# MCTS-SQL: Light-Weight LLMs Can Master the Text-to-SQL Through Monte Carlo Tree Search

**Shuozhi Yuan**[1]*, **Liming Chen**[1], **Miaomiao Yuan**[2], **Zhao Jin**[1]
[1]China Telecom Digital Intelligence
[2]Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
yuansz@chinatelecom.cn

## 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, limiting their applicability in resource-constrained environments. For real world, especially on edge devices, it is crucial for Text-to-SQL to ensure cost-effectiveness. Therefore, enabling the light-weight models for Text-to-SQL is of great practical significance. However, smaller LLMs often struggle with complicated user instruction, redundant schema linking or syntax correctness. To address these challenges, we propose MCTS-SQL, a novel framework that uses Monte Carlo Tree Search 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. However, directly applying MCTS-based methods inevitably leads to significant time and computational overhead. Driven by this issue, we propose a token-level prefix-cache mechanism that stores prior information during iterations, effectively improved the execution speed. Experiments results on the SPIDER and BIRD benchmarks demonstrate the effectiveness of our approach. Using a small open-source Qwen2.5-Coder-1.5B, our method outperforms ChatGPT-3.5. When leveraging a more powerful model Gemini 2.5 to explore the performance upper bound, we achieved results competitive with the SOTA. Our findings demonstrate that even small models can be effectively deployed in practical Text-to-SQL systems with the right strategy.

## 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 resource-constrained environments. For real-world use, especially on edge devices, running efficiently is crucial to keep low resources cost. This raises an important challenge: **How can we enable lightweight models to effectively handle Text-to-SQL tasks ?**

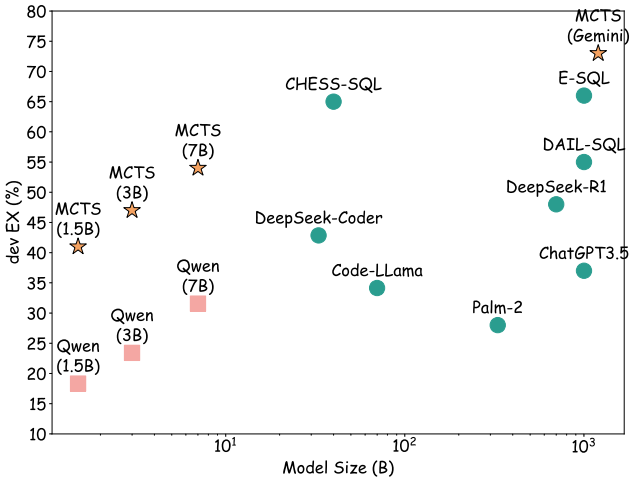


Figure 1: Execution accuracy comparison of MCTS-SQL across some existing methods. The results show that MCTS-SQL significantly enhances the performance of light-weight models, achieving performance comparable to some larger models. And when using Gemini2.5 as the base model, we achieve results competitive with the SOTA.

Most common errors made by these small 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, feedback 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

*Corresponding author

for optimizing SQL generation in Text-to-SQL tasks.

In this paper, we introduce MCTS-SQL, a practical approach that effectively applies MCTS to the Text-to-SQL task. Every components of our design follows one principle: **progressively reducing the search space to align with the limited reasoning capacity of light models.**

To realize this, we introduce three components:(i)**Selector** prunes irrelevant schema elements, reducing prompt complexity; (ii)**Direct Generator** provides a strong initial SQL candidate, avoiding deep search from scratch;(iii)**MCTS-Refiner** iteratively improves the SQL through guided exploration.

While effective, MCTS-based approaches introduce a new challenge: computation cost. Compared to single-shot prediction, iterative refinement inevitably increases token usage and inference time. Rethinking the whole pipeline of MCTS-SQL, we find that a large amount of inputs of every instructions are highly repetitive (e.g., database schema, field descriptions, few-shot examples). To address the latency from re-computing these inputs, we design a **prefix-cache mechanism**. Repeated inputs are computed once and cached, during MCTS iterations, only changing feedback and refinements are processed. This approach reduces inference time by **53%**.

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 Gemini-2.5. Unsurprisingly, we achieve competitive results, with execution accuracy of 72.91% on BIRD. The comparison across some existing methods can be seen in Figure 1.

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

- We propose **MCTS-SQL**, which apply Monte Carlo Tree Search to Text-to-SQL, demonstrating that small models can effectively handle this complex task.
- We design a search-space-reducing pipeline with Selector, Direct Generator, and MCTS-Refiner to guide SQL optimization.
- We introduce a novel **prefix-cache** mechanism to reduce redundant computation, significantly improving inference efficiency.
- Experiments on BIRD and SPIDER show that MCTS-SQL enables small models to achieve practical accuracy. For instance, using Qwen-1.5B, our method reaches **40.69%** EX on BIRD dev set. When leveraging a stronger model Gemini 2.5, it achieves **72.91%** EX on BIRD dev, which is competitive with the current SOTA.

## 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.

### 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, Xiong, and Socher 2017) and SQLNet (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 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. Additionally, MAC-SQL (Wang et al. 2024) introduced a collaborative framework integrating decomposer, auxiliary selector, and refiner modules for iterative SQL refinement. Recently, Xiyan-SQL(Gao et al. 2024), Chase-SQ(Pourreza et al. 2024)L, and DSAIR-SQL(Shkapenyuk et al. 2025) have also achieved improvements in NL2SQL performance through fine-tuning and sophisticated agent engineering. The most similar works, Alpha-SQL(Li et al. 2025) and SQL-o1(Lyu et al. 2025), perform heuristic, action-level search from scratch, whereas our MCTS-SQL adopts a progressive refinement strategy and leverages a prefix-cache mechanism to minimize computation during iterative exploration.

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.

### 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)

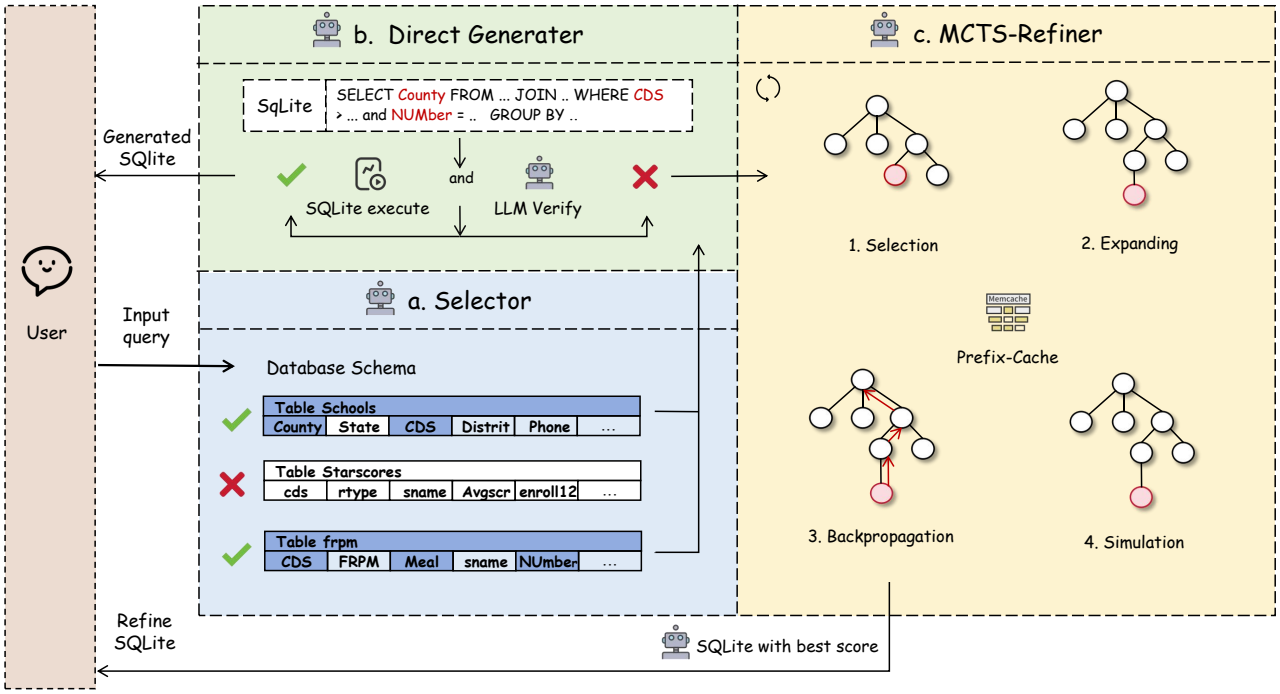


Figure 2: The MCTS-SQL framework consists of three core components: the **Selector**, the **Direct Generator** and the **MCTS-Refiner**. The Selector is used to filter the most relevant tables and columns based on the user's intent. The Direct Generator aims to produce an initial SQL query. And the MCTS-Refiner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

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 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. Moreover, we design a prefix-cache to avoid repeated computational overlead.

## 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. The collaboration process of our MCTS-SQL is presented in Algorithm 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. All the agents in this paper introduce database information through this schema.

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**

## 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 Selector 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:

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

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 Selector serves as a schema pruning module that leverages a large language model to identify tables and columns relevant to the query. Given $(Q, S, \mathcal{K})$, it interprets the query and associated knowledge to produce a focused subset $S' = T', C'$ necessary for answering $Q$. This process eliminates irrelevant schema elements, reducing the noise of schema and preventing prompt overflow caused by the full schema. By reducing the context, the Selector improves both the accuracy and efficiency of subsequent SQL generation.

## 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.

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

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:

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

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.

## 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. The MCTS-Refiner aims to refine SQL queries using the self-critique mechanism to optimize the query iteratively. The MCTS-Refiner is conditionally activated via two checks: (1) Execution Failure: If the generated SQL fails to run. (2) Semantic Error: If execution succeeds but an LLM-based Verifier identifies user's intent mismatch.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 $a$.

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

**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:

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

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

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

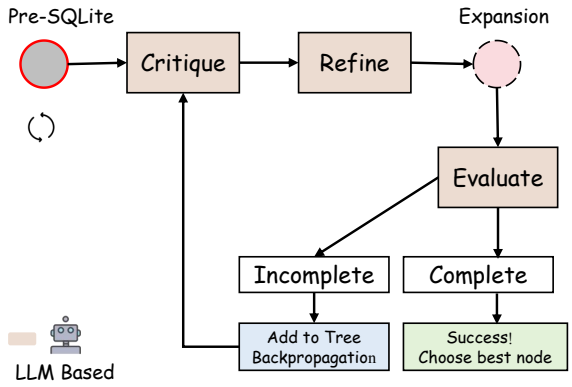


Figure 3: The main workflow of our proposed MCTS-refiner. The SQL generated in the last step is firstly get a critique. Then, based on the critique, a refinement is provided. The search tree is now expanded.If the iteration is complete, the node with best score is selected as the final output, otherwise, the node will be added to the search tree and backpropagation.

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:

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

**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:

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

**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:

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

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 $\epsilon$ 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.

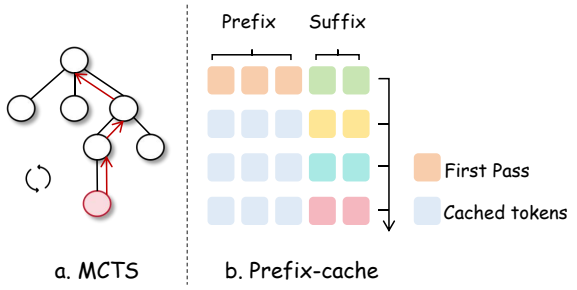


Figure 4: Illustration of the proposed optimization strategies. (a) MCTS-based iterative refinement for SQL generation. (b) Prefix-caching mechanism that reuses cached K/V states for invariant prompt components, reducing redundant computation and improving efficiency during multiple iterations.

## Prefix-Cache

Rethinking the MCTS iterations, we observed that a large part of input prompts remain unchanged, such as database schema, field descriptions, and few-shot examples. Recomputing these repeated components introduces unnecessary latency and computational overhead.

To address this issue, we design a prefix-caching mechanism (as can be seen in Fig.4) that reuses the intermediate K/V states of the transformer decoder layers:

- First Pass: When the model processes an input prefix for the first time, we cache the K/V states generated for decoder layer.
- Subsequent Passes: If the same prefix appears again, the cached K/V states are restored to skip redundant computation and focus on the new suffix.

In practice, we define the prefix as the invariant components(e.g.,field descriptions, few-shot examples), while the suffix consists of the parts that change during iterations, such as instructions, evaluations and execution feedback. To align with the prefix-caching mechanism, the overall prompt is structured as: dataset schema + few-shot demonstrations + specific instructions + feedback.

# 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. We evaluate our MCTS-SQL frameWORK using two Text-to-SQL benchmarks: Spider and BIRD.

## 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, Yu, and Klein 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)

| 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 |
| MCS-SQL+GPT-4 | 63.36 | 65.45 | 61.23 |
| Sql-o1 | 63.4 | 64.8 | - |
| 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 |
| Alpha-SQL | 69.70 | - | - |
| XiYan-SQL | 73.34 | 75.63 | - |
| Chase-SQL | 74.46 | 74.79 | - |
| DSAIR-SQL | 74.32 | 74.12 | - |
| **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 |
| Ours+Gemini2.5-pro | 72.91 | 74.74 | 72.66 |

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.

measures the percentage of predicted SQL queries that output sets consisting of the results from the ground-truth SQL queries.

## 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, GPT-4o and Gemini2.5. In future work, we aim to distill the reasoning precess into a lightweight model, which currently implemented through MCTS refinement.

## 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.

## Experimental Results

**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 Gemini2.5-pro, MCTS-SQL reaching **72.91%** execution accuracy (EX) and **72.66%** 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 | 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 |
| Ours+Gemini2.5-pro | 76.98 | 69.82 | 56.84 | 72.91 |

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

**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 **89.17%** on the development set and **88.74%** on the test set. While existing approaches have already demonstrated strong results on this benchmark, our method continues to deliver highly competitive performance.

## Prefix-Cache Effectiveness Evaluation

To evaluate the effectiveness of the proposed prefix-cache mechanism, we conducted controlled experiments focusing on three key aspects: (I) inference latency, (II) token computation reduction, and (III) execution accuracy impact. All experiments were performed on the BIRD . For a fair comparison, we used the same base model (Qwen2.5-Coder-Instruct-1.5B) under two settings: with prefix-cache and without prefix-cache. The detailed results can be seen in Table 4. The prefix-cache stores the intermediate key-value states for invariant parts of the input, such as database schema and few-shot examples. This enables the model to skip redundant computations during subsequent decoding steps. We measured inference time on 500 multi-turn SQL generation tasks. This pre-computation, though highly ef-

| 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 |
| XiYan-SQL | - | 89.65 |
| **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 |
| Ours+Gemeni2.5-pro | 89.17 | 88.74 |

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

| Setting | Avg Latency | Tokens | Ex Accuracy |
|---|---|---|---|
| single-shot | 0.63s | 197 | 14.71 |
| no-cache | 6.12s | 2274 | 40.69 |
| with-cache | 2.84s | 864 | 40.42 |

Table 4: Impact of Prefix-Cache on Latency, Token Usage, and Execution Accuracy

ficient, means the model's sampling process begins from a fixed distribution. We think that may occasionally guide the model toward a sub-optimal SQL generation path, especially when integrating new, dynamic feedback from the suffix. However, we consider the minor performance is an acceptable trade-off.

## Ablation Study

To evaluate the role of each component in MCTS-SQL, we perform an ablation study using Qwen-1.5B as the base model (Table 5). The results confirm that all modules are essential and reflect a core design principle: progressively reducing the search space to mitigate the limited reasoning capacity of lightweight models.

Removing the Selector lowers overall accuracy from 40.7% to 36.2%, as the model must handle the full schema with redundant tables and columns. This highlights the Selector's importance in filtering irrelevant schema elements and simplifying prompts.Eliminating the Direct Generator further reduces accuracy to 34.8%, since the refinement process lacks a strong starting point. By providing an initial candidate, the Direct Generator decreases the search depth required by the MCTS-Refiner.The MCTS-Refiner proves most critical: without it, accuracy drops sharply to 16.8%, close to single-shot generation. This validates the necessity of iterative, feedback-driven refinement for correcting syntax and semantic errors.Finally, removing the Prefix-Cache has minimal impact on accuracy (40.7% $\rightarrow$ 40.4%) but significantly decrease the latency, demonstrating its role in improving efficiency without sacrificing performance.Overall, these findings show that each component contributes to an efficient and accurate Text-to-SQL process, enabling a lightweight model like Qwen-1.5B to deliver competitive performance under resource constraints.

| Setting | Simp. | Mod. | Chall. | All |
|---|---|---|---|---|
| Full Pipeline | 46.5 | 35.0 | 22.8 | **40.7** |
| w/o Selector | 42.1 | 30.1 | 18.1 | 36.2 |
| w/o Direct Generator | 39.9 | 29.8 | 17.5 | 34.8 |
| w/o MCTS-Refiner | 19.8 | 13.5 | 8.2 | 16.8 |
| w/o Prefix-Cache | 46.4 | 35.0 | 22.8 | 40.4 |
| Single-shot | 17.2 | 12.1 | 7.1 | 14.7 |

Table 5: Ablation study on BIRD dev set using Qwen-1.5B as base model.

## Error Analysis

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 a 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. The rate of syntax errors and errors in table or database selection has significantly decreased to 13%, demonstrating the importance of our approach in addressing common pattern problems. We observe a change in error types during refinement: while syntax and simple field errors decrease rapidly, remaining failures are dominated by deep semantic or schema-linking issues. Specifically, MCTS is effective at correcting errors such as spelling mistakes, operator errors . However, it still struggles with errors caused by ambiguous natural language queries as well as complex multi-table joins involving foreign keys.

## Conclusion

In conclusion, we propose **MCTS-SQL**, a novel framework to enhance Text-to-SQL performance of lightweight models. Our approach employs Monte Carlo Tree Search for iterative SQL refinement with three modules: Selector, Direct Generator, and MCTS-Refiner. To reduce the overhead of multi-step refinement, we design a token-level **prefix-cache mechanism**, which significantly improves inference efficiency. Experiments on SPIDER and BIRD show that, even with a 1.5B model, MCTS-SQL outperforms ChatGPT-3.5, and achieves competitive results with Gemini 2.5. These findings demonstrate that MCTS-SQL, together with prefix-cache, provides a practical and efficient solution for real-world applications under resource constraints. Overall, MCTS offers an approach that empowers lightweight models with stronger capabilities, providing an excellent baseline for other complex cross-domain research.

## References


Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Alp Caferoğlu, H.; and Ulusoy, Ö. 2024. E-SQL: Direct Schema Linking via Question Enrichment in Text-to-SQL. *arXiv e-prints*, arXiv–2409.

Anil, R.; Dai, A. M.; Firat, O.; Johnson, M.; and Lepikhin, D. 2023. PaLM 2 Technical Report. *arXiv preprint arXiv:2305.10403*.

Chen, G.; Liao, M.; Li, C.; and Fan, K. 2024. AlphaMath Almost Zero: process Supervision without process. *arXiv preprint arXiv:2405.03553*.

Di Zhang, X. H.; Zhou, D.; Li, Y.; and Ouyang, W. 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.

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. *arXiv preprint arXiv:2308.15363*.

Gao, Y.; Liu, Y.; Li, X.; Shi, X.; Zhu, Y.; Wang, Y.; Li, S.; Li, W.; Hong, Y.; Luo, Z.; et al. 2024. Xiyan-sql: A multi-generator ensemble framework for text-to-sql. *arXiv e-prints*, arXiv–2411.

Guo, T.; and Gao, H. 2019. Content enhanced bert-based text-to-sql generation. *arXiv preprint arXiv:1910.07179*.

Hammami, L.; Paglialonga, A.; Pruneri, G.; Torresani, M.; Sant, M.; Bono, C.; Caiani, E. G.; and Baili, P. 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.

Hui, B.; Yang, J.; Cui, Z.; Yang, J.; Liu, D.; Zhang, L.; Liu, T.; Zhang, J.; Yu, B.; Lu, K.; et al. 2024. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Kang, N.; Singh, B.; Afzal, Z.; van Mulligen, E. M.; and Kors, J. A. 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.

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

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

Lee, D.; Park, C.; Kim, J.; and Park, H. 2024. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. *arXiv preprint arXiv:2405.07467*.

Li, A.; Han, C.; Guo, T.; Li, H.; and Li, B. 2023a. General Method for Solving Four Types of SAT Problems. *arXiv preprint arXiv:2312.16423*.

Li, B.; Zhang, J.; Fan, J.; Xu, Y.; Chen, C.; Tang, N.; and Luo, Y. 2025. Alpha-sql: Zero-shot text-to-sql using monte carlo tree search. *arXiv preprint arXiv:2502.17248*.

Li, J.; Hui, B.; Qu, G.; Yang, J.; Li, B.; Li, B.; Wang, B.; Qin, B.; Geng, R.; Huo, N.; et al. 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.

Liu, X.; Shen, S.; Li, B.; Ma, P.; Jiang, R.; Zhang, Y.; Fan, J.; Li, G.; Tang, N.; and Luo, Y. 2024. A Survey of NL2SQL with Large Language Models: Where are we, and where are we going? *arXiv preprint arXiv:2408.05109*.

Lyu, S.; Luo, H.; Li, R.; Ou, Z.; Sun, J.; Qin, Y.; Shang, X.; Song, M.; and Zhu, Y. 2025. SQL-o1: A Self-Reward Heuristic Dynamic Search Method for Text-to-SQL. *arXiv preprint arXiv:2502.11741*.

Pitanov, Y.; Skrynnik, A.; Andreychuk, A.; Yakovlev, K.; and Panov, A. 2023. Monte-Carlo Tree Search for Multi-agent Pathfinding: Preliminary Results. In *International Conference on Hybrid Artificial Intelligence Systems*, 649–660. Springer.

Pourreza, M.; Li, H.; Sun, R.; Chung, Y.; Talaei, S.; Kakkar, G. T.; Gan, Y.; Saberi, A.; Ozcan, F.; and Arik, S. O. 2024. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. *arXiv preprint arXiv:2410.01943*.

Shi, L.; Tang, Z.; Zhang, N.; Zhang, X.; and Yang, Z. 2024. A survey on employing large language models for text-to-sql tasks. *arXiv preprint arXiv:2407.15186*.

Shkapenyuk, V.; Srivastava, D.; Johnson, T.; and Ghane, P. 2025. Automatic Metadata Extraction for Text-to-SQL. *arXiv preprint arXiv:2505.19988*.

Talaei, S.; Pourreza, M.; Chang, Y.-C.; Mirhoseini, A.; and Saberi, A. 2024. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*.

Team, G.; Anil, R.; Borgeaud, S.; Alayrac, J.-B.; Yu, J.; Soricut, R.; Schalkwyk, J.; Dai, A. M.; Hauth, A.; Millican, K.; et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Wang, B.; Ren, C.; Yang, J.; Liang, X.; Bai, J.; Chai, L.; Yan, Z.; Zhang, Q.-W.; Yin, D.; Sun, X.; et al. 2024. Mac-sql: A multi-agent collaborative framework for text-to-sql. *arXiv preprint arXiv:2312.11242*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837.

Xu, H. 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*.

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. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In Riloff, E.; Chiang, D.; Hockenmaier, J.; and Tsujii, J., eds., *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 3911–3921. Brussels, Belgium: Association for Computational Linguistics.

Zhong, R.; Yu, T.; and Klein, D. 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)*, 396–411.

Zhong, V.; Xiong, C.; and Socher, R. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR*, abs/1709.00103.