drop. The Selector proves especially useful on both the moderate and challenging subsets, effectively filtering irrelevant schema information. The most significant degradation occurs when the MCTS-Refiner is removed, highlighting its essential role in enhance performance.

In addition, we analyze the sensitivity of two key hyper-parameters in the Monte Carlo Tree Search: the number of child nodes and the maximum number of rollouts. Table 5 presents the detailed results. We observe that variations in these parameters have minimal impact on overall performance—for example, increasing the number of rollouts or child nodes leads to only marginal changes. Based on these findings, we adopt the most efficient configuration to reduce token consumption.

5 Discussion

We conducted an error analysis of the single-prediction model and found that 42% of the errors were caused by syntax mistakes, wrong field selection, or misunderstanding of the schema. MCTS-SQL improves Text-to-SQL performance by using a trial-and-error feedback mechanism to guide the search for better SQL queries. Instead of relying on single-shot generation process, it explores multiple candidate SQL queries and keeps those that are actually effective based on execution results. This allows the method to recover from some generation errors.

6 Conclusion

In conclusion, this paper introduce MCTS-SQL, a novel framework to improve the Text-to-SQL performance of light-weight models. By using Monte Carlo Tree search, we design an iterative mechanism to enhance the qualities of SQL. Our approach, consisting of three modules-Selector, Direct Generator, and MCTS-Refiner, achieves significant improvements over existing methods. Experiments on the SPIDER and BIRD benchmarks show that, using only a 1.5B model, MCTS-SQL outperforms ChatGPT-3.5. And when we use the latest GPT-4o, we achieve SOTA. These results demonstrate that MCTS-SQL can significantly enhance the performance of small, resource-efficient models, making it a practical solution for real-world applications.

7 Limitations

Although MCTS-SQL is effective in enabling light-weight models to be practically applied into the Text-to-SQL task, its limitations are also evident.

The trial-and-error feedback mechanism of Monte Carlo Tree Search is essentially an approach that approximates better results through multiple attempts, which inevitably leads to significantly token consumption and longer inference time. Table 6 shows the average tokens consumption and inference time per task for single-shot inference and MCTS-SQL based on the Qwen-2.5-Coder-instruct-1.5B.

Method	Tokens	Time	Ex on BIRD
single-shot	197	0.63s	14.71
MCTS-SQL	2274	6.12s	40.69

Table 6: Comparison of tokens consumption and inference time per task.

MCTS-SQL uses significantly more tokens and inference time compared to single-shot prediction, but is also achieves substantial performance improvement. We believe this overhead is unavoidable if lightweight models are to be truly applied in real-world.

Moreover, the reasoning steps of MCTS rely heavily on prompt engineering, requiring a careful design and showing limited capability for rapid adaptation across tasks.

With the rise of reasoning models (Guo et al., 2025), implicit reasoning has become possible.

In future work, we plan to use MCTS-SQL to collect a set of data that contains reasoning processes, and distill a model with reasoning capabilities through reinforcement learning. This approach aims to develop a specialized Text-to-SQL model that achieves fast inference and strong robustness with fewer trial-and-error steps.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Hasan Alp Caferoğlu and Özgür Ulusoy. 2024. E-sql: Direct schema linking via question enrichment in text-to-sql. *arXiv e-prints*, pages arXiv–2409.

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, and Dmitry Lepikhin. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. 2024. Alphamath almost zero: process supervision without process. *arXiv preprint arXiv:2405.03553*.

Xiaoshui Huang Di Zhang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. 2024. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b: A technical report. *arXiv preprint arXiv:2406.07394*, 8.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, and 1 others. 2023. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306*.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Tong Guo and Huilin Gao. 2019. Content enhanced bert-based text-to-sql generation. *arXiv preprint arXiv:1910.07179*.

Linda Hammami, Alessia Paglialonga, Giancarlo Pruneri, Michele Torresani, Milena Sant, Carlo Bono, Enrico Gianluca Caiani, and Paolo Baili. 2021. Automated classification of cancer morphology from italian pathology reports using natural language processing techniques: A rule-based approach. *Journal of Biomedical Informatics*, 116:103712.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Structgpt: A general framework for large language model to reason over structured data. *arXiv preprint arXiv:2305.09645*.

Ning Kang, Bharat Singh, Zubair Afzal, Erik M van Mulligen, and Jan A Kors. 2012. Using rule-based natural language processing to improve disease normalization in biomedical text. *Journal of the American Medical Informatics Association*, 20(5):876–881.

George Katsogiannis-Meimarakis and Georgia Koutrika. 2021. A deep dive into deep learning approaches for text-to-sql systems. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2846–2851.

George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 32(4):905–936.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2024. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. *arXiv preprint arXiv:2405.07467*.

Anqi Li, Congying Han, Tiande Guo, Haoran Li, and Bonan Li. 2023a. General method for solving four types of sat problems. *arXiv preprint arXiv:2312.16423*.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2023b. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2024. A survey of nl2sql with large language models: Where are we, and where are we going? *arXiv preprint arXiv:2408.05109*.

Yelisey Pitanov, Alexey Skrynnik, Anton Andreychuk, Konstantin Yakovlev, and Aleksandr Panov. 2023. Monte-carlo tree search for multi-agent pathfinding: Preliminary results. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 649–660. Springer.

Mohammadreza Pourreza and Davood Rafiei. 2024. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36.

Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. 2024. A survey on employing large language models for text-to-sql tasks. *arXiv preprint arXiv:2407.15186*.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, and 1 others. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and 1 others. 2024. Mac-sql: A multi-agent collaborative framework for text-to-sql. *arXiv preprint arXiv:2312.11242*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Haotian Xu. 2023. No train still gain. unleash mathematical reasoning of large language models with monte carlo tree search guided by energy function. *arXiv preprint arXiv:2309.03224*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir 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*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-sql with distilled test suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 396–411.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.